MySQL 5.0 Reference Manual

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 5.0 through 5.0.96.

End of Product Lifecycle. Active development for MySQL Database Server version 5.0 has ended. Oracle offers various support offerings which may be of interest. For details and more information, see the MySQL section of the Lifetime Support Policy for Oracle Technology Products (http://www.oracle.com/us/support/lifetime-support/index.html). Please consider upgrading to a recent version.

MySQL 5.0 features. This manual describes features that are not included in every edition of MySQL 5.0; such features may not be included in the edition of MySQL 5.0 licensed to you. If you have any questions about the features included in your edition of MySQL 5.0, refer to your MySQL 5.0 license agreement or contact your Oracle representative.

For notes detailing the changes in each release, see the MySQL 5.0 Release Notes.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Licensing information. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL 5.0, see this document for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL 5.0, see this document for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Document generated on: 2016-05-11 (revision: 47682)

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Preface and Legal Notices

This is the Reference Manual for the MySQL Database System, version 5.0, through release 5.0.96. Differences between minor versions of MySQL 5.0 are noted in the present text with reference to release numbers (5.0.*x*). For license information, see the Legal Notices.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.0 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 3.23, 4.0, 4.1 Reference Manual* covers the 4.1 series of MySQL software releases.

If you are using MySQL 5.1, please refer to the *MySQL 5.1 Reference Manual*.

Licensing information. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL 5.0, see this document for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL 5.0, see this document for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Legal Notices

Copyright © 1997, 2016, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD,

This documentation is for an older version. If you're

Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

This documentation is for an older version. If you're

Chapter 1 General Information

Table of Contents

End of Product Lifecycle. Active development for MySQL Database Server version 5.0 has ended. Oracle offers various support offerings which may be of interest. For details and more information, see the MySQL section of the Lifetime Support Policy for Oracle Technology Products (http://www.oracle.com/us/support/lifetime-support/index.html). Please consider upgrading to a recent version.

The MySQL™ software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used by Customer without Oracle's express written authorization. Other names may be trademarks of their respective owners.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (http://www.fsf.org/licenses/) or can purchase a standard commercial license from Oracle. See http://www.mysql.com/company/legal/licensing/ for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion of MySQL Database Server capabilities, see Section 1.3.2, “The Main Features of MySQL”.
- For an overview of new MySQL features, see Section 1.4, “What Is New in MySQL 5.0”. For information about the changes in each version, see the Release Notes.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*. For information about upgrading MySQL, see Section 2.19.1, “Upgrading MySQL”.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3, *Tutorial*.
- For information about configuring and administering MySQL Server, see Chapter 5, *MySQL Server Administration*.
- For information about security in MySQL, see Chapter 6, *Security*.
- For information about setting up replication servers, see Chapter 16, *Replication*.
- For information about MySQL Enterprise, the commercial MySQL release with advanced features and management tools, see Chapter 22, *MySQL Enterprise Edition*.
- For answers to a number of questions that are often asked concerning the MySQL Database Server and its capabilities, see Appendix A, *MySQL 5.0 Frequently Asked Questions*.
- For a history of new features and bugfixes, see the Release Notes.

Important

To report problems or bugs, please use the instructions at Section 1.7, “How to Report Bugs or Problems”. If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support.

1.1 About This Manual

This is the Reference Manual for the MySQL Database System, version 5.0, through release 5.0.96. Differences between minor versions of MySQL 5.0 are noted in the present text with reference to release numbers (5.0.*x*). For license information, see the Legal Notices.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.0 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 3.23, 4.0, 4.1 Reference Manual* covers the 4.1 series of MySQL software releases.

If you are using MySQL 5.1, please refer to the *MySQL 5.1 Reference Manual*.

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at http://dev.mysql.com/doc/. Other formats also are available there, including HTML, PDF, and Windows CHM versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see http://docbook.org/

If you have questions about using MySQL, you can ask them using our mailing lists or forums. See Section 1.6.1, “MySQL Mailing Lists”, and Section 1.6.2, “MySQL Community Support at the MySQL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Forums". If you have suggestions concerning additions or corrections to the manual itself, please send them to the http://www.mysql.com/company/contact/.

This manual was originally written by David Axmark and Michael "Monty" Widenius. It is maintained by the MySQL Documentation Team, consisting of Chris Cole, Paul DuBois, Edward Gilmore, Stefan Hinz, David Moss, Philip Olson, Daniel Price, Daniel So, and Jon Stephens.

1.2 Typographical and Syntax Conventions

This manual uses certain typographical conventions:

- `Text in this style` is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: "To reload the grant tables, use the `FLUSH PRIVILEGES` statement."
- **`Text in this style`** indicates input that you type in examples.
- `Text in this style` indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command-line client program) and `mysqld` (the MySQL server executable).
- *`Text in this style`* is used for variable input for which you should substitute a value of your own choosing.
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.
- `Text in this style` is used to indicate a program option that affects how the program is executed, or that supplies information that is needed for the program to function in a certain way. *Example*: "The `--host` option (short form `-h`) tells the `mysql` client program the hostname or IP address of the MySQL server that it should connect to".
- File names and directory names are written like this: "The global `my.cnf` file is located in the `/etc` directory."
- Character sequences are written like this: "To specify a wildcard, use the '`%`' character."

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, `root-shell>` is similar but should be executed as `root`, and `mysql>` indicates a statement that you execute from the `mysql` client program:

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

In some areas different systems may be distinguished from each other to show that commands should be executed in two different environments. For example, while working with replication the commands might be prefixed with `master` and `slave`:

```
master> type a mysql command on the replication master here
slave> type a mysql command on the replication slave here
```

The "shell" is your command interpreter. On Unix, this is typically a program such as `sh`, `csh`, or `bash`. On Windows, the equivalent program is `command.com` or `cmd.exe`, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *`db_name`*, *`tbl_name`*, and *`col_name`*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (“`[`” and “`]`”) indicate optional words or clauses. For example, in the following statement, `IF EXISTS` is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (“`|`”). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (“`[`” and “`]`”):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (“`{`” and “`}`”):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (`...`) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, `SELECT ... INTO OUTFILE` is shorthand for the form of `SELECT` statement that has an `INTO OUTFILE` clause following other parts of the statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *`reset_option`* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the `CC` environment variable and run the `configure` command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using `csh` or `tcsh`, you must issue commands somewhat differently:

```
shell> setenv CC gcc
shell> ./configure
```

1.3 Overview of the MySQL Database Management System

1.3.1 What is MySQL?

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

The MySQL Web site (http://www.mysql.com/) provides the latest information about MySQL software.

- **MySQL is a database management system.**

  A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- **MySQL databases are relational.**

  A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required or optional, and “pointers” between different tables. The database enforces these rules, so that with a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

  The SQL part of “MySQL” stands for “Structured Query Language”. SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

  SQL is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- **MySQL software is Open Source.**

  Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), http://www.fsf.org/licenses/, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (http://www.mysql.com/company/legal/licensing/).

- **The MySQL Database Server is very fast, reliable, scalable, and easy to use.**

  If that is what you are looking for, you should give it a try. MySQL Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention. If you dedicate an entire machine to MySQL, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MySQL can also scale up to clusters of machines, networked together.

  You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See Section 8.13.2, “The MySQL Benchmark Suite”.

  MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- **MySQL Server works in client/server or embedded systems.**

  The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

  We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- **A large amount of contributed MySQL software is available.**

  MySQL Server has a practical set of features developed in close cooperation with our users. It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

1.3.2 The Main Features of MySQL

This section describes some of the important characteristics of the MySQL Database Software. In most respects, the roadmap applies to all versions of MySQL. For information about features as they are introduced into MySQL on a series-specific basis, see the “In a Nutshell” section of the appropriate Manual:

- MySQL 5.7: What Is New in MySQL 5.7
- MySQL 5.6: What Is New in MySQL 5.6
- MySQL 5.5: What Is New in MySQL 5.5
- MySQL 5.1: What Is New in MySQL 5.1

Internals and Portability

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See http://www.mysql.com/support/supportedplatforms/database.html.
- For portability, uses `CMake` in MySQL 5.5 and up. Previous series use GNU Automake, Autoconf, and Libtool.
- Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (http://developer.kde.org/~sewardj/).
- Uses multi-layered server design with independent modules.
- Designed to be fully multi-threaded using kernel threads, to easily use multiple CPUs if they are available.
- Provides transactional and nontransactional storage engines.
- Uses very fast B-tree disk tables (`MyISAM`) with index compression.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Designed to make it relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.
- Uses a very fast thread-based memory allocation system.
- Executes very fast joins using an optimized nested-loop join.
- Implements in-memory hash tables, which are used as temporary tables.
- Implements SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET`, `ENUM`, and OpenGIS spatial types. See Chapter 11, *Data Types*.
- Fixed-length and variable-length string types.

Statements and Functions

- Full operator and function support in the `SELECT` list and `WHERE` clause of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
    -> FROM citizen
    -> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL `GROUP BY` and `ORDER BY` clauses. Support for group functions (`COUNT()`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()`, and `GROUP_CONCAT()`).
- Support for `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- Support for `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` to return the number of rows that were changed (affected), or to return the number of rows matched instead by setting a flag when connecting to the server.
- Support for MySQL-specific `SHOW` statements that retrieve information about databases, storage engines, tables, and indexes. Support for the `INFORMATION_SCHEMA` database, implemented according to standard SQL.
- An `EXPLAIN` statement to show how the optimizer resolves a query.
- Independence of function names from table or column names. For example, `ABS` is a valid column name. The only restriction is that for a function call, no spaces are permitted between the function name and the "`(`" that follows it. See Section 9.3, "Keywords and Reserved Words".
- You can refer to tables from different databases in the same statement.

Security

- A privilege and password system that is very flexible and secure, and that enables host-based verification.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Password security by encryption of all password traffic when you connect to a server.

Scalability and Limits

- Support for large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 200,000 tables and about 5,000,000,000 rows.
- Support for up to 64 indexes per table. Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 767 bytes for `InnoDB` tables, or 1000 for `MyISAM`. An index may use a prefix of a column for `CHAR`, `VARCHAR`, `BLOB`, or `TEXT` column types.

Connectivity

- Clients can connect to MySQL Server using several protocols:
  - Clients can connect using TCP/IP sockets on any platform.
  - On Windows systems, clients can connect using named pipes if the server is started with the `--enable-named-pipe` option. Windows servers also support shared-memory connections if started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.
  - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, enabling MySQL clients to be written in many languages. See Chapter 20, *Connectors and APIs*.
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See MySQL Connector/ODBC Developer Guide.
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See MySQL Connector/J 5.1 Developer Guide.
- MySQL Connector/Net enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/Net is a fully managed ADO.NET driver written in 100% pure C#. See MySQL Connector/Net Developer Guide.

Localization

- The server can provide error messages to clients in many languages. See Section 10.2, “Setting the Error Message Language”.
- Full support for several different character sets, including `latin1` (cp1252), `german`, `big5`, `ujis`, several Unicode character sets, and more. For example, the Scandinavian characters “`å`”, “`ä`” and “`ö`” are permitted in table and column names.
- All data is saved in the chosen character set.
- Sorting and comparisons are done according to the chosen character set and collation (using `latin1` and Swedish collation by default). It is possible to change this when the MySQL server is started. To see

This documentation is for an older version. If you're
This documentation is for an older version. If you're

an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.

- The server time zone can be changed dynamically, and individual clients can specify their own time zone. See Section 10.6, “MySQL Server Time Zone Support”.

Clients and Tools

- MySQL includes several client and utility programs. These include both command-line programs such as `mysqldump` and `mysqladmin`, and graphical programs such as MySQL Workbench.
- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the `mysqlcheck` client. MySQL also includes `myisamchk`, a very fast command-line utility for performing these operations on `MyISAM` tables. See Chapter 4, *MySQL Programs*.
- MySQL programs can be invoked with the `--help` or `-?` option to obtain online assistance.

1.3.3 History of MySQL

We started out with the intention of using the `mSQL` database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to enable third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

MySQL is named after co-founder Monty Widenius's daughter, My.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4 What Is New in MySQL 5.0

The following features are implemented in MySQL 5.0:

- **Information Schema.** The introduction of the `INFORMATION_SCHEMA` database in MySQL 5.0 provided a standards-compliant means for accessing the MySQL Server's metadata; that is, data about the databases (schemas) on the server and the objects which they contain. See Chapter 19, *INFORMATION_SCHEMA Tables*.
- **Instance Manager.** Can be used to start and stop the MySQL Server, even from a remote host. See Section 4.6.10, “`mysqlmanager` — The MySQL Instance Manager”.
- **Precision Math.** MySQL 5.0 introduced stricter criteria for acceptance or rejection of data, and implemented a new library for fixed-point arithmetic. These contributed to a much higher degree of accuracy for mathematical operations and greater control over invalid values. See Section 12.17, “Precision Math”.
- **Storage Engines.** New storage engines were added and performance of others was improved.
  - New storage engines in MySQL 5.0 include `ARCHIVE` and `FEDERATED`. See Section 14.8, “The ARCHIVE Storage Engine”, and Section 14.7, “The FEDERATED Storage Engine”.
  - Performance Improvements in the `InnoDB` Storage Engine:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- New compact storage format which can save up to 20% of the disk space required in previous MySQL/InnoDB versions.
- Faster recovery from a failed or aborted ALTER TABLE.
- Faster implementation of TRUNCATE TABLE.

(See Section 14.2, “The InnoDB Storage Engine”.)

- Performance Improvements in the NDBCLUSTER Storage Engine:
  - Faster handling of queries that use IN and BETWEEN.
  - Condition pushdown: In cases involving the comparison of an unindexed column with a constant, this condition is “pushed down” to the cluster where it is evaluated in all partitions simultaneously, eliminating the need to send nonmatching records over the network. This can make such queries 10 to 100 times faster than in MySQL 4.1 Cluster.

    See Section 13.8.2, “EXPLAIN Syntax”, for more information.

  (See Chapter 17, *MySQL Cluster*.)

- **Stored Routines.** MySQL 5.0 added support for stored procedures and stored functions. See Section 18.2, “Using Stored Routines (Procedures and Functions)”.
- **Triggers.** MySQL 5.0 added limited support for triggers. See Section 18.3, “Using Triggers”.
- **Views.** MySQL 5.0 added support for named, updatable views. See Section 18.4, “Using Views”.
- **Cursors.** Elementary support for server-side cursors. For information about using cursors within stored routines, see Section 13.6.6, “Cursors”. For information about using cursors from within the C API, see Section 20.6.11.3, “mysql_stmt_attr_set()”.
- **Strict Mode and Standard Error Handling.** MySQL 5.0 added a strict mode where by it follows standard SQL in a number of ways in which it did not previously. Support for standard SQLSTATE error messages was also implemented. See Section 5.1.7, “Server SQL Modes”.
- **VARCHAR Data Type.** The effective maximum length of a VARCHAR column was increased to 65,535 bytes, and stripping of trailing whitespace was eliminated. VARCHAR in MySQL 5.0 is more efficient than in previous versions, due to the elimination of the old (and nonstandard) removal of trailing spaces during retrieval. (The actual maximum length of a VARCHAR is determined by the maximum row size and the character set you use. The maximum *effective* column length is subject to a row size of 65,535 bytes, which is shared among all columns.) See Section 11.4, “String Types”.
- **BIT Data Type.** A true BIT column type is available that can be used to store numbers in binary notation. This type is much more efficient for storage and retrieval of Boolean values than the workarounds required in MySQL in versions previous to 5.0. See Section 11.1.1, “Numeric Type Overview”.
- **Optimizer enhancements.** Several optimizer improvements were made to improve the speed of certain types of queries and in the handling of certain types. These include:
  - MySQL 5.0 introduces a new “greedy” optimizer which can greatly reduce the time required to arrive at a query execution plan. This is particularly noticeable where several tables are to be joined and no good join keys can otherwise be found. Without the greedy optimizer, the complexity of the search for an execution plan is calculated as *N*!, where *N* is the number of tables to be joined. The greedy

This documentation is for an older version. If you're
This documentation is for an older version. If you're

optimizer reduces this to `N!/(D-1)!`, where `D` is the depth of the search. Although the greedy optimizer does not guarantee the best possible of all execution plans (this is currently being worked on), it can reduce the time spent arriving at an execution plan for a join involving a great many tables—30, 40, or more—by a factor of as much as 1,000. This should eliminate most if not all situations where users thought that the optimizer had hung when trying to perform joins across many tables.

- Use of the *Index Merge* method to obtain better optimization of `AND` and `OR` relations over different keys. (Previously, these were optimized only where both relations in the `WHERE` clause involved the same key.) This also applies to other one-to-one comparison operators (`>`, `<`, and so on), including `=` and the `IN` operator. This means that MySQL can use multiple indexes in retrieving results for conditions such as `WHERE key1 > 4 OR key2 < 7` and even combinations of conditions such as `WHERE (key1 > 4 OR key2 < 7) AND (key3 >= 10 OR key4 = 1)`. See Section 8.2.1.4, “Index Merge Optimization”.

- A new equality detector finds and optimizes “hidden” equalities in joins. For example, a `WHERE` clause such as

  ```
  t1.c1=t2.c2 AND t2.c2=t3.c3 AND t1.c1 < 5
  ```

  implies these other conditions

  ```
  t1.c1=t3.c3 AND t2.c2 < 5 AND t3.c3 < 5
  ```

  These optimizations can be applied with any combination of `AND` and `OR` operators. See Section 8.2.1.9, “Nested Join Optimization”, and Section 8.2.1.10, “Outer Join Simplification”.

- Optimization of `NOT IN` and `NOT BETWEEN` relations, reducing or eliminating table scans for queries making use of them by mean of range analysis. The performance of MySQL with regard to these relations now matches its performance with regard to `IN` and `BETWEEN`.

- **XA Transactions.** MySQL 5.0 supports XA (distributed) transactions. See Section 13.3.7, “XA Transactions”.

1.5 MySQL Development History

This section describes the general MySQL development history, including major features implemented in or planned for various MySQL releases. The following sections provide information for each release series.

The current production release series is MySQL 5.1, which was declared stable for production use as of MySQL 5.1.30, released in November 2008. The previous production release series was MySQL 5.0, which was declared stable for production use as of MySQL 5.0.15, released in October 2005. “General Availability status” means that future 5.1 and 5.0 development is limited only to bugfixes. For the older MySQL 4.1, 4.0, and 3.23 series, only critical bugfixes are made.

Before upgrading from one release series to the next, please see the notes in Section 2.19.1, “Upgrading MySQL”.

The most requested features and the versions in which they were implemented are summarized in the following table.

Feature	MySQL Series
Unions	4.0
Subqueries	4.1

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Feature	MySQL Series
R-trees	4.1 (for the `MyISAM` storage engine)
Stored procedures and functions	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Triggers	5.0 and 5.1
Event scheduler	5.1
Partitioning	5.1
Pluggable storage engine API	5.1
Plugin API	5.1
InnoDB Plugin	5.1
Row-based replication	5.1
Server log tables	5.1

1.6 MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as the MySQL mailing lists and user forums, and Internet Relay Chat.

1.6.1 MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit http://lists.mysql.com/. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from `lists.mysql.com` to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

To have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- `announce`

  The list for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- `mysql`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- `bugs`

  The list for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See Section 1.7, “How to Report Bugs or Problems”.

- `internals`

  The list for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- `mysqldoc`

  The list for people who work on the MySQL documentation.

- `benchmarks`

  The list for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, file system, disk system, and so on.

- `packagers`

  The list for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- `java`

  The list for discussions about the MySQL server and Java. It is mostly used to discuss JDBC drivers such as MySQL Connector/J.

- `win32`

  The list for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- `myodbc`

  The list for all topics concerning connecting to the MySQL server with ODBC.

- `gui-tools`

  The list for all topics concerning MySQL graphical user interface tools such as MySQL Workbench.

- `cluster`

  The list for discussion of MySQL Cluster.

- `dotnet`

  The list for discussion of the MySQL server and the .NET platform. It is mostly related to MySQL Connector/Net.

- `plusplus`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The list for all topics concerning programming with the C++ API for MySQL.

- `perl`

  The list for all topics concerning Perl support for MySQL with `DBD::mysql`.

If you're unable to get an answer to your questions from a MySQL mailing list or forum, one option is to purchase support from Oracle. This puts you in direct contact with MySQL developers.

The following MySQL mailing lists are in languages other than English. These lists are not operated by Oracle.

- `<mysql-france-subscribe@yahoogroups.com>`

  A French mailing list.

- `<list@tinc.net>`

  A Korean mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

- `<mysql-de-request@lists.4t2.com>`

  A German mailing list. To subscribe, email `subscribe mysql-de your@email.address` to this list. You can find information about this mailing list at http://www.4t2.com/mysql/.

- `<mysql-br-request@listas.linkway.com.br>`

  A Portuguese mailing list. To subscribe, email `subscribe mysql-br your@email.address` to this list.

- `<mysql-alta@elistas.net>`

  A Spanish mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

1.6.1.1 Guidelines for Using the Mailing Lists

Please do not post mail messages from your browser with HTML mode turned on. Many users do not read mail with a browser.

When you answer a question sent to a mailing list, if you consider your answer to have broad interest, you may want to post it to the list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply. Do not feel obliged to quote the entire original message.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.6.2 MySQL Community Support at the MySQL Forums

The forums at http://forums.mysql.com are an important community resource. Many forums are available, grouped into these general categories:

- Migration

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology
- SQL Standards
- Business

1.6.3 MySQL Community Support on Internet Relay Chat (IRC)

In addition to the various MySQL mailing lists and forums, you can find experienced community people on Internet Relay Chat (IRC). These are the best networks/channels currently known to us:

freenode (see http://www.freenode.net/ for servers)

- `#mysql` is primarily for MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl, or C in combination with MySQL are also common.
- `#workbench` is primarily for MySQL Workbench related questions and thoughts, and it is also a good place to meet the MySQL Workbench developers.

If you are looking for IRC client software to connect to an IRC network, take a look at `xChat` (http://www.xchat.org/). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at http://www.silverex.org/download/).

1.6.4 MySQL Enterprise

Oracle offers technical support in the form of MySQL Enterprise. For organizations that rely on the MySQL DBMS for business-critical production applications, MySQL Enterprise is a commercial subscription offering which includes:

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Updates and Quarterly Service Packs
- MySQL Knowledge Base
- 24x7 Technical and Consultative Support

MySQL Enterprise is available in multiple tiers, giving you the flexibility to choose the level of service that best matches your needs. For more information, see MySQL Enterprise.

1.7 How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Start by searching the MySQL online manual at http://dev.mysql.com/doc/. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. In addition, the release notes accompanying the manual can be particularly useful since it is quite possible that a newer version contains a solution to your problem. The release notes are available at the location just given for the manual.

- If you get a parse error for an SQL statement, please check your syntax closely. If you cannot find something wrong with it, it is extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement.

  If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.

- For solutions to some common problems, see Section B.5, “Problems and Common Errors”.

- Search the bugs database at http://bugs.mysql.com/ to see whether the bug has been reported and fixed.

- Search the MySQL mailing list archives at http://lists.mysql.com/. See Section 1.6.1, “MySQL Mailing Lists”.

- You can also use http://www.mysql.com/search/ to search all the Web pages (including the manual) that are located at the MySQL Web site.

If you cannot find an answer in the manual, the bugs database, or the mailing list archives, check with your local MySQL expert. If you still cannot find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

Bugs posted in the bugs database at http://bugs.mysql.com/ that are corrected for a given release are noted in the release notes.

If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support at http://support.oracle.com/.

To discuss problems with other users, you can use one of the MySQL mailing lists. Section 1.6.1, “MySQL Mailing Lists”.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you do not waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The best reports are those that include a full example showing how to reproduce the bug or problem. See Section 21.3, “Debugging and Porting MySQL”.

Remember that it is possible for us to respond to a report containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See How to Report Connector/ODBC Problems or Bugs.

If your report includes long query output lines from test cases that you run with the `mysql` command-line tool, you can make the output more readable by using the `--vertical` option or the `\G` statement terminator. The `EXPLAIN SELECT` example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.7.10). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If `mysqld` died, you should also report the statement that crashed `mysqld`. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` crashes. See Section 21.3, “Debugging and Porting MySQL”.

- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE db_name.tbl_name` statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.

- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` system variable. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

  ```
  SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.ROUTINES;
  ```

  For triggers, you can use this statement:

  ```
  SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.TRIGGERS;
  ```

- For performance-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE tbl_name` for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

  The following is an example of a very good bug report. The statements are run using the `mysql` command-line tool. Note the use of the `\G` statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
         <output from SHOW COLUMNS>
  mysql> EXPLAIN SELECT ...\G
         <output from EXPLAIN>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
         <A short version of the output from SELECT,
         including the time taken to run the query>
  mysql> SHOW STATUS;
         <output from SHOW STATUS>
  ```

- If a bug or problem occurs while running `mysqld`, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

  If you cannot provide a script, you should at least include the output from `mysqladmin variables extended-status processlist` in your report to provide some information on how your system is performing.

- If you cannot produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using `mysqldump` and create a `README` file that describes your problem. Create a compressed archive of your files using `tar` and `gzip` or `zip`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

After you initiate a bug report for our bugs database at http://bugs.mysql.com/, click the Files tab in the bug report for instructions on uploading the archive to the bugs database.

- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.

- When you provide an example of the problem, it is better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you do not want to be visible to others in the bug report, you can upload it using the Files tab as previously described. If the information is really top secret and you do not want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.

- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the `mysqld` server, as well as the options that you use to run any MySQL client programs. The options to programs such as `mysqld` and `mysql`, and to the `configure` script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the `DBI` and `DBD::mysql` modules, include the version numbers for Perl, `DBI`, and `DBD::mysql`.

- If your question is related to the privilege system, please include the output of `mysqladmin reload`, and all the error messages you get when trying to connect. When you test your privileges, you should execute `mysqladmin reload version` and try to connect with the program that gives you trouble.

- If you have a patch for a bug, do include it. But do not assume that the patch is all we need, or that we can use it, if you do not provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we cannot use it.

  If we cannot verify the exact purpose of the patch, we will not use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch will not work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team cannot guess such things without first using a debugger to determine the real cause of a bug.

- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.

- If your data appears corrupt or you get errors when you access a particular table, first check your tables with `CHECK TABLE`. If that statement reports any errors:

  - The `InnoDB` crash recovery mechanism handles cleanup when the server is restarted after being killed, so in typical operation there is no need to “repair” tables. If you encounter an error with `InnoDB` tables, restart the server and see whether the problem persists, or whether the error affected only cached data in memory. If data is corrupted on disk, consider restarting with the `innodb_force_recovery` option enabled so that you can dump the affected tables.

  - For non-transactional tables, try to repair them with `REPAIR TABLE` or with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you are running Windows, please verify the value of `lower_case_table_names` using the `SHOW VARIABLES LIKE 'lower_case_table_names'` statement. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in Section 9.2.2, “Identifier Case Sensitivity”.

- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See Section 5.4.1, “The Error Log”. Please include any relevant information from this file in your bug report. Normally `mysqld` should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of `mysqld` dying, it is much easier for us to provide you with a fix for the problem. See Section B.5.1, “How to Determine What Is Causing a Problem”.
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software are thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See Section 2.4.2, “Choosing Which MySQL Distribution to Install”.

1.8 MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, “SQL:2003” refers to the standard released in 2003, and “SQL:2008” refers to the most recent version of the standard, released in 2008. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface is an example of this strategy. See Section 13.2.4, “HANDLER Syntax”.

We continue to support transactional and nontransactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

We are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

MySQL supports ODBC levels 0 to 3.51.

MySQL supports high-availability database clustering using the `NDBCLUSTER` storage engine. See Chapter 17, *MySQL Cluster*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

XML support is to be implemented in a future version of the database server.

Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

For more information on setting the SQL mode, see Section 5.1.7, “Server SQL Modes”.

Running MySQL in ANSI Mode

To run MySQL Server in ANSI mode, start `mysqld` with the `--ansi` option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

To achieve the same effect at runtime, execute these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` system variable to `'ANSI'` enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

Running the server in ANSI mode with `--ansi` is not quite the same as setting the SQL mode to `'ANSI'` because the `--ansi` option also sets the transaction isolation level.

See Section 5.1.3, “Server Command Options”.

1.8.1 MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably won't find in other SQL DBMSs. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you add a version number after the "`!`" character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

  MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to file names in the database directory. This has a few implications:

  - Database and table names are case sensitive in MySQL Server on operating systems that have case-sensitive file names (such as most Unix systems). See Section 9.2.2, "Identifier Case Sensitivity".
  - You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` storage engine. For example, it is possible to rename a `MyISAM` table by renaming the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

  Database and table names cannot contain path name separator characters ("`/`", "`\`").

- General language syntax

  - By default, strings can be enclosed by either "`"`" or "`'`", not just by "`'`". (If the `ANSI_QUOTES` SQL mode is enabled, strings can be enclosed only by "`'`" and the server interprets strings enclosed by "`"`" as identifiers.)
  - "`\`" is the escape character in strings.
  - In SQL statements, you can access tables from different databases with the *`db_name.tbl_name`* syntax. Some SQL servers provide the same functionality but call this `User space`. MySQL Server doesn't support tablespaces such as used in statements like this: `CREATE TABLE ralph.my_table ... IN my_tablespace`.

- SQL statement syntax

  - The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
  - The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See Section 13.1.6, "CREATE DATABASE Syntax", Section 13.1.13, "DROP DATABASE Syntax", and Section 13.1.1, "ALTER DATABASE Syntax".
  - The `DO` statement.
  - `EXPLAIN SELECT` to obtain a description of how tables are processed by the query optimizer.
  - The `FLUSH` and `RESET` statements.
  - The `SET` statement. See Section 13.7.4, "SET Syntax".
  - The `SHOW` statement. See Section 13.7.5, "SHOW Syntax". The information produced by many of the MySQL-specific `SHOW` statements can be obtained in more standard fashion by using `SELECT` to query `INFORMATION_SCHEMA`. See Chapter 19, *INFORMATION_SCHEMA Tables*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Use of `LOAD DATA INFILE`. In many cases, this syntax is compatible with Oracle's `LOAD DATA INFILE`. See Section 13.2.6, “LOAD DATA INFILE Syntax”.
- Use of `RENAME TABLE`. See Section 13.1.20, “RENAME TABLE Syntax”.
- Use of `REPLACE` instead of `DELETE` plus `INSERT`. See Section 13.2.7, “REPLACE Syntax”.
- Use of `CHANGE col_name`, `DROP col_name`, or `DROP INDEX`, `IGNORE` or `RENAME` in `ALTER TABLE` statements. Use of multiple `ADD`, `ALTER`, `DROP`, or `CHANGE` clauses in an `ALTER TABLE` statement. See Section 13.1.4, “ALTER TABLE Syntax”.
- Use of index names, indexes on a prefix of a column, and use of `INDEX` or `KEY` in `CREATE TABLE` statements. See Section 13.1.10, “CREATE TABLE Syntax”.
- Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
- Use of `IF EXISTS` with `DROP TABLE` and `DROP DATABASE`.
- The capability of dropping multiple tables with a single `DROP TABLE` statement.
- The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
- `INSERT INTO tbl_name SET col_name = ...` syntax.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
- Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See Section 13.2.8, “SELECT Syntax”.
- Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.
- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See Section 12.16, “GROUP BY (Aggregate) Functions”.
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator. See Section 9.4, “User-Defined Variables”.

- Data types
  - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.
  - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.
- Functions and operators
  - To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
  - MySQL Server understands the `||` and `&&` operators to mean logical OR and AND, as in the C programming language. In MySQL Server, `||` and `OR` are synonyms, as are `&&` and `AND`. Because of this nice syntax, MySQL Server doesn't support the standard SQL `||` operator for string

This documentation is for an older version. If you're
This documentation is for an older version. If you're

concatenation; use `CONCAT()` instead. Because `CONCAT()` takes any number of arguments, it is easy to convert use of the `||` operator to MySQL Server.

- Use of `COUNT(DISTINCT value_list)` where *`value_list`* has more than one element.
- String comparisons are case-insensitive by default, with sort ordering determined by the collation of the current character set, which is `latin1` (cp1252 West European) by default. If you don't like this, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.
- The `%` operator is a synonym for `MOD()`. That is, *`N % M`* is equivalent to `MOD(N,M)`. `%` is supported for C programmers and for compatibility with PostgreSQL.
- The `=`, `<>`, `<=`, `<`, `>=`, `>`, `<<`, `>>`, `<=>`, `AND`, `OR`, or `LIKE` operators may be used in expressions in the output column list (to the left of the `FROM`) in `SELECT` statements. For example:

```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```

- The `LAST_INSERT_ID()` function returns the most recent `AUTO_INCREMENT` value. See Section 12.13, “Information Functions”.
- `LIKE` is permitted on numeric values.
- The `REGEXP` and `NOT REGEXP` extended regular expression operators.
- `CONCAT()` or `CHAR()` with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)
- The `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `MD5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()`, and `WEEKDAY()` functions.
- Use of `TRIM()` to trim substrings. Standard SQL supports removal of single characters only.
- The `GROUP BY` functions `STD()`, `BIT_OR()`, `BIT_AND()`, `BIT_XOR()`, and `GROUP_CONCAT()`. See Section 12.16, “GROUP BY (Aggregate) Functions”.

1.8.2 MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- For `VARCHAR` columns, trailing spaces are removed when the value is stored. (This is fixed in MySQL 5.0.3). See Section B.5.7, “Known Issues in MySQL”.
- In some cases, `CHAR` columns are silently converted to `VARCHAR` columns when you define a table or alter its structure. (This no longer occurs as of MySQL 5.0.3). See Section 13.1.10.4, “Silent Column Specification Changes”.
- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a `REVOKE` statement to revoke privileges for a table. For more information, see Section 13.7.1.5, “REVOKE Syntax”.
- The `CAST()` function does not support cast to `REAL` or `BIGINT`. See Section 12.10, “Cast Functions and Operators”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Standard SQL requires that a `HAVING` clause in a `SELECT` statement be able to refer to columns in the `GROUP BY` clause. This cannot be done before MySQL 5.0.2.

1.8.2.1 SELECT INTO TABLE

MySQL Server doesn't support the `SELECT ... INTO TABLE` Sybase SQL extension. Instead, MySQL Server supports the `INSERT INTO ... SELECT` standard SQL syntax, which is basically the same thing. See Section 13.2.5.1, “INSERT ... SELECT Syntax”. For example:

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use `SELECT ... INTO OUTFILE` or `CREATE TABLE ... SELECT`.

You can use `SELECT ... INTO` with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See Section 13.2.8.1, “SELECT ... INTO Syntax”.

1.8.2.2 UPDATE

If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.8.2.3 Transactions and Atomic Operations

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the `InnoDB` and `BDB` transactional storage engines. `InnoDB` provides *full* ACID compliance. See Chapter 14, *Storage Engines*. For information about `InnoDB` differences from standard SQL with regard to treatment of transaction errors, see Section 14.2.12, “InnoDB Error Handling”.

The other nontransactional storage engines in MySQL Server (such as `MyISAM`) follow a different paradigm for data integrity called “atomic operations.” In transactional terms, `MyISAM` tables effectively always operate in `autocommit = 1` mode. Atomic operations often offer comparable integrity with higher performance.

Because MySQL Server supports both paradigms, you can decide whether your applications are best served by the speed of atomic operations or the use of transactional features. This choice can be made on a per-table basis.

As noted, the tradeoff for transactional versus nontransactional storage engines lies mostly in performance. Transactional tables have significantly higher memory and disk space requirements, and more CPU overhead. On the other hand, transactional storage engines such as `InnoDB` also offer many significant features. MySQL Server's modular design enables the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

But how do you use the features of MySQL Server to maintain rigorous integrity even with the nontransactional `MyISAM` tables, and how do these features compare with the transactional storage engines?

- If your applications are written in a way that is dependent on being able to call `ROLLBACK` rather than `COMMIT` in critical situations, transactions are more convenient. Transactions also ensure that unfinished

This documentation is for an older version. If you're
This documentation is for an older version. If you're

updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

If you use nontransactional tables, MySQL Server in almost all cases enables you to resolve potential problems by including simple checks before updates and by running simple scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. You can normally fix tables perfectly with no data integrity loss just by using the MySQL log or even adding one extra log.

- More often than not, critical transactional updates can be rewritten to be atomic. Generally speaking, all integrity problems that transactions solve can be done with `LOCK TABLES` or atomic updates, ensuring that there are no automatic aborts from the server, which is a common problem with transactional database systems.
- To be safe with MySQL Server, regardless of whether you use transactional tables, you only need to have backups and have binary logging turned on. When that is true, you can recover from any situation that you could with any other transactional database system. It is always good to have backups, regardless of which database system you use.

The transactional paradigm has its advantages and disadvantages. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be necessary, or is necessary. However, even if you are new to the atomic operations paradigm, or more familiar with transactions, do consider the speed benefit that nontransactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In situations where integrity is of highest importance, MySQL Server offers transaction-level reliability and integrity even for nontransactional tables. If you lock tables with `LOCK TABLES`, all updates stall until integrity checks are made. If you obtain a `READ LOCAL` lock (as opposed to a write lock) for a table that enables concurrent inserts at the end of the table, reads are permitted, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With `INSERT DELAYED`, you can write inserts that go into a local queue until the locks are released, without having the client wait for the insert to complete. See Section 8.11.3, “Concurrent Inserts”, and Section 13.2.5.2, “INSERT DELAYED Syntax”.

“Atomic,” in the sense that we mean it, is nothing magical. It only means that you can be sure that while each specific update is running, no other user can interfere with it, and there can never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there are no dirty reads.

Following are some techniques for working with nontransactional tables:

- Loops that need transactions normally can be coded with the help of `LOCK TABLES`, and you don't need cursors to update records on the fly.
- To avoid using `ROLLBACK`, you can employ the following strategy:

  1. Use `LOCK TABLES` to lock all the tables you want to access.
  2. Test the conditions that must be true before performing the update.
  3. Update if the conditions are satisfied.
  4. Use `UNLOCK TABLES` to release your locks.

  This is usually a much faster method than using transactions with possible rollbacks, although not always. The only situation this solution doesn't handle is when someone kills the threads in the middle of an update. In that case, all locks are released but some of the updates may not have been executed.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- You can also use functions to update records in a single operation. You can get a very efficient application by using the following techniques:

  - Modify columns relative to their current value.

  - Update only those columns that actually have changed.

  For example, when we are updating customer information, we update only the customer data that has changed and test only that none of the changed data, or data that depends on the changed data, has changed compared to the original row. The test for changed data is done with the `WHERE` clause in the `UPDATE` statement. If the record wasn't updated, we give the client a message: “Some of the data you have changed has been changed by another user.” Then we show the old row versus the new row in a window so that the user can decide which version of the customer record to use.

  This gives us something that is similar to column locking but is actually even better because we only update some of the columns, using values that are relative to their current values. This means that typical `UPDATE` statements look something like these:

  ```
  UPDATE tablename SET pay_back=pay_back+125;

  UPDATE customer
    SET
      customer_date='current_date',
      address='new address',
      phone='new phone',
      money_owed_to_us=money_owed_to_us-125
    WHERE
      customer_id=id AND address='old address' AND phone='old phone';
  ```

  This is very efficient and works even if another client has changed the values in the `pay_back` or `money_owed_to_us` columns.

- In many cases, users have wanted `LOCK TABLES` or `ROLLBACK` for the purpose of managing unique identifiers. This can be handled much more efficiently without locking or rolling back by using an `AUTO_INCREMENT` column and either the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. See Section 12.13, “Information Functions”, and Section 20.6.7.37, “mysql_insert_id()”.

  You can generally code around the need for row-level locking. Some situations really do need it, and `InnoDB` tables support row-level locking. Otherwise, with `MyISAM` tables, you can use a flag column in the table and do something like the following:

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID;
  ```

  MySQL returns `1` for the number of affected rows if the row was found and `row_flag` wasn't `1` in the original row. You can think of this as though MySQL Server changed the preceding statement to:

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
  ```

1.8.2.4 Foreign Key Differences

MySQL's implementation of foreign keys differs from the SQL standard in the following key respects:

- If there are several rows in the parent table that have the same referenced key value, `InnoDB` acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you

This documentation is for an older version. If you're
This documentation is for an older version. If you're

have defined a `RESTRICT` type constraint, and there is a child row with several parent rows, `InnoDB` does not permit the deletion of any of those parent rows.

`InnoDB` performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

- A `FOREIGN KEY` constraint that references a non-`UNIQUE` key is not standard SQL but rather an `InnoDB` extension.
- If `ON UPDATE CASCADE` or `ON UPDATE SET NULL` recurses to update the *same table* it has previously updated during the same cascade, it acts like `RESTRICT`. This means that you cannot use self-referential `ON UPDATE CASCADE` or `ON UPDATE SET NULL` operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential `ON DELETE SET NULL`, on the other hand, is possible, as is a self-referential `ON DELETE CASCADE`. Cascading operations may not be nested more than 15 levels deep.
- In an SQL statement that inserts, deletes, or updates many rows, foreign key constraints (like unique constraints) are checked row-by-row. When performing foreign key checks, `InnoDB` sets shared row-level locks on child or parent records that it must examine. MySQL checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. This means that it is not possible to delete a row that refers to itself using a foreign key.

For information about how the `InnoDB` storage engine handles foreign keys, see Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”.

1.8.2.5 '--' as the Start of a Comment

Standard SQL uses the C syntax `/* this is a comment */` for comments, and MySQL Server supports this syntax as well. MySQL also support extensions to this syntax that enable MySQL-specific SQL to be embedded in the comment, as described in Section 9.6, “Comment Syntax”.

Standard SQL uses “`--`” as a start-comment sequence. MySQL Server uses “`#`” as the start comment character. MySQL Server 3.23.3 and up also supports a variant of the “`--`” comment style. That is, the “`--`” start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for `payment`:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if `payment` has a negative value such as `-1`:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a legal expression in SQL, but “`--`” is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that permitting comments to start with “`--`” can have serious consequences.

Using our implementation requires a space following the “`--`” for it to be recognized as a start-comment sequence in MySQL Server 3.23.3 and newer. Therefore, `credit--1` is safe to use.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Another safe feature is that the `mysql` command-line client ignores lines that start with "`--`".

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL script in a text file that contains "`--`" comments, you should use the `replace` utility as follows to convert the comments to use "`#`" characters before executing the script:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
         | mysql db_name
```

That is safer than executing the script in the usual way:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the script file "in place" to change the "`--`" comments to "`#`" comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

See Section 4.8.2, "`replace` — A String-Replacement Utility".

1.8.3 How MySQL Deals with Constraints

MySQL enables you to work both with transactional tables that permit rollback and with nontransactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a nontransactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce illegal values to the closest legal values.

Beginning with MySQL 5.0.2, several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement execution or abort when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See Section 5.1.7, "Server SQL Modes".

The following sections describe how MySQL Server handles different types of constraints.

1.8.3.1 PRIMARY KEY and UNIQUE Index Constraints

Normally, errors occurs for data-change statements (such as `INSERT` or `UPDATE`) that would violate primary-key, unique-key, or foreign-key constraints. If you are using a transactional storage engine such

This documentation is for an older version. If you're
This documentation is for an older version. If you're

as InnoDB, MySQL automatically rolls back the statement. If you are using a nontransactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

MySQL supports an IGNORE keyword for INSERT, UPDATE, and so forth. If you use it, MySQL ignores primary-key or unique-key violations and continues processing with the next row. See the section for the statement that you are using (Section 13.2.5, “INSERT Syntax”, Section 13.2.10, “UPDATE Syntax”, and so forth).

You can get information about the number of rows actually inserted or updated with the mysql_info() C API function. You can also use the SHOW WARNINGS statement. See Section 20.6.7.35, “mysql_info()”, and Section 13.7.5.37, “SHOW WARNINGS Syntax”.

Only InnoDB tables support foreign keys. See Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”.

1.8.3.2 FOREIGN KEY Constraints

Foreign keys let you cross-reference related data across tables, and foreign key constraints help keep this spread-out data consistent.

MySQL supports ON UPDATE and ON DELETE foreign key references in CREATE TABLE and ALTER TABLE statements. The available referential actions are RESTRICT (the default), CASCADE, SET NULL, and NO ACTION.

SET DEFAULT is also supported by the MySQL Server but is currently rejected as invalid by InnoDB. Since MySQL does not support deferred constraint checking, NO ACTION is treated as RESTRICT. For the exact syntax supported by MySQL for foreign keys, see Section 13.1.10.3, “Using FOREIGN KEY Constraints”.

MATCH FULL, MATCH PARTIAL, and MATCH SIMPLE are allowed, but their use should be avoided, as they cause the MySQL Server to ignore any ON DELETE or ON UPDATE clause used in the same statement. MATCH options do not have any other effect in MySQL, which in effect enforces MATCH SIMPLE semantics full-time.

MySQL requires that foreign key columns be indexed; if you create a table with a foreign key constraint but no index on a given column, an index is created.

You can obtain information about foreign keys from the INFORMATION_SCHEMA.KEY_COLUMN_USAGE table. An example of a query against this table is shown here:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
     > FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
     > WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+--------------+---------------+-------------+-----------------+
| TABLE_SCHEMA | TABLE_NAME    | COLUMN_NAME | CONSTRAINT_NAME |
+--------------+---------------+-------------+-----------------+
| fk1          | myuser        | myuser_id   | f               |
| fk1          | product_order | customer_id | f2              |
| fk1          | product_order | product_id  | f1              |
+-------------+--------------+-------------+-----------------+
3 rows in set (0.01 sec)
```

Only InnoDB tables support foreign keys. See Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”, for information specific to foreign key support in InnoDB.

1.8.3.3 Constraints on Invalid Data

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Before MySQL 5.0.2, MySQL is forgiving of illegal or improper data values and coerces them to legal values for data entry. In MySQL 5.0.2 and up, that remains the default behavior, but you can enable strict SQL mode to select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. Section 5.1.7, “Server SQL Modes”.

This section describes the default (forgiving) behavior of MySQL, as well as the strict SQL mode and how it differs.

If you are not using strict mode, then whenever you insert an “incorrect” value into a column, such as a `NULL` into a `NOT NULL` column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error: The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that does not start with a number into a numeric column, MySQL Server stores 0.
- Invalid values for `ENUM` and `SET` columns are handled as described in Section 1.8.3.4, “ENUM and SET Constraints”.
- MySQL permits you to store certain incorrect date values into `DATE` and `DATETIME` columns (such as `'2000-02-31'` or `'2000-02-00'`). In this case, when an application has not enabled strict SQL mode, it up to the application to validate the dates before storing them. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value `'0000-00-00'` is stored in the column instead.
- If you try to store `NULL` into a column that doesn't take `NULL` values, an error occurs for single-row `INSERT` statements. For multiple-row `INSERT` statements or for `INSERT INTO ... SELECT` statements, MySQL Server stores the implicit default value for the column data type. In general, this is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. Implicit default values are discussed in Section 11.6, “Data Type Default Values”.
- If an `INSERT` statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit `DEFAULT` clause. If the definition has no such `DEFAULT` clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules in nonstrict mode is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it is better to “do the best you can” and then continue as if nothing happened.

In MySQL 5.0.2 and up, you can select stricter treatment of input values by using the `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` SQL modes:

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

`STRICT_TRANS_TABLES` enables strict mode for transactional storage engines, and also to some extent for nontransactional engines. It works like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- For transactional storage engines, bad data values occurring anywhere in a statement cause the statement to abort and roll back.

- For nontransactional storage engines, a statement aborts if the error occurs in the first row to be inserted or updated. (When the error occurs in the first row, the statement can be aborted to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement, because the table has already been changed by the first row. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with `STRICT_TRANS_TABLES`, a wrong value causes MySQL to roll back all updates done so far, if that can be done without changing the table. But once the table has been changed, further errors result in adjustments and warnings.

For even stricter checking, enable `STRICT_ALL_TABLES`. This is the same as `STRICT_TRANS_TABLES` except that for nontransactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a nontransactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for nontransactional tables, either use single-row statements or else use `STRICT_TRANS_TABLES` if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only legal values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using `INSERT IGNORE` or `UPDATE IGNORE` rather than `INSERT` or `UPDATE` without `IGNORE`.

1.8.3.4 ENUM and SET Constraints

`ENUM` and `SET` columns provide an efficient way to define columns that can contain only a given set of values. See Section 11.4.4, “The ENUM Type”, and Section 11.4.5, “The SET Type”. However, before MySQL 5.0.2, `ENUM` and `SET` columns do not provide true constraints on entry of invalid data:

- `ENUM` columns always have a default value. If you specify no default value, then it is `NULL` for columns that can have `NULL`, otherwise it is the first enumeration value in the column definition.

- If you insert an incorrect value into an `ENUM` column or if you force a value into an `ENUM` column with `IGNORE`, it is set to the reserved enumeration value of `0`, which is displayed as an empty string in string context.

- If you insert an incorrect value into a `SET` column, the incorrect value is ignored. For example, if the column can contain the values `'a'`, `'b'`, and `'c'`, an attempt to assign `'a,x,b,y'` results in a value of `'a,b'`.

As of MySQL 5.0.2, you can configure the server to use strict SQL mode. See Section 5.1.7, “Server SQL Modes”. With strict mode enabled, the definition of a `ENUM` or `SET` column does act as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An `ENUM` value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as `ENUM('a','b','c')`, values such as `''`, `'d'`, or `'ax'` are illegal and are rejected.

- A `SET` value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as `SET('a','b','c')`, values such as `'d'` or `'a,b,c,d'` are illegal and are rejected.

Errors for invalid values can be suppressed in strict mode if you use `INSERT IGNORE` or `UPDATE IGNORE`. In this case, a warning is generated rather than an error. For `ENUM`, the value is inserted as the error member (`0`). For `SET`, the value is inserted as given except that any invalid substrings are deleted. For example, `'a,x,b,y'` results in a value of `'a,b'`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

1.9 Credits

The following sections list developers, contributors, and supporters that have helped to make MySQL what it is today.

1.9.1 Contributors to MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize those who have made contributions of one kind or another to the `MySQL distribution`. Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola `<qwerg@mbox.vol.it>` or `<qwerg@tin.it>`

  The initial port to Win32/NT.

- Per Eric Olsson

  For constructive criticism and real testing of the dynamic record format.

- Irena Pancirov `<irena@mail.yacc.it>`

  Win32 port with Borland compiler. `mysqlshutdown.exe` and `mysqlwatch.exe`.

- David J. Hughes

  For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. `mysqladmin` and `mysql` client are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the MySQL syntax a superset of `mSQL`. Many of the API's ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to the MySQL API. The MySQL software doesn't contain any code from `mSQL`. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based on the corresponding (noncopyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to MySQL Server. (`mSQL` is copyrighted David J. Hughes.)

- Patrick Lynch

  For helping us acquire http://www.mysql.com/.

- Fred Lindberg

  For setting up qmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko `<igor@frog.kiev.ua>`

  `mysqldump` (previously `msqldump`, but ported and enhanced by Monty).

- Yuri Dario

  For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

  Author of `mysqlhotcopy`.

- Zarko Mocnik `<zarko.mocnik@dem.si>`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Sorting for Slovenian language.

- "TAMITO" <tommy@valley.ne.jp>

  The _MB character set macros and the ujis and sjis character sets.

- Joshua Chamas <joshua@chamas.com>

  Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier <Yves.Carlier@rug.ac.be>

  mysqlaccess, a program to show the access rights for a user.

- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)

  For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

  Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper <pixel@organic.com>

  For setting up a searchable mailing list archive at his site.

- Rick Mehalick <Rick_Mehalick@i-o.com>

  For xmysql, a graphical X client for MySQL Server.

- Doug Sisk <sisk@wix.com>

  For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. <axeld@vial.ethz.ch>

  For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

  For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

  For providing RPM versions for MySQL 3.21.

- David Sacerdote <davids@secnet.com>

  Ideas for secure checking of DNS host names.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

  Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

  A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

  FROM_UNIXTIME() time formatting, ENCRYPT() functions, and bison advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

  Ported (and extended) the benchmark suite to DBI/DBD. Have been of great help with crash-me and running benchmarks. Some new date functions. The mysql_setpermission script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

  User-defined functions (UDFs); CREATE FUNCTION and DROP FUNCTION.

- Andreas F. Bobak <bobak@relog.ch>

  The AGGREGATE extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

  Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

  The libmysql.dll library.

- James Pereria <jpereira@iafrica.com>

  Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

  Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

  Examples in the MySQL Tutorial.

- Steve Harvey

  For making mysqlaccess more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

  Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

  Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

  Emulation of pthread_mutex() for OS/2.

- Benjamin Pflugmann

  Extended MERGE tables to handle INSERTS. Active member on the MySQL mailing lists.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Jocelyn Fournier

  Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

  Maintaining the OS X packages and providing invaluable feedback on how to create OS X packages.

- Robert Rutherford

  Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

  Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

- Google Inc.

  We wish to recognize Google Inc. for contributions to the MySQL distribution: Mark Callaghan's SMP Performance patches and other patches.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, `<jehamby@lightside>`, `<psmith@BayNetworks.com>`, `<duane@connect.com.au>`, Ted Deppner `<ted@psyber.com>`, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch `<dkoch@amcity.com>`

  Irix setup.

- Luuk de Boer `<luuk@wxs.nl>`

  Benchmark questions.

- Tim Sailer `<tps@users.buoy.com>`

  `DBD::mysql` questions.

- Boyd Lynn Gerber `<gerberb@zenez.com>`

  SCO-related questions.

- Richard Mehalick `<RM186061@shellus.com>`

  `xmysql`-related questions and basic installation questions.

- Zeev Suraski `<bourbon@netvision.net.il>`

  Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Francesc Guasch <frankie@citel.upc.es>

  General questions.

- Jonathan J Smith <jsmith@wtp.net>

  Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

  Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

  Is flexible and can handle Linux and perhaps HP-UX.

- John Lyon <jlyon@imag.net>

  Questions about installing MySQL on Linux systems, using either `.rpm` files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

  Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

  ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

  `DBD`, Linux, some SQL syntax questions.

1.9.2 Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

  Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

  Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

  For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

  First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded versions were based.

- Jay Flaherty <fty@mediapulse.com>

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

  Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

  French error messages.

- Petr Snajdr, <snajdr@pvt.net>

  Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

  Polish error messages.

- Miguel Angel Fernandez Roiz

  Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

  Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

  Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

  Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

  German error messages.

- Billik Stefan <billik@sun.uniag.sk>

  Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

  Romanian error messages.

- Peter Feher

  Hungarian error messages.

- Roberto M. Serqueira

  Portuguese error messages.

- Carsten H. Pedersen

  Danish error messages.

- Arjen Lentz

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

1.9.3 Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We cannot list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at http://solutions.mysql.com/software/.

- Tim Bunce, Alligator Descartes

  For the `DBD` (Perl) interface.

- Andreas Koenig `<a.koenig@mind.de>`

  For the Perl interface for MySQL Server.

- Jochen Wiedmann `<wiedmann@neckar-alb.de>`

  For maintaining the Perl `DBD::mysql` module.

- Eugene Chan `<eugene@acenet.com.sg>`

  For porting PHP for MySQL Server.

- Georg Richter

  MySQL 4.1 testing and bug hunting. New PHP 5.0 `mysqli` extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli `<maruzz@matrice.it>`

  For porting iODBC (Unix ODBC).

- Xavier Leroy `<Xavier.Leroy@inria.fr>`

  The author of LinuxThreads (used by the MySQL Server on Linux).

1.9.4 Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

  From whom we got an excellent compiler (`gcc`), an excellent debugger (`gdb` and the `libc` library (from which we have borrowed `strto.c` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

  For a really great editor/environment.

- Julian Seward

  Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Dorothea Lütkehaus and Andreas Zeller

  For `DDD` (The Data Display Debugger) which is an excellent graphical front end to `gdb`).

1.9.5 Supporters of MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize the following companies, which helped us finance the development of the `MySQL server`, such as by paying us for developing a new feature or giving us hardware for development of the `MySQL server`.

- VA Linux / Andover.net

  Funded replication.

- NuSphere

  Editing of the MySQL manual.

- Stork Design studio

  The MySQL Web site in use between 1998-2000.

- Intel

  Contributed to development on Windows and Linux platforms.

- Compaq

  Contributed to Development on Linux/Alpha.

- SWSoft

  Development on the embedded `mysqld` version.

- FutureQuest

  The `--skip-show-database` option.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 2 Installing and Upgrading MySQL

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

End of Product Lifecycle. Active development for MySQL Database Server version 5.0 has ended. Oracle offers various support offerings which may be of interest. For details and more information, see the MySQL section of the Lifetime Support Policy for Oracle Technology Products (http://www.oracle.com/us/support/lifetime-support/index.html). Please consider upgrading to a recent version.

2.1 MySQL Installation Overview

This chapter describes how to obtain and install MySQL. You can choose to install MySQL Enterprise or MySQL Community Server:

- MySQL Enterprise is Oracle Corporation's commercial offering for modern enterprise businesses. It includes MySQL Enterprise Server and the services provided by MySQL Network. To install MySQL Enterprise, see Section 2.3, “Notes for MySQL Enterprise Server”.
- MySQL Community Server is for users who are comfortable configuring and administering MySQL by themselves. To install MySQL Community Server, see Section 2.4, “Notes for MySQL Community Server”.

If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see Section 2.19.1, “Upgrading MySQL”, for information about upgrade procedures and about issues that you should consider before upgrading.

If you are interested in migrating to MySQL from another database system, you may wish to read Section A.8, “MySQL 5.0 FAQ: Migration”, which contains answers to some common questions concerning migration issues.

2.2 Determining Your Current MySQL Version

To determine the version and release of your currently installed MySQL installation, there are a number of options.

- Using a command client (`mysql`), the server version of the MySQL server to which you are connected is shown once you are connected. The server version information includes `community` or `enterprise` accordingly.

  For example, here is the output from a MySQL Community Server edition installed on Linux:

This documentation is for an older version. If you're

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.27-standard MySQL Community Edition - Standard (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

This is an example of the output from MySQL Enterprise Server on Windows:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.0.28-enterprise-gpl-nt MySQL Enterprise Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

- You may also determine the version information using the version variables. Both the version and version_comment variables contain version information for the server to which you are connected. Use the `SHOW VARIABLES` statement to obtain the information you want, as shown in this example:

```
mysql> SHOW VARIABLES LIKE "%version%";
+-------------------------+-----------------------------------------+
| Variable_name           | Value                                   |
+-------------------------+-----------------------------------------+
| protocol_version        | 10                                      |
| version                 | 5.0.27-standard                         |
| version_comment         | MySQL Community Edition - Standard (GPL) |
| version_compile_machine | i686                                    |
| version_compile_os      | pc-linux-gnu                            |
+-------------------------+-----------------------------------------+
5 rows in set (0.04 sec)
```

  You can also obtain server version information in the `mysql` client using the `SELECT VERSION()` statement. In addition, MySQL Workbench also shows the server version in the **Server Status** tab. However, in both of these cases, only the value of `version` is shown.

- The `STATUS` command displays the version as well as version comment information. For example:

```
mysql> STATUS;
--------------
./client/mysql  Ver 14.12 Distrib 5.0.29, for pc-linux-gnu (i686) using readline 5.0

Connection id:          8
Current database:
Current user:           mc@localhost
SSL:                    Not in use
Current pager:          /usr/bin/less
Using outfile:          ''
Using delimiter:        ;
Server version:         5.0.27-standard MySQL Community Edition - Standard (GPL)
Protocol version:       10
Connection:             Localhost via UNIX socket
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    latin1
Conn.  characterset:    latin1
UNIX socket:            /tmp/mysql.sock
Uptime:                 1 day 3 hours 58 min 43 sec

Threads: 2  Questions: 17  Slow queries: 0  Opens: 11  Flush tables: 1  Open tables: 6  Queries per seco
--------------
```

2.3 Notes for MySQL Enterprise Server

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To obtain MySQL Enterprise, visit http://enterprise.mysql.com if you're a customer. Otherwise, visit http://www.mysql.com/products/enterprise/. The platforms that are officially supported for MySQL Enterprise are listed at http://www.mysql.com/support/supportedplatforms/enterprise.html.

MySQL Enterprise Server is available for download in the form of *Quarterly Service Pack (QSP)* or *Monthly Rapid Update (MRU)* binary releases.

To install MySQL Enterprise Server, you should use the latest available Quarterly Service Pack (QSP). This includes an accumulation of the bug fixes provided in all predecessor QSP and MRU releases.

MRU releases are provided on a monthly basis and represent the most current Enterprise Server bug fixes. Each MRU is an accumulation of the bug fixes included in its predecessor. Customers should standardize on the latest MRU release only if it includes a needed bug fix.

2.3.1 Enterprise Server Distribution Types

Enterprise Server releases will be created for the following packages from the MySQL 5.0 tree:

- `mysql-enterprise`: Released under a commercial license and includes the following storage engines: `MyISAM`, `MEMORY`, `MERGE`, `InnoDB`, `ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, `FEDERATED`.
- `mysql-enterprise-gpl`: Same as `mysql-enterprise`, but released under the GPL.
- `mysql-cluster`: `mysql-enterprise` plus MySQL Cluster (`NDB`).
- `mysql-classic`: Released under a commercial license, does not include `InnoDB`.
- `mysql-community`: Same as `mysql-enterprise-gpl`, but available for the community, and released every 6 months.

To satisfy different user requirements, we provide several servers. `mysqld` is an optimized server that is a smaller, faster binary. `mysqld-debug` is compiled with debugging support but is otherwise configured identically to the nondebug server.

Each of these servers is compiled from the same source distribution, though with different configuration options. All native MySQL clients can connect to servers from either MySQL version.

2.3.2 Upgrading MySQL Enterprise Server

When upgrading to MySQL Enterprise from Community Server you need only follow the installation process to install and upgrade the packages to the latest version provided by MySQL Enterprise. You will also need to install the latest MySQL Enterprise Service Pack and any outstanding MySQL Hot-fix packs.

Be aware, however, that you must take into account any of the changes when moving between major releases. You should also check the Release Notes for details on major changes between revisions of MySQL Enterprise Server.

You should also review the notes and advice contained within Section 2.19.1, “Upgrading MySQL”.

2.4 Notes for MySQL Community Server

2.4.1 Overview of MySQL Community Server Installation

1. **Determine whether MySQL runs and is supported on your platform.** Not all platforms are equally suitable for running MySQL, and not all platforms on which MySQL is known to run are officially supported by Oracle Corporation.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. **Choose which distribution to install.** Several versions of MySQL are available, and most are available in multiple distribution formats. You can choose from prepackaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source trees for those who want to see our most recent developments and to help us test new code. To determine which version and type of distribution you should use, see Section 2.4.2, “Choosing Which MySQL Distribution to Install”.

3. **Download the distribution that you want to install.** For download instructions, see Section 2.5, “How to Get MySQL”. To verify the integrity of the distribution, use the instructions in Section 2.6, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

4. **Install the distribution.** To install MySQL from a binary distribution, use the instructions in Section 2.9, “Standard MySQL Installation from a Binary Distribution”. To install MySQL from a source distribution or from the current development source tree, use the instructions in Section 2.17, “Installing MySQL from Source”.

   If you encounter installation difficulties, see Section 2.20, “Operating System-Specific Notes”, for information on solving problems for particular platforms.

5. **Perform any necessary postinstallation setup.** After installing MySQL, read Section 2.18, “Postinstallation Setup and Testing”, which contains important information about making sure the MySQL server is working properly. It also describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The information in this section applies whether you install MySQL using a binary or source distribution.

6. **Perform setup for running benchmarks (optional).** If you want to use the MySQL benchmark scripts, Perl support for MySQL must be available. See Section 2.22, “Perl Installation Notes”, for more information.

The sections immediately following this one contain necessary information about choosing, downloading, and verifying your distribution. The instructions in later sections of the chapter describe how to install the distribution that you choose. For binary distributions, see the instructions in Section 2.9, “Standard MySQL Installation from a Binary Distribution”. To build MySQL from source, use the instructions in Section 2.17, “Installing MySQL from Source”.

2.4.2 Choosing Which MySQL Distribution to Install

MySQL is available on a number of operating systems and platforms. For information about those platforms that are officially supported, see http://www.mysql.com/support/supportedplatforms/database.html on the MySQL Web site.

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.4.2.1 Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity.

Production Releases

- MySQL 5.7: Latest General Availability (Production) release
- MySQL 5.6: Previous General Availability (Production) release

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- MySQL 5.5: Older General Availability (Production) release
- MySQL 5.1, 5.0: Older Production releases for which active development has ended

MySQL 4.1, 4.0, and 3.23 are old releases that are no longer supported.

See http://www.mysql.com/about/legal/lifecycle/ for information about support policies and schedules. For supported platform information, see http://www.mysql.com/support/supportedplatforms/database.html.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, use the most recent General Availability series listed in the preceding descriptions. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an older system and want to upgrade, but do not want to take the chance of having a nonseamless upgrade, you should upgrade to the latest version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make only small, relatively “safe” changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Be aware that development releases are not as stable as production releases.

We do not use a complete code freeze because this prevents us from making bugfixes and other fixes that must be done. We may add small things that should not affect anything that currently works in a production release. Naturally, relevant bugfixes from an earlier series propagate to later series.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our source code repositories (see Section 2.17.2, “Installing MySQL Using a Development Source Tree”). These are not “releases” as such, but are available as previews of the code on which future releases are to be based.

The naming scheme in MySQL 5.0 uses release names that consist of three numbers and a suffix; for example, **mysql-5.0.14-rc**. The numbers within the release name are interpreted as follows:

- The first number (**5**) is the major version and describes the file format. All MySQL 5 releases have the same file format.
- The second number (**0**) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (**14**) is the version number within the release series. This is incremented for each new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix that indicates the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **alpha** indicates that the release is for preview purposes only. Known bugs should be documented in the Release Notes. Most alpha releases implement new commands and extensions. Active development that may involve major code changes can occur in an alpha release. However, we do conduct testing before issuing a release.
- **beta** indicates that the release is appropriate for use with new development. Within beta releases, the features and compatibility should remain consistent. However, beta releases may contain numerous and major unaddressed bugs.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

No APIs, externally visible structures, or columns for SQL statements will change during future beta, release candidate, or production releases.

- **rc** indicates a Release Candidate. Release candidates are believed to be stable, having passed all of MySQL's internal testing, and with all known fatal runtime bugs fixed. However, the release has not been in widespread use long enough to know for sure that all bugs have been identified. Only minor fixes are added. (A release candidate is what formerly was known as a gamma release.)

- If there is no suffix, it indicates that the release is a General Availability (GA) or Production release. GA releases are stable, having successfully passed through all earlier release stages and are believed to be reliable, free of serious bugs, and suitable for use in production systems. Only critical bugfixes are applied to the release.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with these tools:

- **An internal test suite.** The `mysql-test` directory contains an extensive set of test cases. We run these tests for every server binary. See Section 21.1.2, “The MySQL Test Suite”, for more information about this test suite.

- **The MySQL benchmark suite.** This suite runs a range of common queries. It is also a test to determine whether the latest batch of optimizations actually made the code faster. See Section 8.13.2, “The MySQL Benchmark Suite”.

We also perform additional integration and nonfunctional testing of the latest MySQL version in our internal production environment. Integration testing is done with different connectors, storage engines, replication modes, backup, partitioning, stored programs, and so forth in various combinations. Additional nonfunctional testing is done in areas of performance, concurrency, stress, high volume, upgrade and downgrade.

2.4.2.2 Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your platform. Binary distributions are available in native format for many platforms, such as RPM packages for Linux, DMG packages for OS X, and PKG packages for Solaris. Distributions are also available in more generic formats such as Zip archives or compressed `tar` files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.

- To satisfy different user requirements, we provide several servers in binary distributions. `mysqld` is an optimized server that is a smaller, faster binary. `mysqld-debug` is compiled with debugging support.

  Each of these servers is compiled from the same source distribution, though with different configuration options. All native MySQL clients can connect to servers from either MySQL version.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- You want to configure `mysqld` to ensure that features are available that might not be included in the standard binary distributions. Here is a list of the most common extra options that you may want to use to ensure feature availability:
  - `--with-berkeley-db` (not available on all platforms)
  - `--with-libwrap`
  - `--with-named-z-libs` (this is done for some of the binaries)
  - `--with-debug[=full]`

  For additional information, see Section 2.17.3, “MySQL Source-Configuration Options”.
- You want to configure `mysqld` without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You want to use the latest sources from one of the Bazaar repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.
- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.4.2.3 How and When Updates Are Released

MySQL is evolving quite rapidly and we want to share new developments with other MySQL users. We try to produce a new release whenever we have new and useful features that others also seem to have a need for.

We also try to help users who request features that are easy to implement. We take note of what our licensed users want, and we especially take note of what our support customers want and try to help them in this regard.

No one is *required* to download a new release. The Release Notes help you determine whether the new release has something you really want.

We use the following policy when updating MySQL:

- Enterprise Server releases are meant to appear every 18 months, supplemented by quarterly service packs and monthly rapid updates. Community Server releases are meant to appear 2−3 times per year.
- Releases are issued within each series. For each release, the last number in the version is one more than the previous release within the same series.
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or noncritical but annoying bugs. The fixes are available in source form immediately from our public Bazaar repositories, and are included in the next release.
- If by any chance a security vulnerability or critical bug is found in a release, our policy is to fix it in a new release as soon as possible. (We would like other companies to do this, too!)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.4.2.4 MySQL Binaries Compiled by Oracle Corporation

Oracle Corporation provides a set of binary distributions of MySQL. In addition to binaries provided in platform-specific package formats, we offer binary distributions for a number of platforms in the form of compressed `tar` files (`.tar.gz` files). See Section 2.9, “Standard MySQL Installation from a Binary Distribution”. For Windows distributions, see Section 2.10, “Installing MySQL on Microsoft Windows”.

If you want to compile MySQL from a source distribution, see Section 2.17, “Installing MySQL from Source”. To compile a debug version of MySQL, see Section 2.17.3, “MySQL Source-Configuration Options” for options that enable debugging.

2.5 How to Get MySQL

Check our downloads page at http://dev.mysql.com/downloads/ for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see http://dev.mysql.com/downloads/mirrors.html. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

To obtain the latest development source, see Section 2.17.2, “Installing MySQL Using a Development Source Tree”.

2.6 Verifying Package Integrity Using MD5 Checksums or GnuPG

After downloading the MySQL package that suits your needs and before attempting to install it, make sure that it is intact and has not been tampered with. There are three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using `GnuPG`, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site.

2.6.1 Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify against the package that you downloaded. The correct MD5 checksum is listed on the downloads page for each MySQL product, and you will compare it against the MD5 checksum of the file (product) that you download.

Each operating system and setup offers its own version of tools for checking the MD5 checksum. Typically the command is named `md5sum`, or it may be named `md5`, and some operating systems do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can also download the source code from http://www.gnu.org/software/textutils/. If you have OpenSSL installed, you can use the command `openssl md5 package_name` instead. A Windows implementation of the `md5` command line utility is available from http://www.fourmilab.ch/md5/. `winMd5Sum` is a graphical MD5 checking tool that can be obtained from http://www.nullriver.com/index/products/winmd5sum. Our Microsoft Windows examples will assume the name `md5.exe`.

Linux and Microsoft Windows examples:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> md5sum mysql-standard-5.0.96-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.0.96-linux-i686.tar.gz
```

```
shell> md5.exe mysql-installer-community-5.0.96.msi
aaab65abbec64d5e907dcd41b8699945  mysql-installer-community-5.0.96.msi
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note

Make sure to verify the checksum of the *archive file* (for example, the `.zip`, `.tar.gz`, or `.msi` file) and not of the files that are contained inside of the archive. In other words, verify the file before extracting its contents.

2.6.2 Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

We sign MySQL downloadable packages with GnuPG (GNU Privacy Guard). GnuPG is an Open Source alternative to the well-known Pretty Good Privacy (PGP) by Phil Zimmermann. See http://www.gnupg.org/ for more information about GnuPG and how to obtain and install it on your system. Most Linux distributions ship with GnuPG installed by default. For more information about GnuPG, see http://www.openpgp.org/.

To verify the signature for a specific package, you first need to obtain a copy of our public GPG build key, which you can download from http://pgp.mit.edu/. The key that you want to obtain is named `mysql-build@oss.oracle.com`. Alternatively, you can cut and paste the key directly from the following text:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.9 (SunOS)

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q2TXlTUUwgUmVs
ZWFzZSBFbmdpbmVlcmluZyA8bXlzcWwtYnVpbGRAb3NzLm9yYWNsZS5jb20+iGkE
ExECACkCGyMGCwkIBwMCBBUCCAMEFgIDAQIeAQIXgAIZAQUCUwHUZgUJGmbLywAK
CRCMcY07UHLh9V+DAKCjS1gGwgVI/eut+5L+l2v3ybl+ZgCcD7ZoA341HtoroV3U
6xRD09fUgeq0O015U1FMIFBhY2thZ2Ugc2lnbmluZyBrZXkgKHd3dy5teXNxbC5j
b20pIDxidWlsZEBteXNxbC5jb20+iG8EMBECAC8FAk53Pa0oHSBidWlsZEBteXNx
bC5jb20gd2lsbCBzdG9wIHdvcmtpbmcgc29vbgAKCRCMcY07UHLh9bU9AJ9xDK0o
xJFL9vTl9OSZC4lX0K9AzwCcCrS9cnJyz79eaRjL0s2r/CcljdyIZQQTEQIAHQUC
R6yUtAUJDTBYqAULBwoDBAMVAwIDFgIBAheAABIJEIxxjTtQcuH1B2VHUEcAAQGu
kgCffz4GUEjzXkOi71VcwgCxASTgbe0An34LPr1j9fCbrXWXO14msIADfb5piEwE
ExECAAwFAj4+o9EFgwlmALsACgkQSVDhKrJykfIk4QCfWbEeKN+3TRspe+5xKj+k
QJSammIAnjUz0xFWPlVx0f8o38qNG1bq0cU9iEwEExECAAwFAj5CggMFgwliIokA
CgkQtvXNTca6JD+WkQCgiGmnoGjMojynp5ppvMXkyUkfnykAoK79E6h8rwkSDZou
iz7nMRisH8uyiEYEEBECAAYFAj+s468ACgkQr8UjSHiDdA/2lgCg21IhIMMABTYd
p/IBiUsP/JQLiEoAnRzMywEtujQz/E9ono7H1DkebDa4iEYEEBECAAYFAj+0Q3cA
CgkQhZavqzBzTmbGwwCdFqD1frViC7WRt8GKoOS7hzNN32kAnirlbwpnT7a6NOsQ
83nk11a2dePhiEYEEBECAAYFAkNbs+oACgkQi9gubzC5S1x/dACdELKoXQKkwJN0
gZztsM7kjsIgyFMAnRRMbHQ7V39XC90OIpaPjk3a01tgiEYEExECAAYFAkTxMyYA
CgkQ9knE9GCTUwwKcQCgibak/SwhxWH1ijRhgYCo5GtM4vcAnAhtzL57wcw1Kg1X
m7nVGetUqJ7fiEwEEBECAAwFAkGBywEFgwYi2YsACgkQGFnQH2d7oexCjQCcD8sJ
NDc/mS8m8OGDUOx9VMWcnGkAnj1YWOD+Qhxo3mI/Ul9oEAhNkjcfiEwEEBECAAwF
```


This documentation is for an older version. If you're

```
AkGByzQFgwYi2VgACgkQgcL36+ITtpIiIwCdFVNVUB8xe8mFXoPm4d9Z54PTjpMA
niSPA/ZsfJ3oOMLKar4F0QPPrdrGiEwEEBECAAwFAkGBy2IFgwYi2SoACgkQa3Ds
2V3D9HMJqgCbBYzr5GPXOXgP88jKzmdbjweqXeEAnRss4G2G/3qD7uhTL1SPT1SH
jWUXiEwEEBECAAwFAkHQkyQFgwXUEWgACgkQfSXKCsEpp8JiVQCghvWvkPqowsw8
w7WSseTcw1tflvkAni+vLHl/DqIly0LkZYn5jzK1dpvfiEwEEBECAAwFAkIrW7oF
gwV5SNIACgkQ5hukiRXruavzEwCgkzL5QkLSypcw9LGHcFSx1ya0VL4An35nXkum
g6cCJ1NP8r2I4NcZWIrqiEwEEhECAAwFAkAqWToFgwd6S1IACgkQPKEfNJT6+GEm
XACcD+A53A5OGM7w750W11ukq4iZ9ckAnRMvndAqn3YTOxxlLPj2UPZiSgSqiEwE
EhECAAwFAkA9+roFgwdmqdIACgkQ8tdcY+OcZZyy3wCgtDcwlaq20w0cNuXFLLNe
EUaFFTwAni6RHN80moSVAdDTRkzZacJU3M5QiEwEEhECAAwFAkEOCoQFgwaWmggA
CgkQOcor9D1qil/83QCeITZ9wIo7XAMjC6y4ZWUL4m+edZsAoMOhRIRi42fmrNFu
vNZbnMGej81viEwEEhECAAwFAkApTQFgwUj/1gACgkQBA3AhXyDn6jjJACcD1A4
UtXk84J13JQyoH9+dy24714Aniwlsso/9ndICJOkqs2j5dlHFq6oiEwEExECAAwF
Aj5NTYQFgwlXVwgACgkQLbt2v63UyTMFDACglT5G5NVKf5Mj65bFSlPzb92zk2QA
n1uc2h19/IwwrsbIyK/9POJ+JMP7iEwEExECAAwFAkHXgHYFgwXNJBYACgkQZu/b
yM2C/T4/vACfXe67xiSHB80wkmFZ2krb+oz/gBAAnjR2ucpbaonkQQgnC3GnBqmC
vNaJiEwEExECAAwFAkIYgQ4FgwWMI34ACgkQdsEDHKIxbqGg7gCfQi2HcrHn+yLF
uNlH1oSOh48ZM0oAn3hKV0uIRJphonHaUYiUP1ttWgdBiGUEExECAB0FCwcKAwQD
FQMCAxYCAQIXgAUCS3AvygUJEPPzpwASB2VHUEcAAQEJEIxxjTtQcuH1sNsAniYp
YBGqy/HhMnw3WE8kXahOOR5KAJ4xUmWPGYP4l3hKxyNK9OAUbpDVYIh7BDARAgA7
BQJCdzX1NB0AT29wcy4uLiBzaG91bGQgaGF2ZSBiZWVuIGxvY2FsISBJJ20gKnNv
KiBzdHVwaWQuLi4ACgkQOcor9D1qil/vRwCdFo08f66oKLiuEAqzlf9iDlPozEEA
n2EgvCYLCCHjfGosrkrU3WK5NFVgiI8EMBECAE8FAkVvAL9IHQBTaG91bGQgaGF2
ZSBiZWVuIGEgbG9jYWwgc2lnbmF0dXJlLCBvciBzb21ldGhpbmcgLSBXVEYgd2Fz
IEkgdGhpbmtpbmc/AAoJEDnKK/Q9aopfoPsAn3BVqKOalJeF0xPSvLR90PsRlnmG
AJ44oisY7Tl3NJbPgZal8W32fbqgbIkCIgQQAQIADAUCQYHLhQWDBiLZBwAKCRCq
4+bOZqFEaKgvEACCErnaHGyUYa0wETjj6DLEXsqeOiXad4i9aBQxnD35GUgcFofC
/nCY4XcnCMMEnmdQ9ofUuU3OBJ6BNJIbEusAabgLooebP/3KEaiCIiyhHYU5jarp
ZAh+Zopgs3Oc11mQ1tIaS69iJxrGTLodkAsAJAeEUwTPq9fHFFzC1eGBysoyFWg4
bIjz/zClI+qyTbFA5g6tRoiXTo8ko7QhY2AA5UGEg+83Hdb6akC04Z2QRErxKAqr
phHzj8XpjVOsQAdAi/qVKQeNKROlJ+iq6+YesmcWGfzeb87dGNweVFDJIGA0qY27
pTb2lExYjsRFN4Cb13NfodAbMTOxcAWZ7jAPCxAPlHUG++mHMrhQXEToZnBFE4nb
nC7vOBNgWdjUgXcpkUCkop4b17BFpR+k8ZtYLSS8p2LLz4uAeCcSm2/msJxT7rC/
FvoH8428oHincqs2ICo9zO/Ud4HmmO0O+SsZdVKIIjinGyOVWb4OOzkAlnnhEZ3o
6hAHcREIsBgPwEYVTj/9ZdC0AO44Nj9cU7awaqgtrnwwfr/o4V2gl8bLSkltZU27
/29HeuOeFGjlFe0YrDd/aRNsxbyb2O28H4sG1CVZmC5uK1iQBDiSyA7Q0bbdofCW
oQzm5twlpKWnY8Oe0ub9XP5p/sVfck4FceWFHwv+/PC9RzSl33lQ6vM2wIkCIgQT
AQIADAUCQp8KHAWDBQWacAAKCRDYwgoJWiRXzyE+D/9uc7z6fIsalfOYoLN60ajA
bQbI/uRKBFugyZ5RoaItusn9Z2rAtn61WrFhu4uCSJtFN1ny2RERg40f56pTghKr
D+YEt+Nze6+FKQ5AbGIdFsR/2bUk+ZZRSt83e14Lcb6ii/fJfzkoIox9ltkifQxq
Y7Tvk4noKu4oLSc8O1Wsfc/y0B9sYUUCmUfcnq58DEmGie9ovUslmyt5NPnveXxp
5UeaRc5Rqt9tK2B4A+7/cqENrdZJbAMSunt2+2fkYiRunAFPKPBdJBsY1sxeL/A9
aKe0viKEXQdAWqdNZKNCi8rd/oOP99/9lMbFudAbX6nL2DSb1OG2Z7NWEqgIAzjm
pwYYPCKeVz5Q8R+if9/fe5+STY/55OaI33fJ2H3v+U435VjYqbrerWe36xJItcJe
qUzW71fQtXi1CTEl3w2ch7VF5oj/QyjabLnAlHgSlkSi6p7By5C2MnbCHlCfPnIi
nPhFoRcRGPjJe9nFwGs+QblvS/Chzc2WX3s/2SWm4gEUKRX4zsAJ5ocyfa/vkxCk
SxK/erWlCPf/J1T70+i5waXDN/E3enSet/WL7h94pQKpjz8OdGL4JSBHuAVGA+a+
dknqnPF0KMKLhjrgV+L7O84FhbmAP7PXm3xmiMPriXf+el5fZZequQoIagf8rdRH
HhRJxQgI0HNknkaOqs8dtrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWs
EN/lxaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLm
RDRiRjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hk
AWzE7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkb
f4fmLe11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8Ilbf5vSYHb
uE5p/1oIDznkg/p8kW+3FxuWryccciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+
Lwqqa8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1Z
aSafanFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGo
TbOWI39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev4
2LmuQT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkK
Ht926s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUO
etdZWhe70YGNPw1yjWJT1IhUBBgRAgAMBQJOdz3tBQkT+wG4ABIHZUdQRwABAQkQ
jHGNO1By4fUUmwCbBYr2+bBEn/L2BOcnw9Z/QFWuhRMAoKVgCFm5fadQ3Afi+UQl
AcOphrnJ
=443I
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use `gpg --import`. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> gpg --import mysql_pubkey.asc
gpg: key 5072E1F5: public key "MySQL Release Engineering
<mysql-build@oss.oracle.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: no ultimately trusted keys found
```

You can also download the key from the public keyserver using the public key id, `5072E1F5`:

```
shell> gpg --recv-keys 5072E1F5
gpg: requesting key 5072E1F5 from hkp server keys.gnupg.net
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
1 new user ID
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
53 new signatures
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:           new user IDs: 1
gpg:         new signatures: 53
```

If you want to import the key into your RPM configuration to validate RPM install packages, you should be able to import the key directly:

```
shell> rpm --import mysql_pubkey.asc
```

If you experience problems or require RPM specific information, see Section 2.6.4, “Signature Checking Using RPM”.

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension, as shown by the examples in the following table.

Table 2.1 MySQL Package and Signature Files for Source files

File Type	File Name
Distribution file	`mysql-standard-5.0.96-linux-i686.tar.gz`
Signature file	`mysql-standard-5.0.96-linux-i686.tar.gz.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

If the downloaded package is valid, you will see a "Good signature" similar to:

```
shell> gpg --verify mysql-standard-5.0.96-linux-i686.tar.gz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

The `Good signature` message indicates that the file signature is valid, when compared to the signature listed on our site. But you might also see warnings, like so:

```
shell> gpg --verify mysql-standard-5.0.96-linux-i686.tar.gz.asc
gpg: Signature made Wed 23 Jan 2013 02:25:45 AM PST using DSA key ID 5072E1F5
gpg: checking the trustdb
gpg: no ultimately trusted keys found
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5
```

That is normal, as they depend on your setup and configuration. Here are explanations for these warnings:

- *gpg: no ultimately trusted keys found*: This means that the specific key is not "ultimately trusted" by you or your web of trust, which is okay for the purposes of verifying file signatures.
- *WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.*: This refers to your level of trust in your belief that you possess our real public key. This is a personal decision. Ideally, a MySQL developer would hand you the key in person, but more commonly, you downloaded it. Was the download tampered with? Probably not, but this decision is up to you. Setting up a web of trust is one method for trusting them.

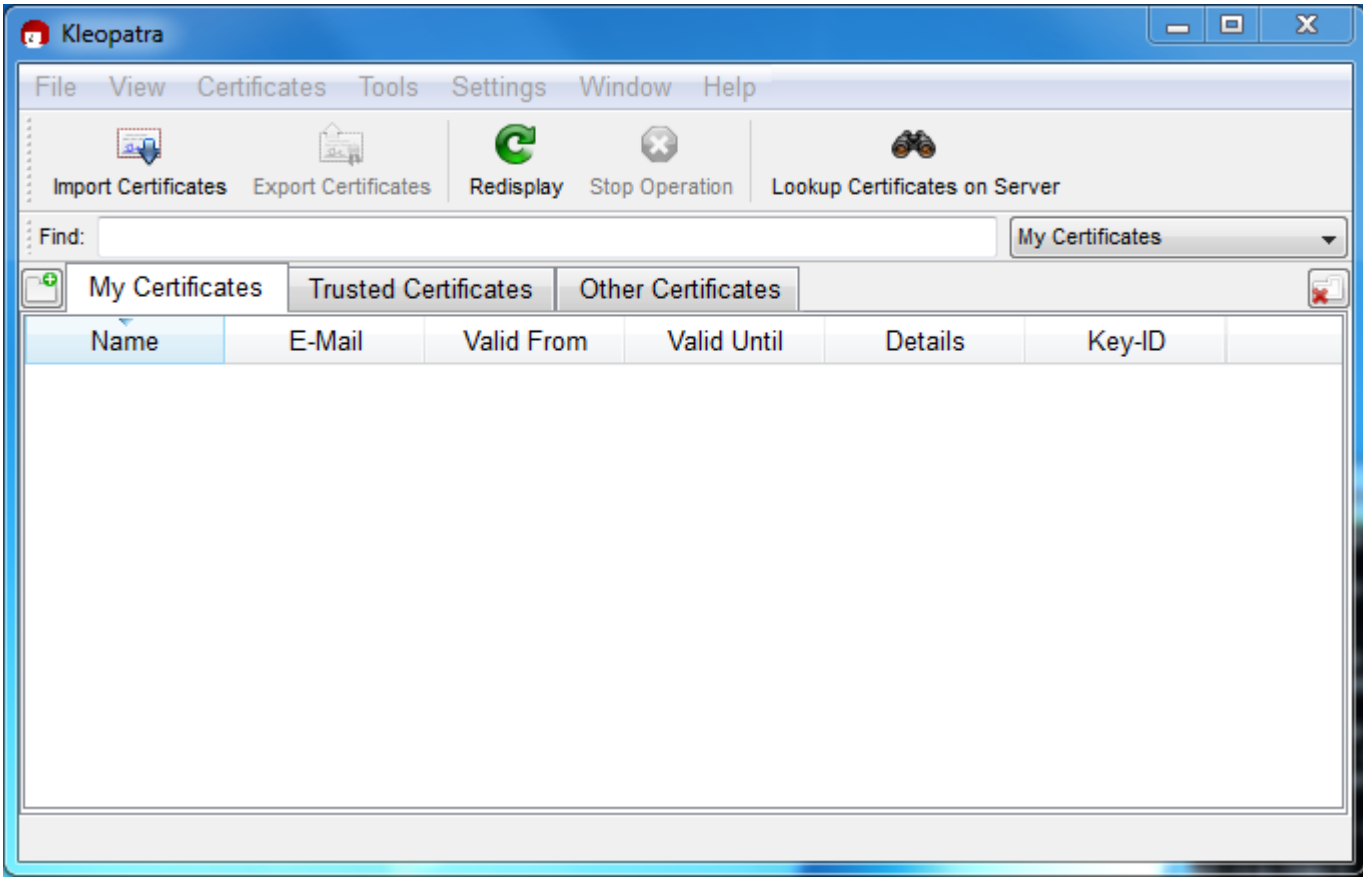
See the GPG documentation for more information on how to work with public keys.

2.6.3 Signature Checking Using Gpg4win for Windows

The Section 2.6.2, “Signature Checking Using GnuPG” section describes how to verify MySQL downloads using GPG. That guide also applies to Microsoft Windows, but another option is to use a GUI tool like Gpg4win. You may use a different tool but our examples are based on Gpg4win, and utilize its bundled `Kleopatra` GUI.

Download and install Gpg4win, and then load Kleopatra. The dialog should look similar to:

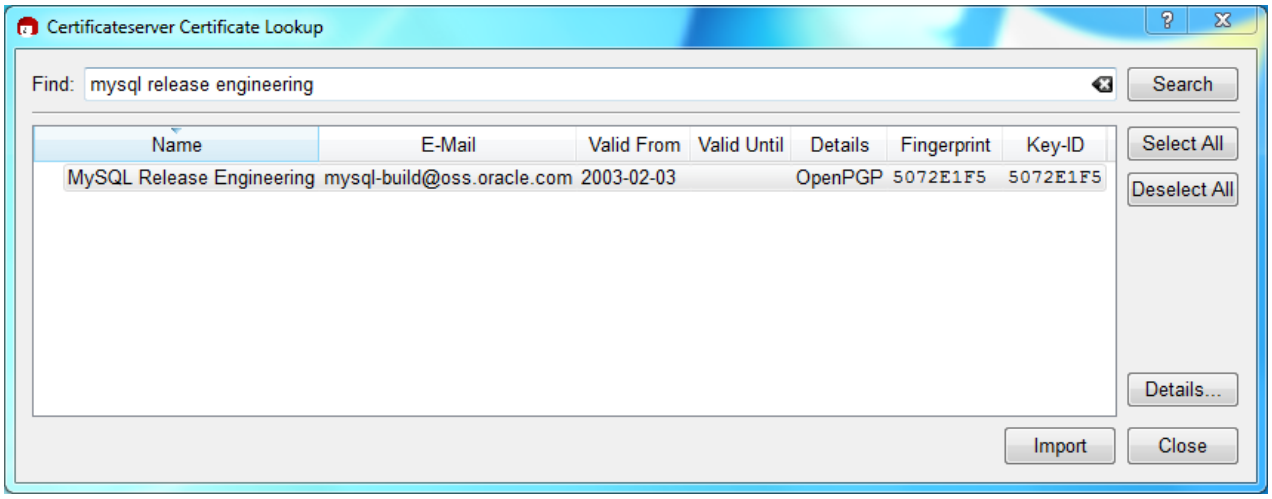
Figure 2.1 Initial screen after loading Kleopatra



This documentation is for an older version. If you're

Next, add the MySQL Release Engineering certificate. Do this by clicking **File**, **Lookup Certificates on Server**. Type "Mysql Release Engineering" into the search box and press **Search**.

Figure 2.2 Finding the MySQL Release Engineering certificate

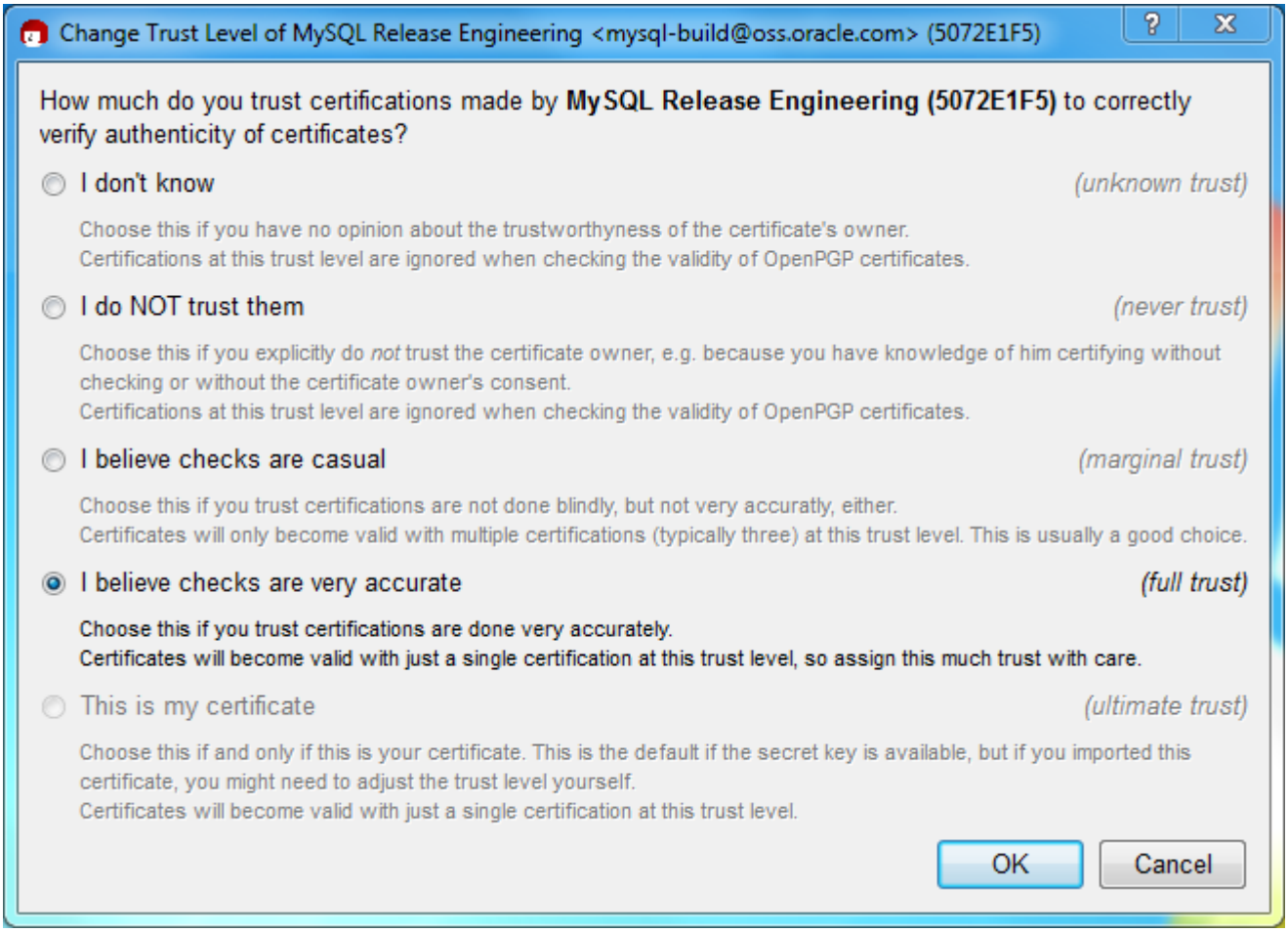


Select the "MySQL Release Engineering" certificate. The Fingerprint and Key-ID must be "5072E1F5", or choose **Details...** to confirm the certificate is valid. Now, import it by clicking **Import**. An import dialog will be displayed, choose **Okay**, and this certificate will now be listed under the **Imported Certificates** tab.

Next, configure the trust level for our certificate. Select our certificate, then from the main menu select **Certificates**, **Change Owner Trust...**. We suggest choosing **I believe checks are very accurate** for our certificate, as otherwise you might not be able to verify our signature. Select **I believe checks are very accurate** and then press **OK**.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Figure 2.3 Changing the Trust level



Next, verify the downloaded MySQL package file. This requires files for both the packaged file, and the signature. The signature file must have the same name as the packaged file but with an appended `.asc` extension, as shown by the example in the following table. The signature is linked to on the downloads page for each MySQL product. You must create the `.asc` file with this signature.

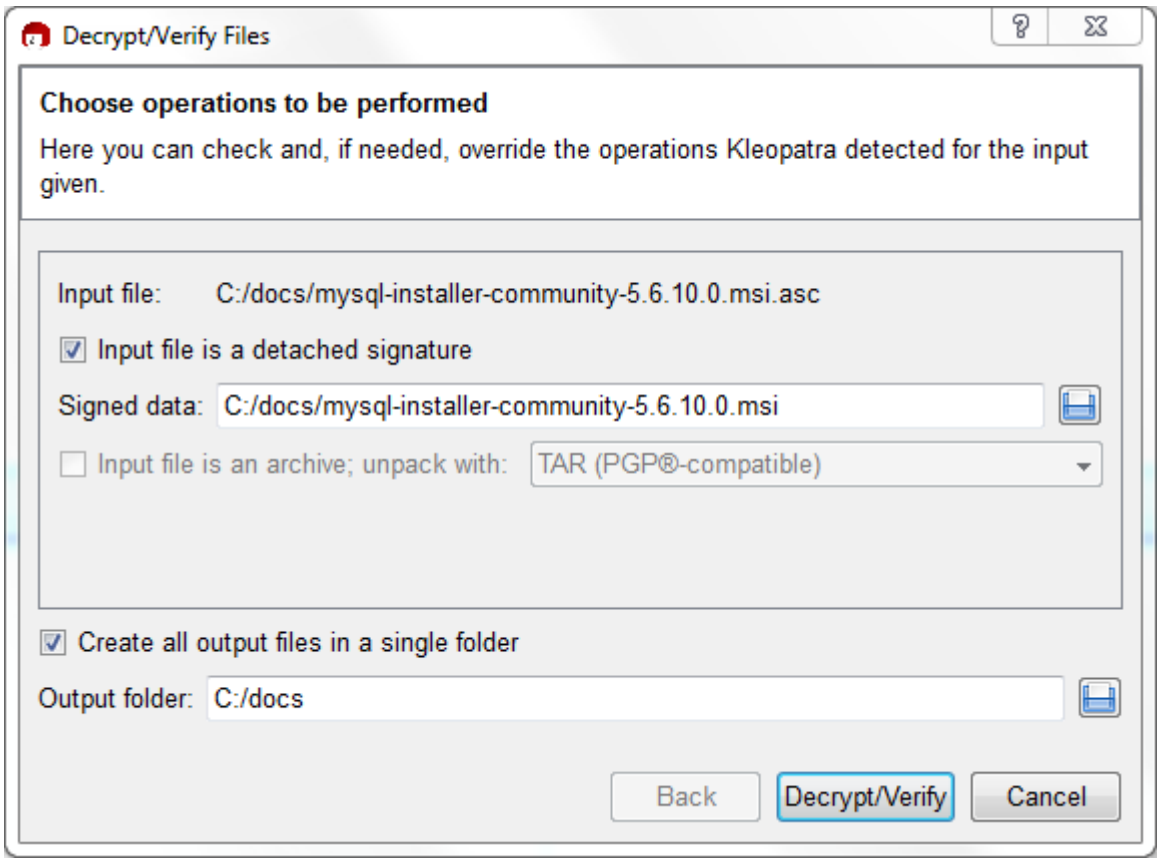
Table 2.2 MySQL Package and Signature Files for MySQL Installer for Microsoft Windows

File Type	File Name
Distribution file	`mysql-installer-community-5.0.96.msi`
Signature file	`mysql-installer-community-5.0.96.msi.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file. Either drag and drop the signature (`.asc`) file into Kleopatra, or load the dialog from **File**, **Decrypt/Verify Files...**, and then choose either the `.msi` or `.asc` file.

This documentation is for an older version. If you're

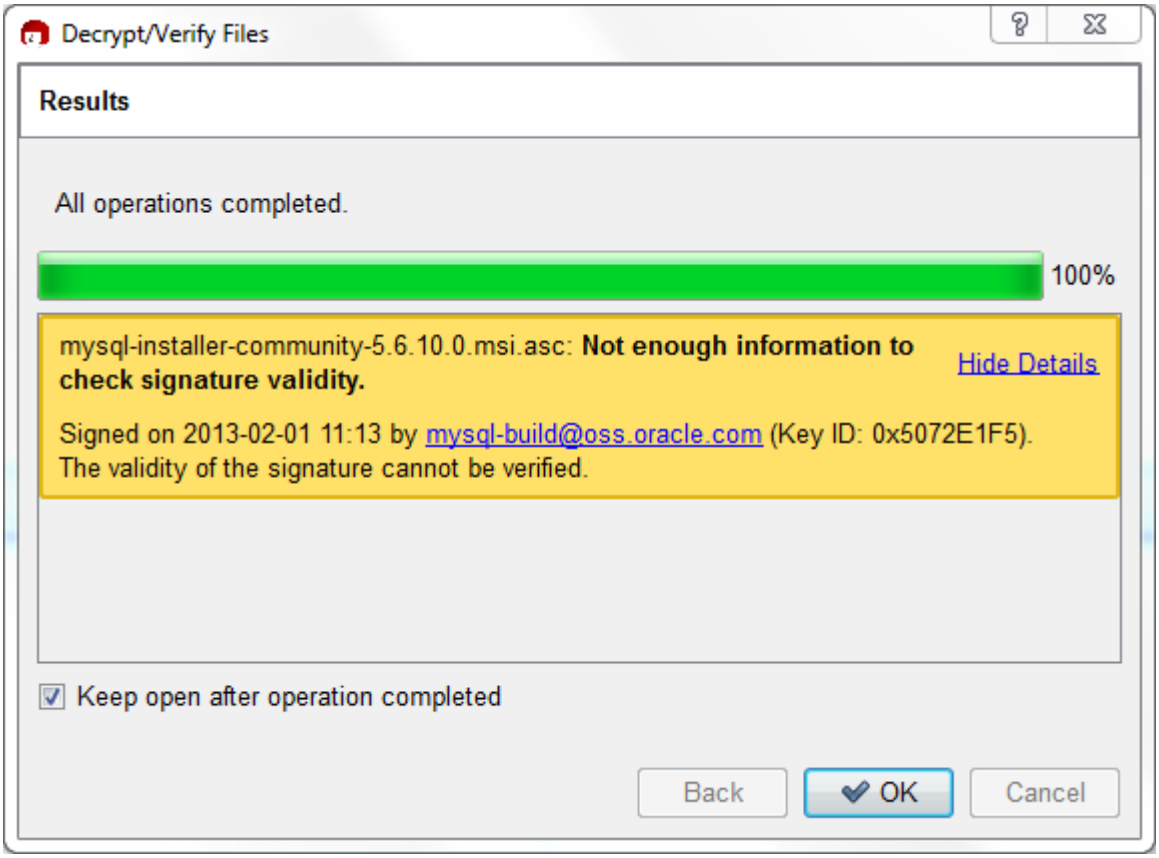
Figure 2.4 The Decrypt/Verify Files dialog



Click **Decrypt/Verify** to check the file. The two most common results will look like the following, and although the yellow warning looks problematic, the following means that the file check passed with success. You may now run this installer.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

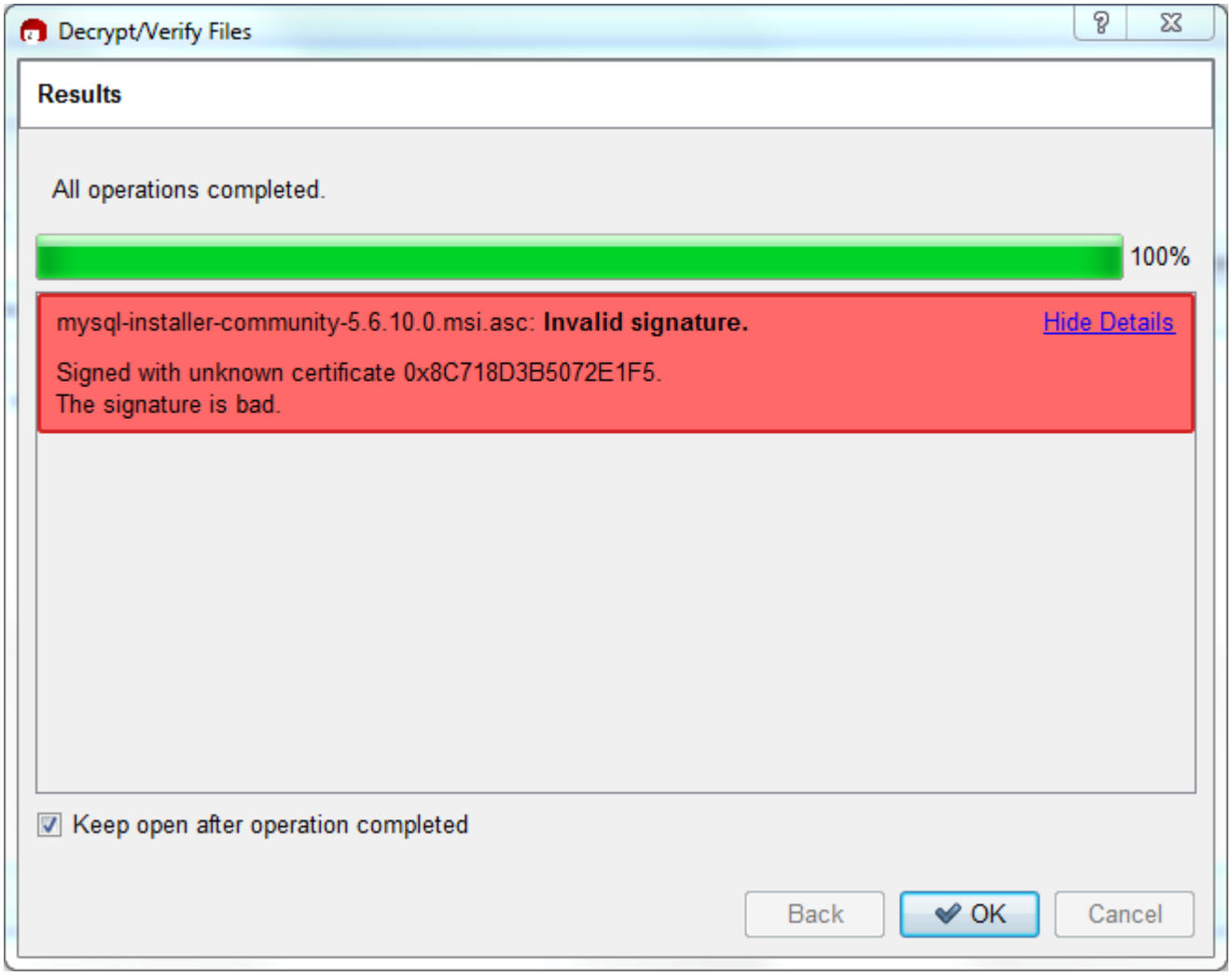
Figure 2.5 The Decrypt/Verify Results: Good



Seeing a red "The signature is bad" error means the file is invalid. Do not execute the MSI file if you see this error.

This documentation is for an older version. If you're

Figure 2.6 The Decrypt/Verify Results: Bad



The Section 2.6.2, “Signature Checking Using GnuPG” section explains why you probably don't see a green `Good signature` result.

2.6.4 Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-5.0.96-0.glibc23.i386.rpm
MySQL-server-5.0.96-0.glibc23.i386.rpm: md5 gpg OK
```

Note

If you are using RPM 4.1 and it complains about `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, even though you have imported the MySQL public build

This documentation is for an older version. If you're

key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, RPM maintains a separate keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key, then use `rpm --import` to import the key. For example:

```
shell> gpg --export -a 5072e1f5 > 5072e1f5.asc
shell> rpm --import 5072e1f5.asc
```

Alternatively, `rpm` also supports loading the key directly from a URL, and you can use this manual page:

```
shell> rpm --import http://dev.mysql.com/doc/refman/5.0/en/checking-gpg-signature.html
```

If you need to obtain the MySQL public key, see Section 2.6.2, “Signature Checking Using GnuPG”.

2.7 Installation Layouts

This section describes the default layout of the directories created by installing binary or source distributions provided by Oracle Corporation. A distribution provided by another vendor might use a layout different from those shown here.

For MySQL 5.0 on Windows, the default installation directory is `C:\Program Files\MySQL\MySQL Server 5.0`. (Some Windows users prefer to install in `C:\mysql`, the directory that formerly was used as the default. However, the layout of the subdirectories remains the same.) The installation directory has the following subdirectories:

Table 2.3 MySQL Installation Layout for Windows

Directory	Contents of Directory
`bin`	Client programs and the `mysqld` server
`data`	Log files, databases
`examples`	Example programs and scripts
`include`	Include (header) files
`lib`	Libraries
`scripts`	Utility scripts
`share`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation

Installations created from our Linux RPM distributions result in files under the system directories shown in the following table.

Table 2.4 MySQL Installation Layout for Linux RPM Packages

Directory	Contents of Directory
`/usr/bin`	Client programs and scripts
`/usr/sbin`	The `mysqld` server
`/var/lib/mysql`	Log files, databases
`/usr/share/info`	MySQL manual in Info format

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Directory	Contents of Directory
`/usr/share/man`	Unix man pages
`/usr/include/mysql`	Include (header) files
`/usr/lib/mysql`	Libraries
`/usr/share/mysql`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
`/usr/share/sql-bench`	Benchmarks

On Unix, a `tar` file binary distribution is installed by unpacking it at the installation location you choose (typically `/usr/local/mysql`) and creates the following directories in that location:

Table 2.5 MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
`bin`	Client programs and the `mysqld` server
`data`	Log files, databases
`docs`	MySQL manual in Info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`scripts`	`mysql_install_db`
`share/mysql`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
`sql-bench`	Benchmarks

By default, when you install MySQL after compiling it from a source distribution, the installation step installs files under `/usr/local`. Components are installed in the directories shown in the following table. To configure particular installation locations, use the options described at Section 2.17.3, “MySQL Source-Configuration Options”.

Table 2.6 MySQL Layout for Installation from Source

Directory	Contents of Directory
`bin`	Client programs and scripts
`include/mysql`	Include (header) files
`Docs`	MySQL manual in Info format
`man`	Unix manual pages
`lib/mysql`	Libraries
`libexec`	The `mysqld` server
`share/mysql`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
`sql-bench`	Benchmarks
`var`	Log files, databases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Within its installation directory, the layout of a source installation differs from that of a binary installation in the following ways:

- The `mysqld` server is installed in the `libexec` directory rather than in the `bin` directory.
- The data directory is `var` rather than `data`.
- `mysql_install_db` is installed in the `bin` directory rather than in the `scripts` directory.
- The header file and library directories are `include/mysql` and `lib/mysql` rather than `include` and `lib`.

To create your own binary installation from a compiled source distribution, execute the `scripts/make_binary_distribution` script from the top directory of the source distribution.

2.8 Compiler-Specific Build Characteristics

In some cases, the compiler used to build MySQL affects the features available for use. The notes in this section apply for binary distributions provided by Oracle Corporation or that you compile yourself from source.

`icc` (Intel C++ Compiler) Builds

A server built with `icc` has these characteristics:

- SSL support is not included.

2.9 Standard MySQL Installation from a Binary Distribution

The next several sections cover the installation of MySQL on platforms where we offer packages using the native packaging format of the respective platform. (This is also known as performing a binary installation.) However, binary distributions of MySQL are available for many other platforms as well. See Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”, for generic installation instructions for these packages that apply to all platforms.

See Section 2.4, “Notes for MySQL Community Server”, for more information on what other binary distributions are available and how to obtain them.

2.10 Installing MySQL on Microsoft Windows

Important

The MySQL server 5.0 branch is old and not recommended for new installations. Consider installing the latest stable branch, which today is MySQL server 5.7.

A native Windows distribution of MySQL has been available since version 3.21 and represents a sizable percentage of the daily downloads of MySQL. This section describes the process for installing MySQL on Windows.

Note

If you are upgrading MySQL from an existing installation older than MySQL 4.1.5, you must first perform the procedure described in Section 2.10.7, “Upgrading MySQL on Windows”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To run MySQL on Windows, you need the following:

- A Windows operating system such as XP, Vista, and Server 2003. Newer versions of Windows than these are not supported. Windows 95/98/ME/2000 and versions of Windows older than these are no longer supported. For supported platform information, see http://www.mysql.com/support/supportedplatforms/database.html.

  A Windows operating system permits you to run the MySQL server as a service. See Section 2.10.4.7, "Starting MySQL as a Windows Service".

  Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the `Service Control Manager`.

- TCP/IP protocol support.

- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements (generally a minimum of 200 megabytes is recommended.)

For a list of limitations on the use of MySQL on the Windows platform, see Section C.7.6, "Windows Platform Limitations".

There may also be other requirements, depending on how you plan to use MySQL:

- To connect to the MySQL server using ODBC, you must have a Connector/ODBC driver. See Chapter 20, *Connectors and APIs*.

- If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer file system. Do not forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See Section 13.1.10, "CREATE TABLE Syntax".

MySQL for Windows is available in several distribution formats:

- Binary distributions are available that contain a setup program that installs everything you need so that you can start the server immediately. Another binary distribution format contains an archive that you simply unpack in the installation location and then configure yourself. For details, see Section 2.10.1, "Choosing An Installation Package".

- The source distribution format contains all the code and support files for building the executables using the Visual Studio compiler system.

Generally speaking, you should use a binary distribution that includes an installer. It is simpler to use than the others, and you need no additional tools to get MySQL up and running. The installer for the Windows version of MySQL, combined with a GUI Configuration Wizard, automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

Caution

Virus-scanning software such as Norton/Symantec Anti-Virus on directories containing MySQL data and temporary tables can cause issues, both in terms of the performance of MySQL and the virus-scanning software misidentifying the contents of the files as containing spam. This is due to the fingerprinting mechanism used by the virus-scanning software, and the way in which MySQL rapidly updates different files, which may be identified as a potential security risk.

After installing MySQL Server, it is recommended that you disable virus scanning on the main directory (`datadir`) used to store your MySQL table data. There is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

> usually a system built into the virus scanning software to enable specific directories to be ignored.
>
> In addition, by default, MySQL creates temporary files in the standard Windows temporary directory. To prevent the temporary files also being scanned, configure a separate temporary directory for MySQL temporary files and add this directory to the virus scanning exclusion list. To do this, add a configuration option for the `tmpdir` parameter to your `my.ini` configuration file. For more information, see Section 2.10.4.2, “Creating an Option File”.

The following section describes how to install MySQL on Windows using a binary distribution. To use an installation package that does not include an installer, follow the procedure described in Section 2.10.4, “Installing MySQL on Microsoft Windows Using a noinstall Zip Archive”. To install using a source distribution, see Section 2.10.8, “Installing MySQL from Source on Windows”.

MySQL distributions for Windows can be downloaded from http://dev.mysql.com/downloads/. See Section 2.5, “How to Get MySQL”.

2.10.1 Choosing An Installation Package

For MySQL 5.0, there are multiple installation package formats to choose from when installing MySQL on Windows.

- **The Essentials package.** This package has a file name similar to `mysql-essential-5.0.96-win32.msi` and contains the minimum set of files needed to install MySQL on Windows, including the Configuration Wizard. This package does not include optional components such as the embedded server and benchmark suite.
- **The Complete package.** This package has a file name similar to `mysql-5.0.96-win32.zip` and contains all files needed for a complete Windows installation, including the Configuration Wizard. This package includes optional components such as the embedded server and benchmark suite.
- **The no-install archive.** This package has a file name similar to `mysql-noinstall-5.0.96-win32.zip` and contains all the files found in the Complete install package, with the exception of the Configuration Wizard. This package does not include an automated installer, and must be manually installed and configured.

The Essentials package is recommended for most users. It is provided as an `.msi` file for use with the Windows Installer. The Complete and Noinstall distributions are packaged as Zip archives. To use them, you must have a tool that can unpack `.zip` files.

Your choice of install package affects the installation process you must follow. If you choose to install either an Essentials or Complete install package, see Section 2.10.2, “Installing MySQL on Microsoft Windows Using an MSI Package”. If you choose to install a Noinstall archive, see Section 2.10.4, “Installing MySQL on Microsoft Windows Using a noinstall Zip Archive”.

2.10.2 Installing MySQL on Microsoft Windows Using an MSI Package

New MySQL users can use the MySQL Installation Wizard and MySQL Configuration Wizard to install MySQL on Windows. These are designed to install and configure MySQL in such a way that new users can immediately get started using MySQL.

The MySQL Installation Wizard and MySQL Configuration Wizard are available in the Essentials and Complete install packages. They are recommended for most standard MySQL installations. Exceptions include users who need to install multiple instances of MySQL on a single server host and advanced users who want complete control of server configuration.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.10.2.1 Using the MySQL Installation Wizard

MySQL Installation Wizard is an installer for the MySQL server that uses the latest installer technologies for Microsoft Windows. The MySQL Installation Wizard, in combination with the MySQL Configuration Wizard, enables a user to install and configure a MySQL server that is ready for use immediately after installation.

The MySQL Installation Wizard is the standard installer for all MySQL server distributions, version 4.1.5 and higher. Users of previous versions of MySQL need to shut down and remove their existing MySQL installations manually before installing MySQL with the MySQL Installation Wizard. See Upgrading MySQL with the Installation Wizard, for more information on upgrading from a previous version.

The Microsoft Windows Installer (MSI) is the standard for application installations on Windows 2000 and later versions. The MySQL Installation Wizard makes use of this technology to provide a smoother and more flexible installation process.

The Microsoft Windows Installer Engine was updated with the release of Windows XP; those using a previous version of Windows can reference this Microsoft Knowledge Base article for information on upgrading to the latest version of the Windows Installer Engine.

In addition, Microsoft has introduced the WiX (Windows Installer XML) toolkit, which is the first highly acknowledged Open Source project from Microsoft. We have switched to WiX because it is an Open Source project and it enables us to handle the complete Windows installation process in a flexible manner using scripts.

Improving the MySQL Installation Wizard depends on the support and feedback of users. If you find that the MySQL Installation Wizard is lacking some feature important to you, or if you discover a bug, please report it in our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”.

Downloading and Starting the MySQL Installation Wizard

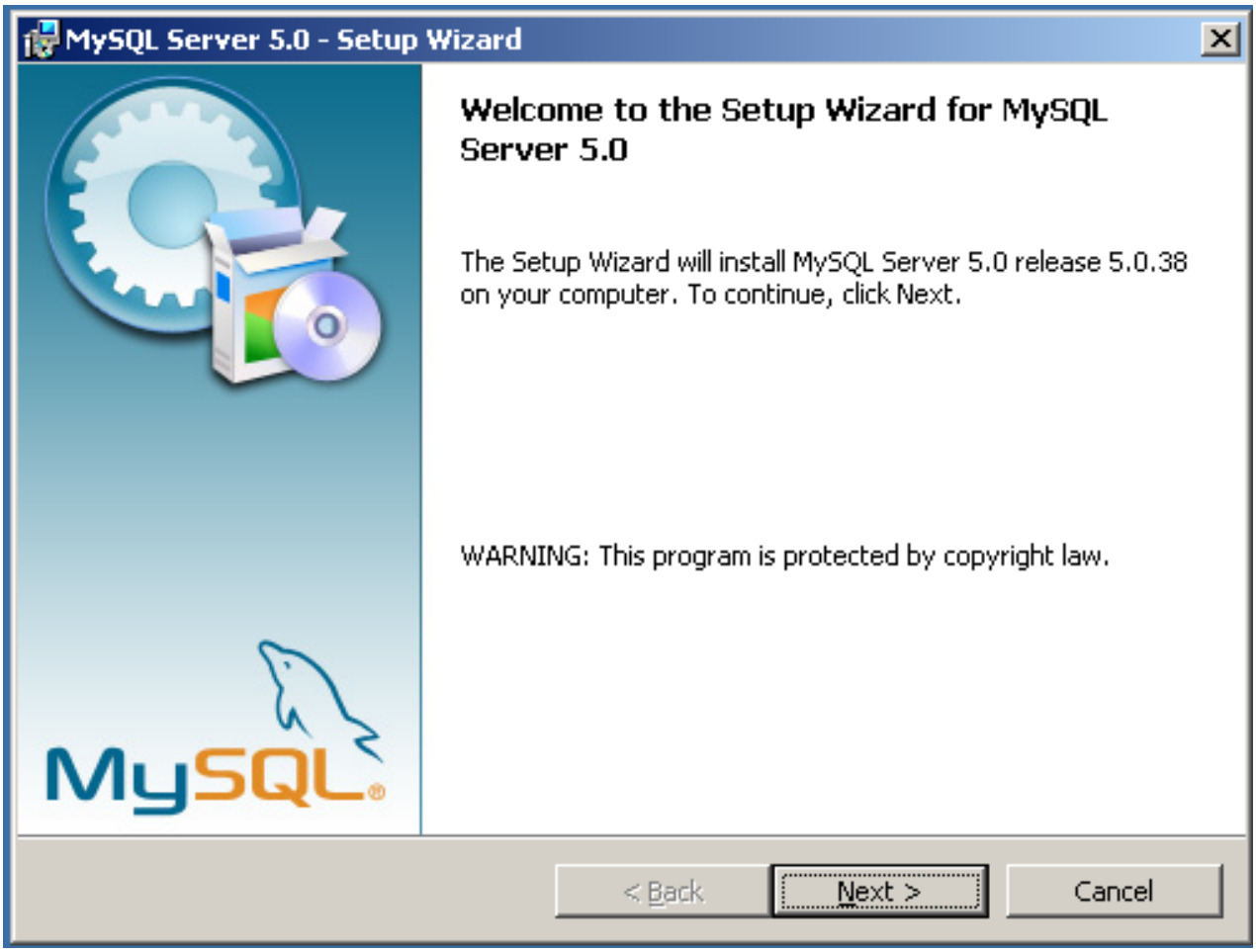
MySQL installation packages can be downloaded from http://dev.mysql.com/downloads/. If the package you download is contained within a Zip archive, you need to extract the archive first.

Note

If you are installing on Windows Vista or newer, it is best to open a network port for MySQL to use before beginning the installation. To do this, first ensure that you are logged in as an Administrator, then go to the `Control Panel` and double-click the `Windows Firewall` icon. Choose the `Allow a program through Windows Firewall` option and click the **Add port** button. Enter `MySQL` into the **Name** text box and `3306` (or other port of your choice) into the **Port number** text box. Also ensure that the **TCP** protocol radio button is selected. If you wish, you can also limit access to the MySQL server by choosing the **Change scope** button. Confirm your choices by clicking the **OK** button. If you do not open a port prior to installation, you cannot configure the MySQL server immediately after installation. Additionally, when running the MySQL Installation Wizard on Windows Vista or newer, ensure that you are logged in as a user with administrative rights.

The process for starting the wizard depends on the contents of the installation package you download. If there is a `setup.exe` file present, double-click it to start the installation process. If there is an `.msi` file present, double-click it to start the installation process.

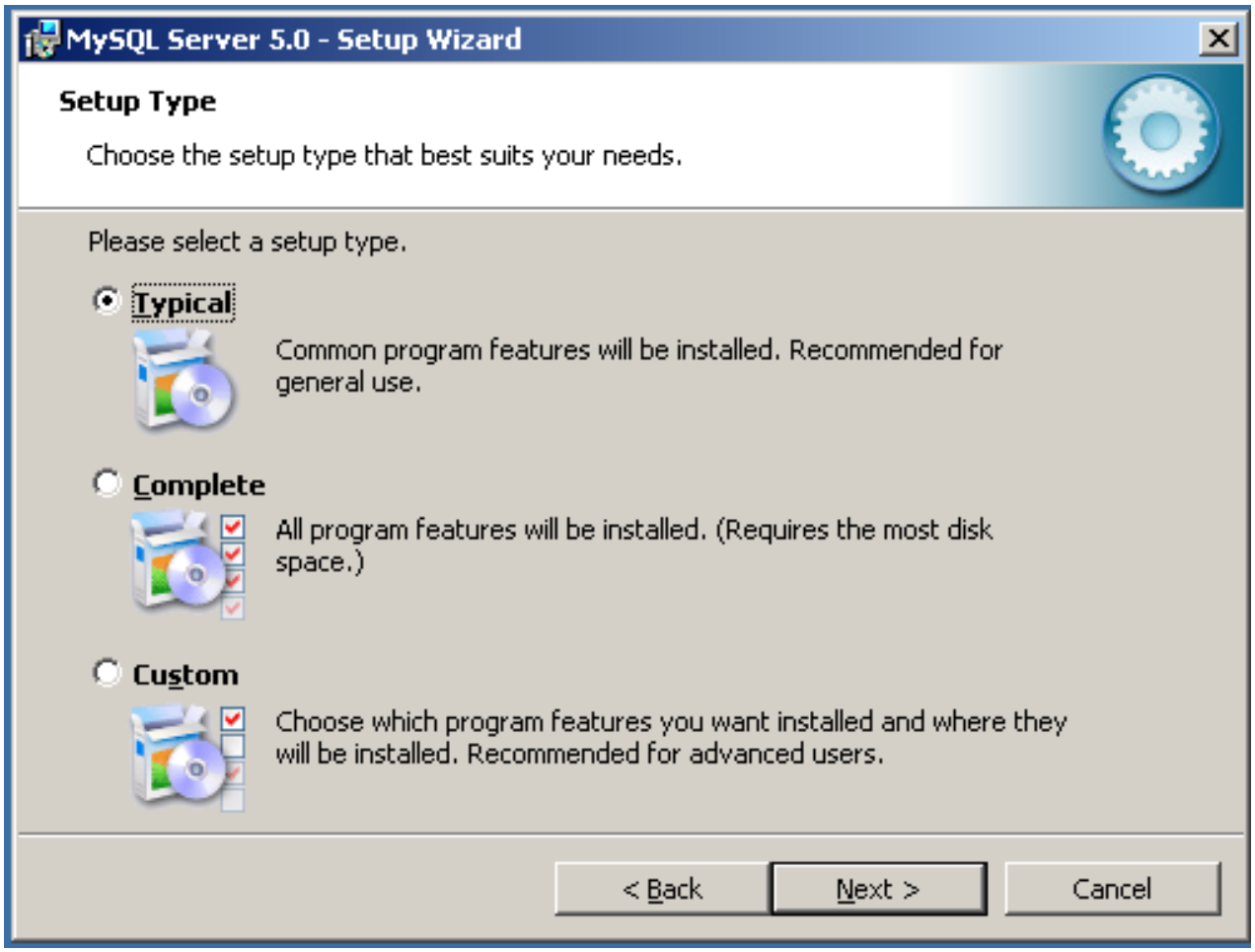
This documentation is for an older version. If you're
This documentation is for an older version. If you're



Choosing an Installation Type

There are three installation types available: **Typical**, **Complete**, and **Custom**.

This documentation is for an older version. If you're



The **Typical** installation type installs the MySQL server, the `mysql` command-line client, and the command-line utilities. The command-line clients and utilities include `mysqldump`, `myisamchk`, and several other tools to help you manage the MySQL server.

The **Complete** installation type installs all components included in the installation package. The full installation package includes components such as the embedded server library, the benchmark suite, support scripts, and documentation.

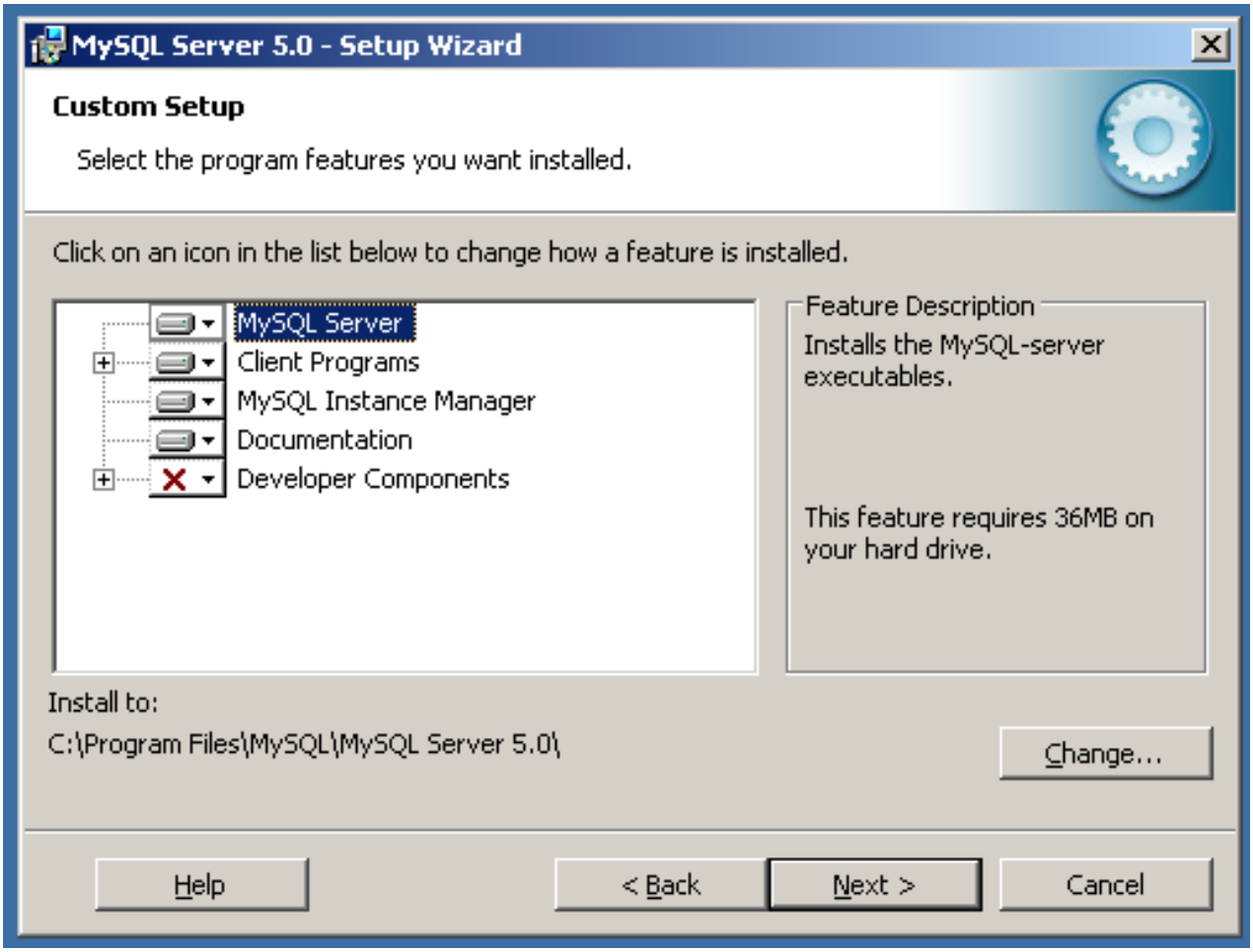
The **Custom** installation type gives you complete control over which packages you wish to install and the installation path that is used. See The Custom Installation Dialog, for more information on performing a custom install.

If you choose the **Typical** or **Complete** installation types and click the **Next** button, you advance to the confirmation screen to verify your choices and begin the installation. If you choose the **Custom** installation type and click the **Next** button, you advance to the custom installation dialog, described in The Custom Installation Dialog.

The Custom Installation Dialog

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, choose the **Custom** installation type.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



A tree view on the left side of the custom install dialog lists all available components. Components that are not installed have a red **X** icon; components that are installed have a gray icon. To change whether a component is installed, click that component's icon and choose a new option from the drop-down list that appears.

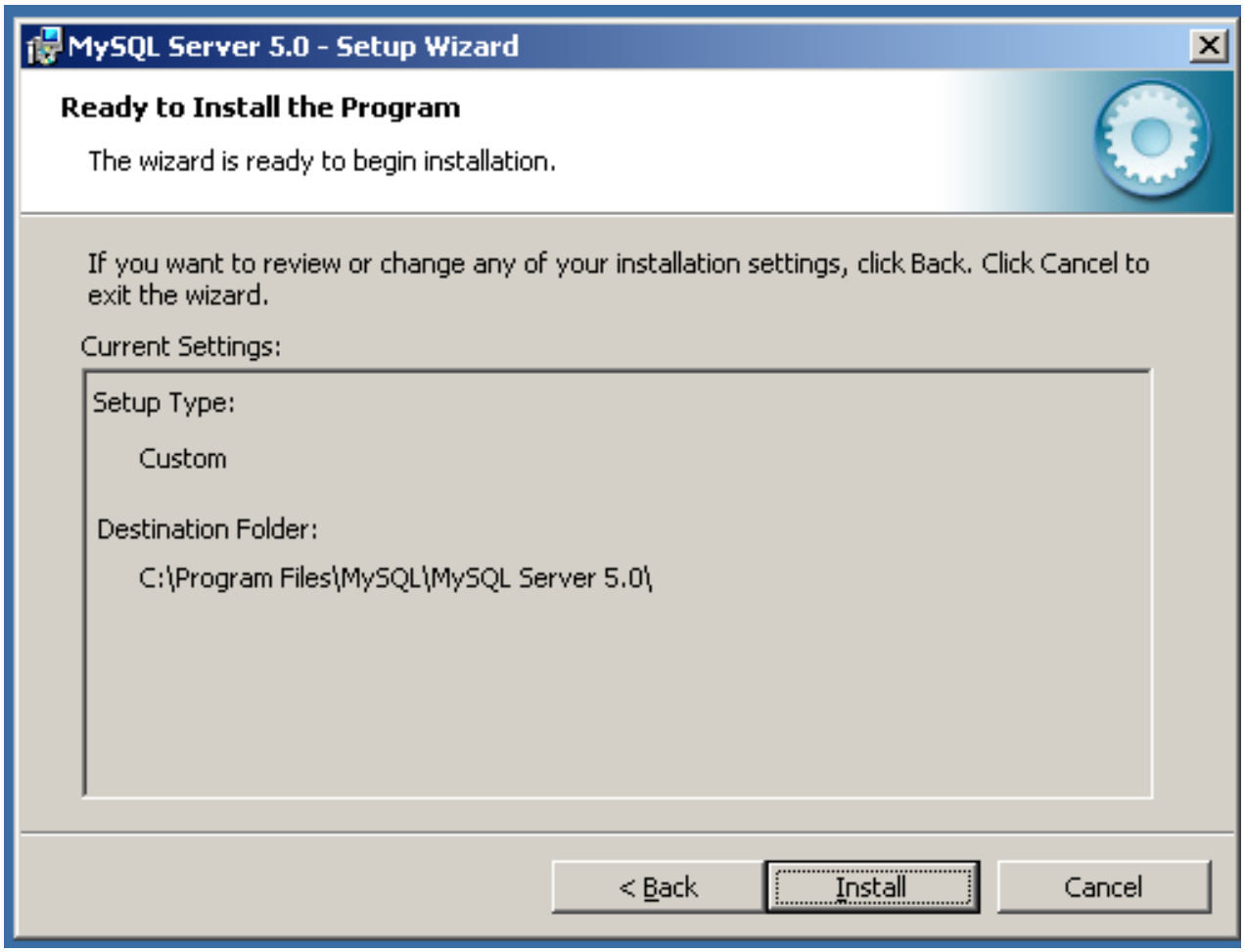
You can change the default installation path by clicking the **Change...** button to the right of the displayed installation path.

After choosing your installation components and installation path, click the **Next** button to advance to the confirmation dialog.

The Confirmation Dialog

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



To install MySQL if you are satisfied with your settings, click the **Install** button. To change your settings, click the **Back** button. To exit the MySQL Installation Wizard without installing MySQL, click the **Cancel** button.

After installation is complete, you have the option of registering with the MySQL Web site. Registration gives you access to post in the MySQL forums at forums.mysql.com, along with the ability to report bugs at bugs.mysql.com and to subscribe to our newsletter. The final screen of the installer provides a summary of the installation and gives you the option to launch the MySQL Configuration Wizard, which you can use to create a configuration file, install the MySQL service, and configure security settings.

Changes Made by MySQL Installation Wizard

Once you click the **Install** button, the MySQL Installation Wizard begins the installation process and makes certain changes to your system which are described in the sections that follow.

Changes to the Registry

The MySQL Installation Wizard creates one Windows registry key in a typical install situation, located in `HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB`.

The MySQL Installation Wizard creates a key named after the release series of the server that is being installed, such as `MySQL Server 5.0`. It contains two string values, `Location` and `Version`. The `Location` string contains the path to the installation directory. In a default installation it contains `C:`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`\Program Files\MySQL\MySQL Server 5.0\`. The `Version` string contains the release number. For example, for an installation of MySQL Server 5.0.96, the key contains a value of `5.0.96`.

These registry keys are used to help external tools identify the installed location of the MySQL server, preventing a complete scan of the hard-disk to determine the installation path of the MySQL server. The registry keys are not required to run the server, and if you install MySQL using the `noinstall` Zip archive, the registry keys are not created.

Changes to the Start Menu

The MySQL Installation Wizard creates a new entry in the Windows **Start** menu under a common MySQL menu heading named after the release series of MySQL that you have installed. For example, if you install MySQL 5.0, the MySQL Installation Wizard creates a **MySQL Server 5.0** section in the **Start** menu.

The following entries are created within the new **Start** menu section:

- **MySQL Command-Line Client**: This is a shortcut to the `mysql` command-line client and is configured to connect as the `root` user. The shortcut prompts for a `root` user password when you connect.
- **MySQL Server Instance Config Wizard**: This is a shortcut to the MySQL Configuration Wizard. Use this shortcut to configure a newly installed server, or to reconfigure an existing server.
- **MySQL Documentation**: This is a link to the MySQL server documentation that is stored locally in the MySQL server installation directory. This option is not available when the MySQL server is installed using the Essentials installation package.

Changes to the File System

The MySQL Installation Wizard by default installs the MySQL 5.0 server to `C:\Program Files\MySQL\MySQL Server 5.0`, where `Program Files` is the default location for applications in your system, and `5.0` is the major version of your MySQL server. This is the recommended location for the MySQL server, replacing the former default location `C:\mysql`.

By default, all MySQL applications are stored in a common directory at `C:\Program Files\MySQL`, where `Program Files` is the default location for applications in your Windows installation. A typical MySQL installation on a developer machine might look like this:

```
C:\Program Files\MySQL\MySQL Server 5.0
C:\Program Files\MySQL\MySQL Workbench 5.1 OSS
```

This approach makes it easier to manage and maintain all MySQL applications installed on a particular system.

Upgrading MySQL with the Installation Wizard

The MySQL Installation Wizard can perform server upgrades automatically using the upgrade capabilities of MSI. That means you do not need to remove a previous installation manually before installing a new release. The installer automatically shuts down and removes the previous MySQL service before installing the new version.

Automatic upgrades are available only when upgrading between installations that have the same major and minor version numbers. For example, you can upgrade automatically from MySQL 5.0.5 to MySQL 5.0.6, but not from MySQL 4.1 to MySQL 5.0.

See Section 2.10.7, “Upgrading MySQL on Windows”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.10.3 MySQL Server Instance Configuration Wizard

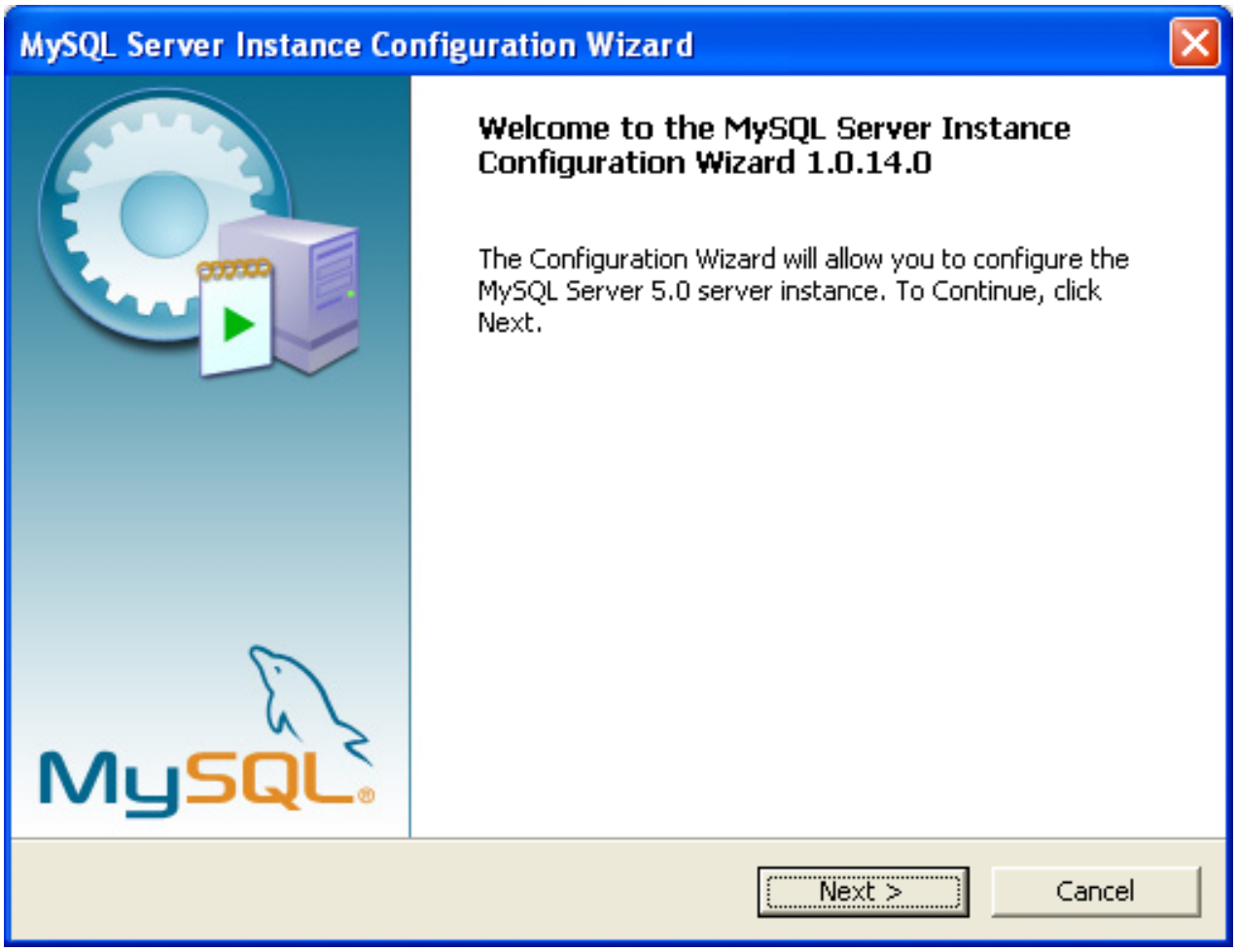
The MySQL Server Instance Configuration Wizard helps automate the process of configuring your server. It creates a custom MySQL configuration file (`my.ini` or `my.cnf`) by asking you a series of questions and then applying your responses to a template to generate the configuration file that is tuned to your installation.

The MySQL Server Instance Configuration Wizard is included with the MySQL 5.0 server. The MySQL Server Instance Configuration Wizard is only available for Windows.

2.10.3.1 Starting the MySQL Server Instance Configuration Wizard

The MySQL Server Instance Configuration Wizard is normally started as part of the installation process. You should only need to run the MySQL Server Instance Configuration Wizard again when you need to change the configuration parameters of your server.

If you chose not to open a port prior to installing MySQL on Windows Vista or newer, you can choose to use the MySQL Server Configuration Wizard after installation. However, you must open a port in the Windows Firewall. To do this see the instructions given in Downloading and Starting the MySQL Installation Wizard. Rather than opening a port, you also have the option of adding MySQL as a program that bypasses the Windows Firewall. One or the other option is sufficient—you need not do both. Additionally, when running the MySQL Server Configuration Wizard on Windows Vista or newer, ensure that you are logged in as a user with administrative rights.



This documentation is for an older version. If you're

You can launch the MySQL Configuration Wizard by clicking the **MySQL Server Instance Config Wizard** entry in the **MySQL** section of the Windows **Start** menu.

Alternatively, you can navigate to the `bin` directory of your MySQL installation and launch the `MySQLInstanceConfig.exe` file directly.

The MySQL Server Instance Configuration Wizard places the `my.ini` file in the installation directory for the MySQL server. This helps associate configuration files with particular server instances.

To ensure that the MySQL server knows where to look for the `my.ini` file, an argument similar to this is passed to the MySQL server as part of the service installation:

```
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.0\my.ini"
```

Here, *`C:\Program Files\MySQL\MySQL Server 5.0`* is replaced with the installation path to the MySQL Server. The `--defaults-file` option instructs the MySQL server to read the specified file for configuration options when it starts.

Apart from making changes to the `my.ini` file by running the MySQL Server Instance Configuration Wizard again, you can modify it by opening it with a text editor and making any necessary changes. You can also modify the server configuration with the http://www.mysql.com/products/administrator/ utility. For more information about server configuration, see Section 5.1.3, “Server Command Options”.

MySQL clients and utilities such as the `mysql` and `mysqldump` command-line clients are not able to locate the `my.ini` file located in the server installation directory. To configure the client and utility applications, create a new `my.ini` file in the Windows installation directory (for example, `C:\WINDOWS`).

Under Windows Server 2003, Windows Server 2000 and Windows XP, MySQL Server Instance Configuration Wizard will configure MySQL to work as a Windows service. To start and stop MySQL you use the `Services` application that is supplied as part of the Windows Administrator Tools.

2.10.3.2 Choosing a Maintenance Option

If the MySQL Server Instance Configuration Wizard detects an existing configuration file, you have the option of either reconfiguring your existing server, or removing the server instance by deleting the configuration file and stopping and removing the MySQL service.

To reconfigure an existing server, choose the **Re-configure Instance** option and click the **Next** button. Any existing configuration file is not overwritten, but renamed (within the same directory) using a timestamp (Windows) or sequential number (Linux). To remove the existing server instance, choose the **Remove Instance** option and click the **Next** button.

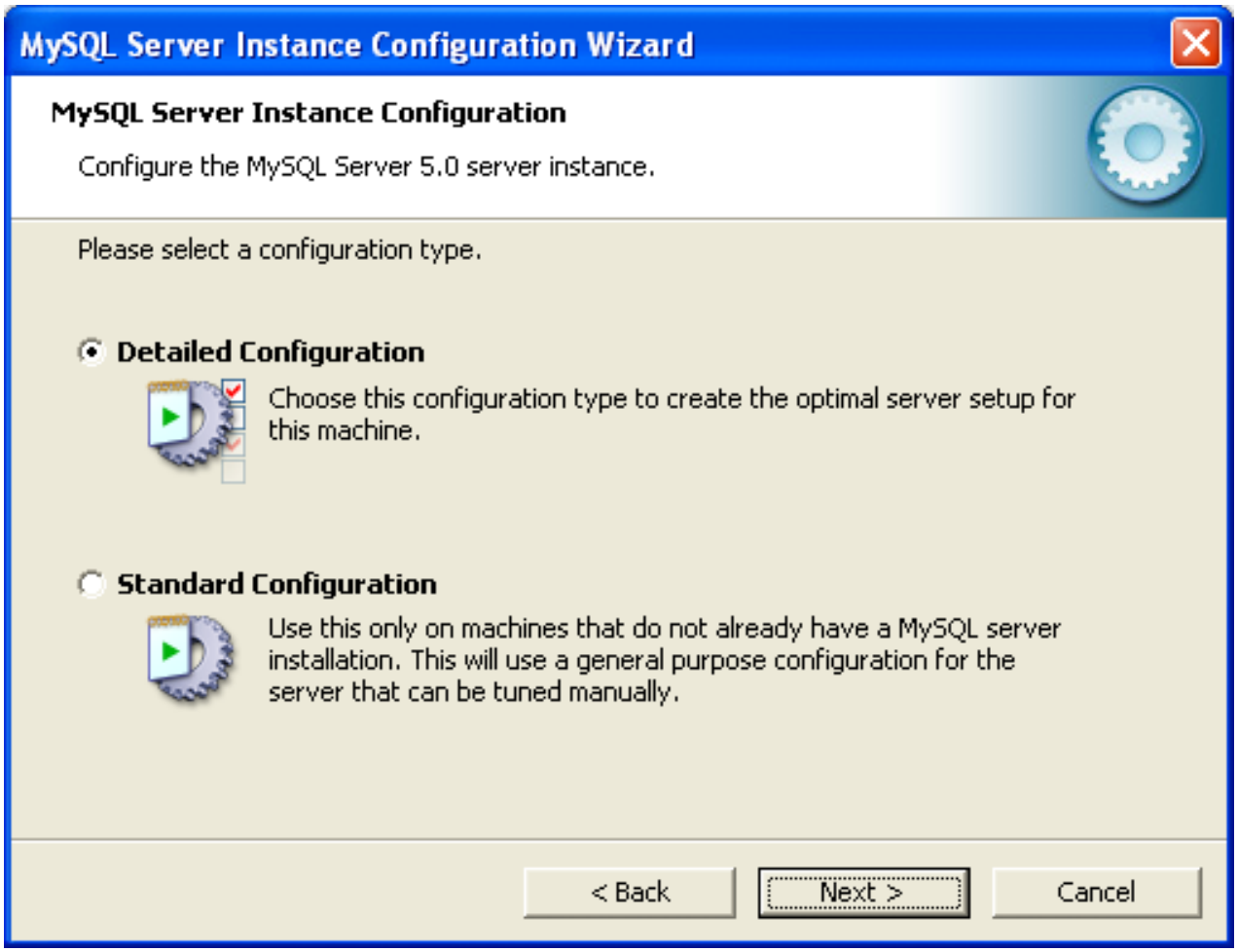
If you choose the **Remove Instance** option, you advance to a confirmation window. Click the **Execute** button. The MySQL Server Configuration Wizard stops and removes the MySQL service, and then deletes the configuration file. The server installation and its `data` folder are not removed.

If you choose the **Re-configure Instance** option, you advance to the **Configuration Type** dialog where you can choose the type of installation that you wish to configure.

2.10.3.3 Choosing a Configuration Type

When you start the MySQL Server Instance Configuration Wizard for a new MySQL installation, or choose the **Re-configure Instance** option for an existing installation, you advance to the **Configuration Type** dialog.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



There are two configuration types available: **Detailed Configuration** and **Standard Configuration**. The **Standard Configuration** option is intended for new users who want to get started with MySQL quickly without having to make many decisions about server configuration. The **Detailed Configuration** option is intended for advanced users who want more fine-grained control over server configuration.

If you are new to MySQL and need a server configured as a single-user developer machine, the **Standard Configuration** should suit your needs. Choosing the **Standard Configuration** option causes the MySQL Configuration Wizard to set all configuration options automatically with the exception of **Service Options** and **Security Options**.

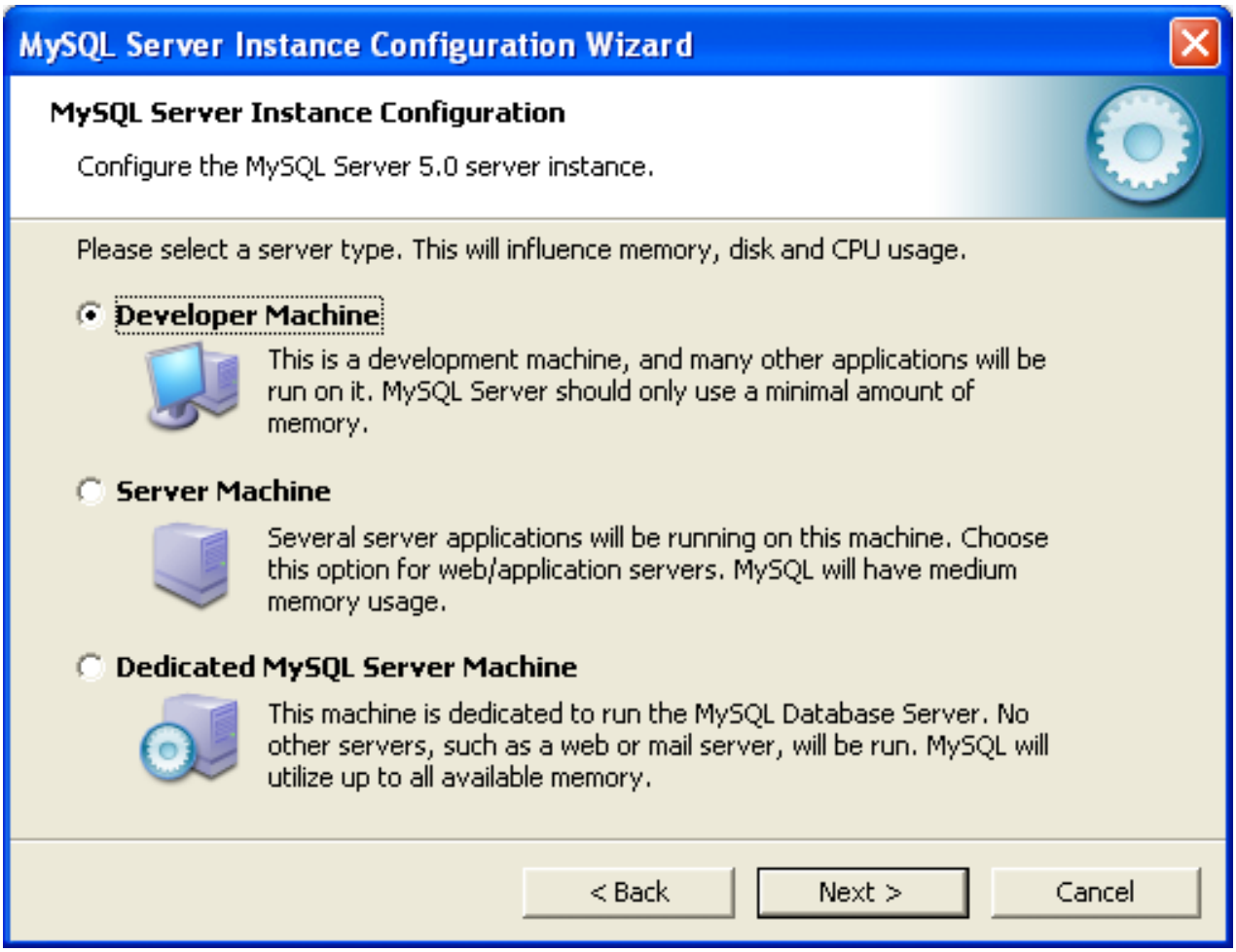
The **Standard Configuration** sets options that may be incompatible with systems where there are existing MySQL installations. If you have an existing MySQL installation on your system in addition to the installation you wish to configure, the **Detailed Configuration** option is recommended.

To complete the **Standard Configuration**, please refer to the sections on **Service Options** and **Security Options** in Section 2.10.3.10, “The Service Options Dialog”, and Section 2.10.3.11, “The Security Options Dialog”, respectively.

2.10.3.4 The Server Type Dialog

There are three different server types available to choose from. The server type that you choose affects the decisions that the MySQL Server Instance Configuration Wizard makes with regard to memory, disk, and processor usage.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



- **Developer Machine**: Choose this option for a typical desktop workstation where MySQL is intended only for personal use. It is assumed that many other desktop applications are running. The MySQL server is configured to use minimal system resources.
- **Server Machine**: Choose this option for a server machine where the MySQL server is running alongside other server applications such as FTP, email, and Web servers. The MySQL server is configured to use a moderate portion of the system resources.
- **Dedicated MySQL Server Machine**: Choose this option for a server machine that is intended to run only the MySQL server. It is assumed that no other applications are running. The MySQL server is configured to use all available system resources.

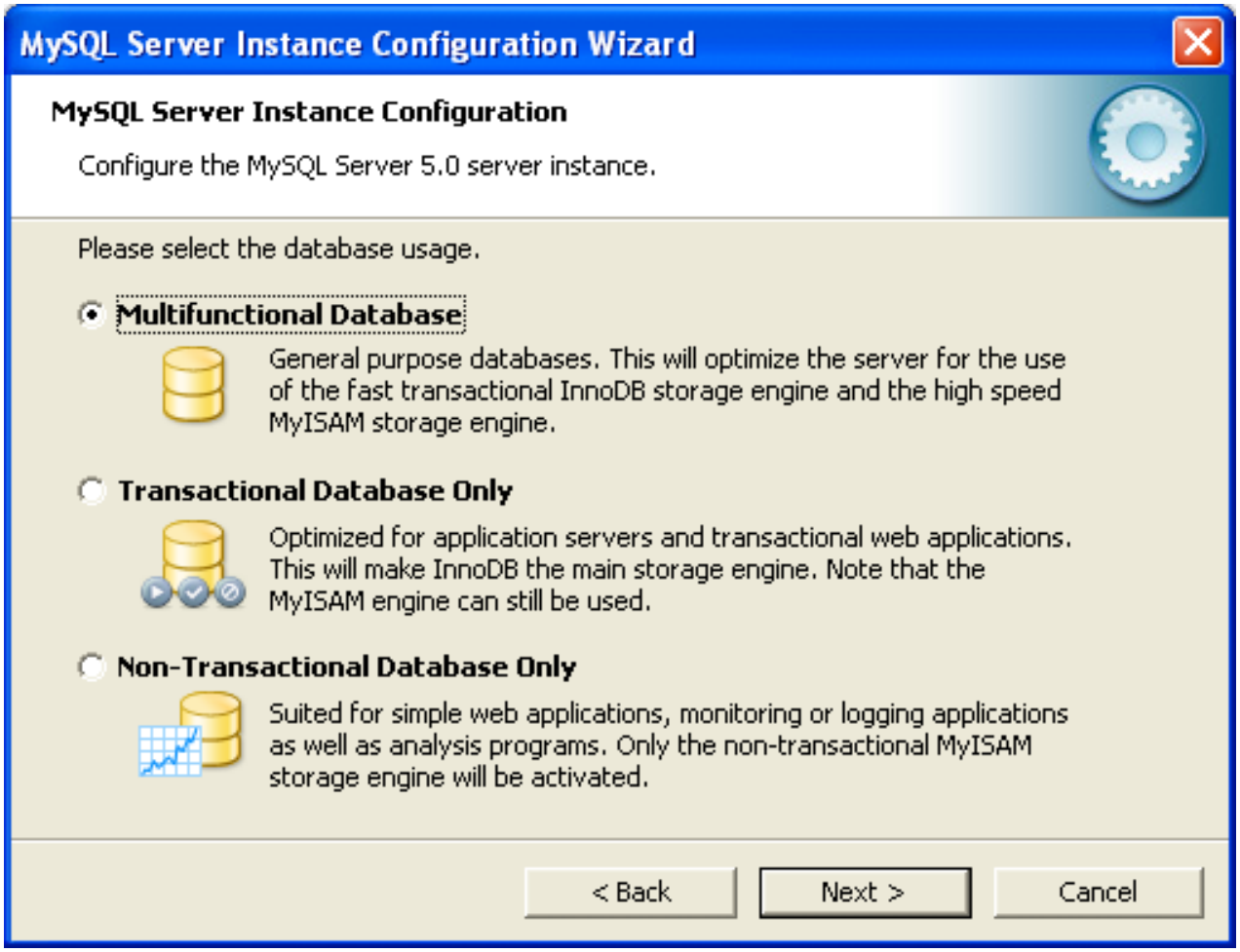
Note

By selecting one of the preconfigured configurations, the values and settings of various options in your `my.cnf` or `my.ini` will be altered accordingly. The default values and options as described in the reference manual may therefore be different to the options and values that were created during the execution of the configuration wizard.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.10.3.5 The Database Usage Dialog

The **Database Usage** dialog enables you to indicate the storage engines that you expect to use when creating MySQL tables. The option you choose determines whether the `InnoDB` storage engine is available and what percentage of the server resources are available to `InnoDB`.



- **Multifunctional Database**: This option enables both the `InnoDB` and `MyISAM` storage engines and divides resources evenly between the two. This option is recommended for users who use both storage engines on a regular basis.
- **Transactional Database Only**: This option enables both the `InnoDB` and `MyISAM` storage engines, but dedicates most server resources to the `InnoDB` storage engine. This option is recommended for users who use `InnoDB` almost exclusively and make only minimal use of `MyISAM`.
- **Non-Transactional Database Only**: This option disables the `InnoDB` storage engine completely and dedicates all server resources to the `MyISAM` storage engine. This option is recommended for users who do not use `InnoDB`.

The Configuration Wizard uses a template to generate the server configuration file. The **Database Usage** dialog sets one of the following option strings:

```
Multifunctional Database:        MIXED
Transactional Database Only:     INNODB
Non-Transactional Database Only: MYISAM
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

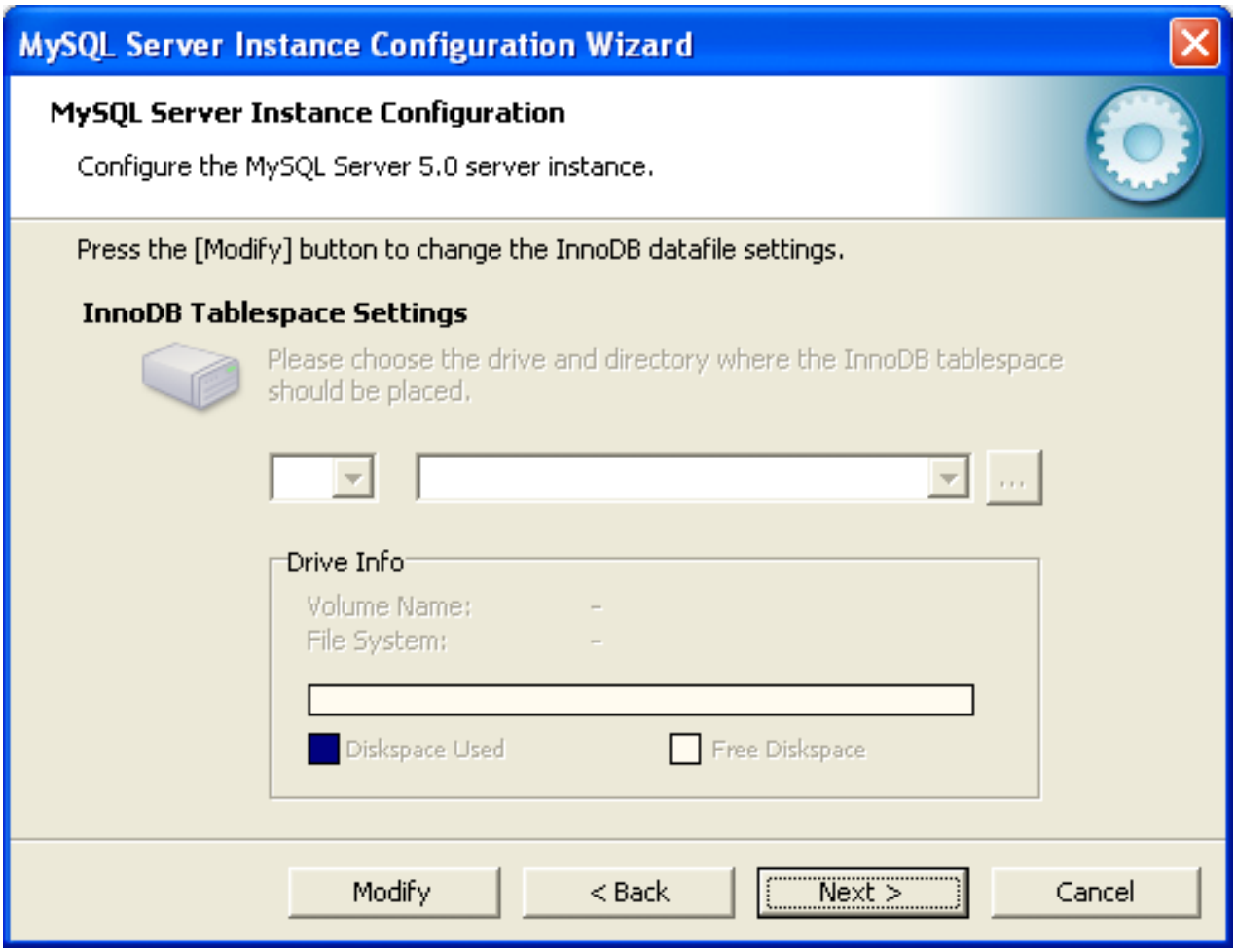
When these options are processed through the default template (my-template.ini) the result is:

```
Multifunctional Database:
default-storage-engine=InnoDB
_myisam_pct=50

Transactional Database Only:
default-storage-engine=InnoDB
_myisam_pct=5

Non-Transactional Database Only:
default-storage-engine=MyISAM
_myisam_pct=100
skip-innodb
```

The `_myisam_pct` value is used to calculate the percentage of resources dedicated to `MyISAM`. The remaining resources are allocated to `InnoDB`.

2.10.3.6 The InnoDB Tablespace Dialog

Some users may want to locate the `InnoDB` tablespace files in a different location than the MySQL server data directory. Placing the tablespace files in a separate location can be desirable if your system has a higher capacity or higher performance storage device available, such as a RAID storage system.



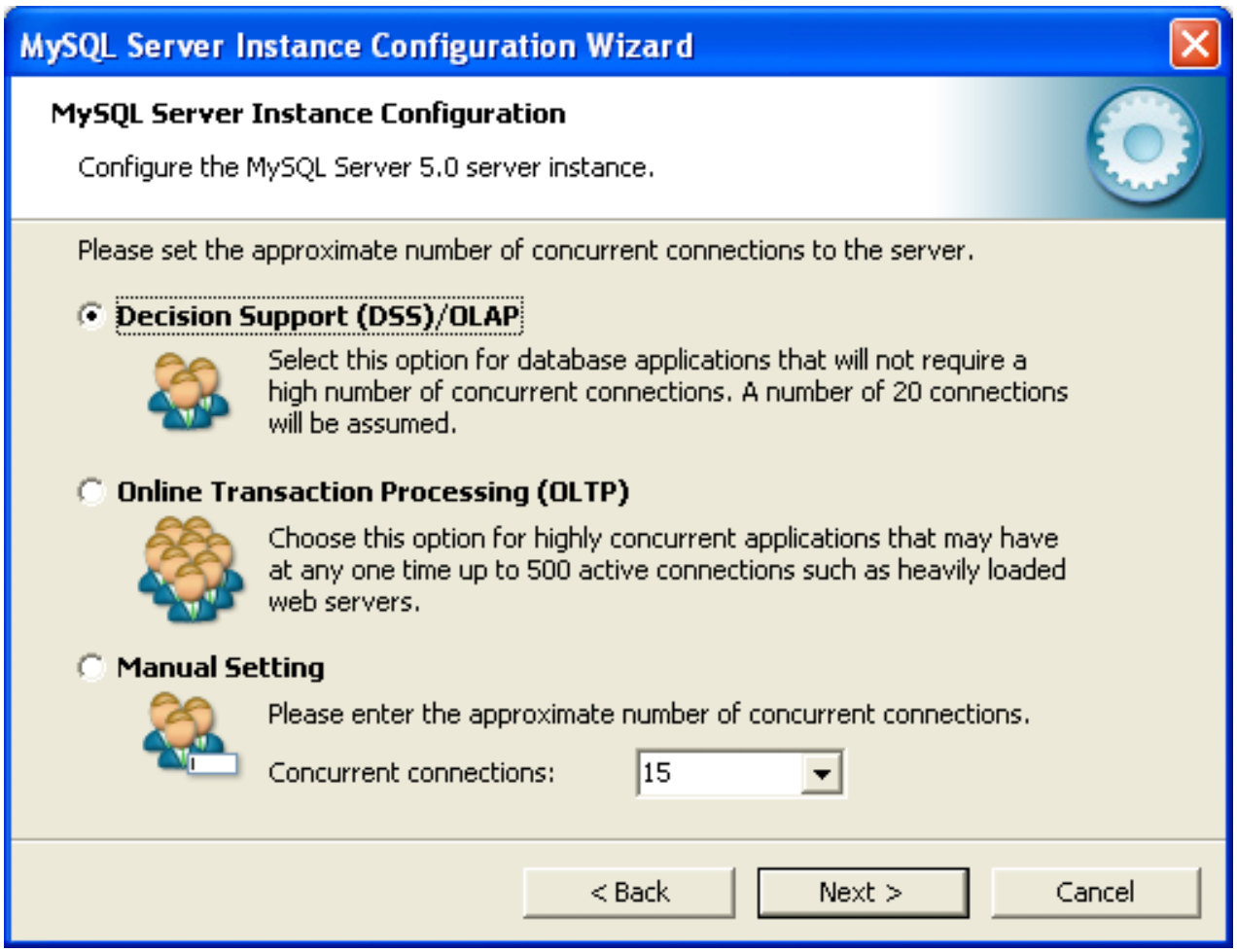
This documentation is for an older version. If you're

To change the default location for the `InnoDB` tablespace files, choose a new drive from the drop-down list of drive letters and choose a new path from the drop-down list of paths. To create a custom path, click the **...** button.

If you are modifying the configuration of an existing server, you must click the **Modify** button before you change the path. In this situation you must move the existing tablespace files to the new location manually before starting the server.

2.10.3.7 The Concurrent Connections Dialog

To prevent the server from running out of resources, it is important to limit the number of concurrent connections to the MySQL server that can be established. The **Concurrent Connections** dialog enables you to choose the expected usage of your server, and sets the limit for concurrent connections accordingly. It is also possible to set the concurrent connection limit manually.

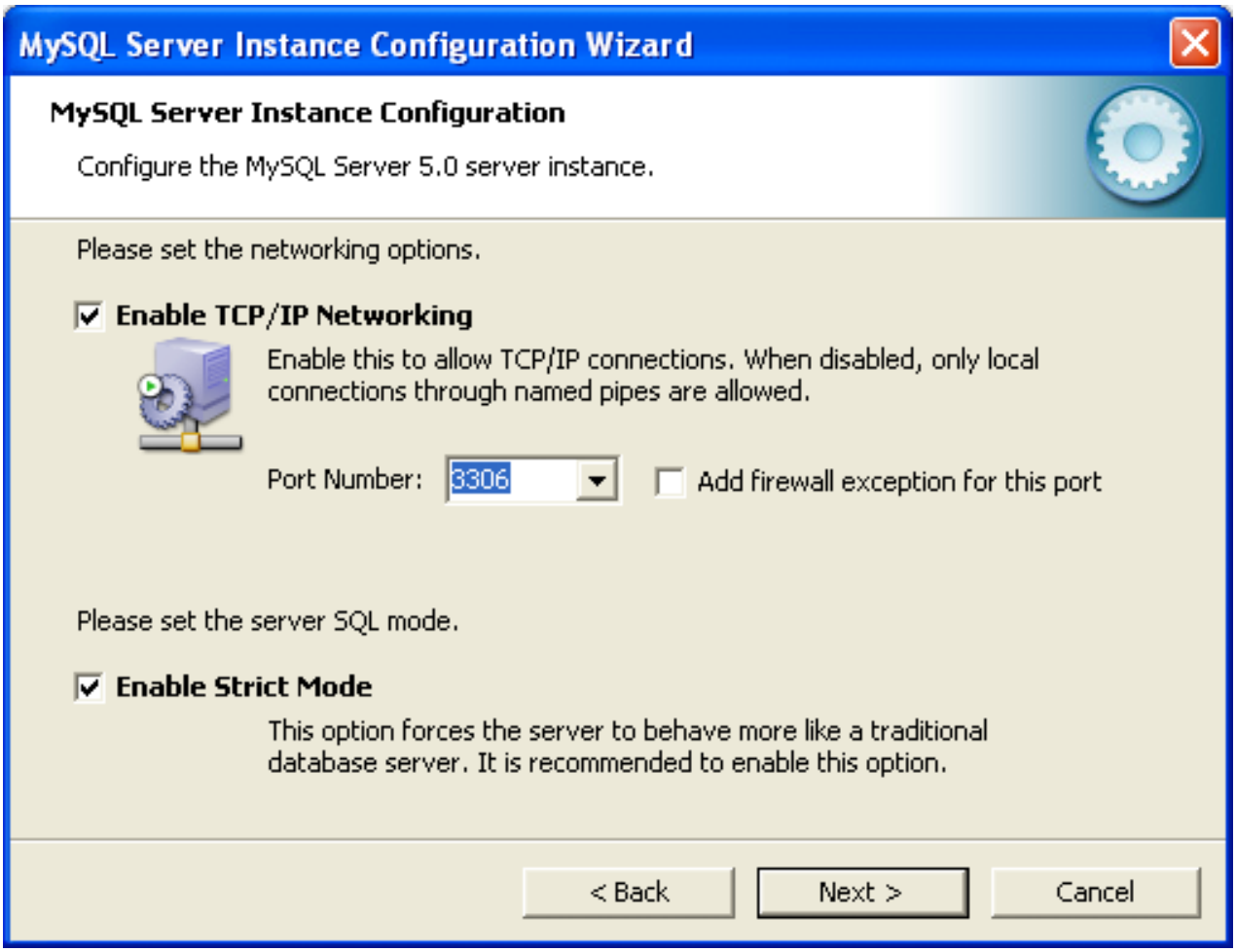


- **Decision Support (DSS)/OLAP**: Choose this option if your server does not require a large number of concurrent connections. The maximum number of connections is set at 100, with an average of 20 concurrent connections assumed.
- **Online Transaction Processing (OLTP)**: Choose this option if your server requires a large number of concurrent connections. The maximum number of connections is set at 500.
- **Manual Setting**: Choose this option to set the maximum number of concurrent connections to the server manually. Choose the number of concurrent connections from the drop-down box provided, or enter the maximum number of connections into the drop-down box if the number you desire is not listed.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.10.3.8 The Networking and Strict Mode Options Dialog

Use the **Networking Options** dialog to enable or disable TCP/IP networking and to configure the port number that is used to connect to the MySQL server.



TCP/IP networking is enabled by default. To disable TCP/IP networking, uncheck the box next to the **Enable TCP/IP Networking** option.

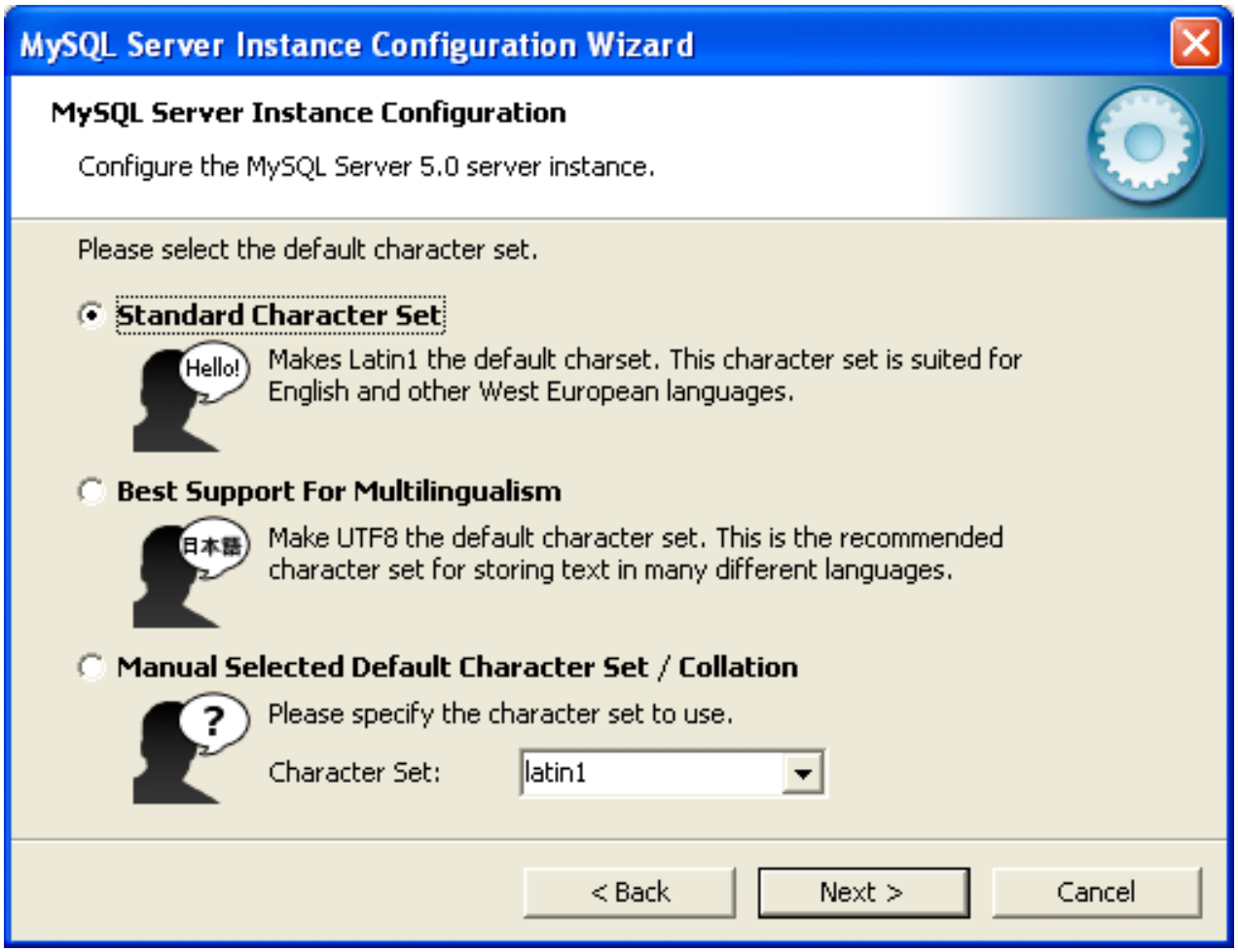
Port 3306 is used by default. To change the port used to access MySQL, choose a new port number from the drop-down box or type a new port number directly into the drop-down box. If the port number you choose is in use, you are prompted to confirm your choice of port number.

Set the **Server SQL Mode** to either enable or disable strict mode. Enabling strict mode (default) makes MySQL behave more like other database management systems. *If you run applications that rely on MySQL's old "forgiving" behavior, make sure to either adapt those applications or to disable strict mode.* For more information about strict mode, see Section 5.1.7, "Server SQL Modes".

2.10.3.9 The Character Set Dialog

The MySQL server supports multiple character sets and it is possible to set a default server character set that is applied to all tables, columns, and databases unless overridden. Use the **Character Set** dialog to change the default character set of the MySQL server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



- **Standard Character Set**: Choose this option if you want to use `latin1` as the default server character set. `latin1` is used for English and many Western European languages.
- **Best Support For Multilingualism**: Choose this option if you want to use `utf8` as the default server character set. This is a Unicode character set that can store characters from many different languages.
- **Manual Selected Default Character Set / Collation**: Choose this option if you want to pick the server's default character set manually. Choose the desired character set from the provided drop-down list.

2.10.3.10 The Service Options Dialog

On Windows platforms, the MySQL server can be installed as a Windows service. When installed this way, the MySQL server can be started automatically during system startup, and even restarted automatically by Windows in the event of a service failure.

The MySQL Server Instance Configuration Wizard installs the MySQL server as a service by default, using the service name `MySQL`. If you do not wish to install the service, uncheck the box next to the **Install As Windows Service** option. You can change the service name by picking a new service name from the drop-down box provided or by entering a new service name into the drop-down box.

Note

Service names can include any legal character except forward (`/`) or backward (`\`) slashes, and must be less than 256 characters long.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Warning

If you are installing multiple versions of MySQL onto the same machine, you *must* choose a different service name for each version that you install. If you do not choose a different service for each installed version then the service manager information will be inconsistent and this will cause problems when you try to uninstall a previous version.

If you have already installed multiple versions using the same service name, you must manually edit the contents of the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services` parameters within the Windows registry to update the association of the service name with the correct server version.

Typically, when installing multiple versions you create a service name based on the version information. For example, you might install MySQL 5.x as `mysql5`, or specific versions such as MySQL 5.0.56 as `mysql50056`.

To install the MySQL server as a service but not have it started automatically at startup, uncheck the box next to the **Launch the MySQL Server Automatically** option.

2.10.3.11 The Security Options Dialog

It is strongly recommended that you set a `root` password for your MySQL server, and the MySQL Server Instance Configuration Wizard requires by default that you do so. If you do not wish to set a `root` password, uncheck the box next to the **Modify Security Settings** option.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To set the `root` password, enter the desired password into both the **New root password** and **Confirm** boxes. If you are reconfiguring an existing server, you need to enter the existing `root` password into the **Current root password** box.

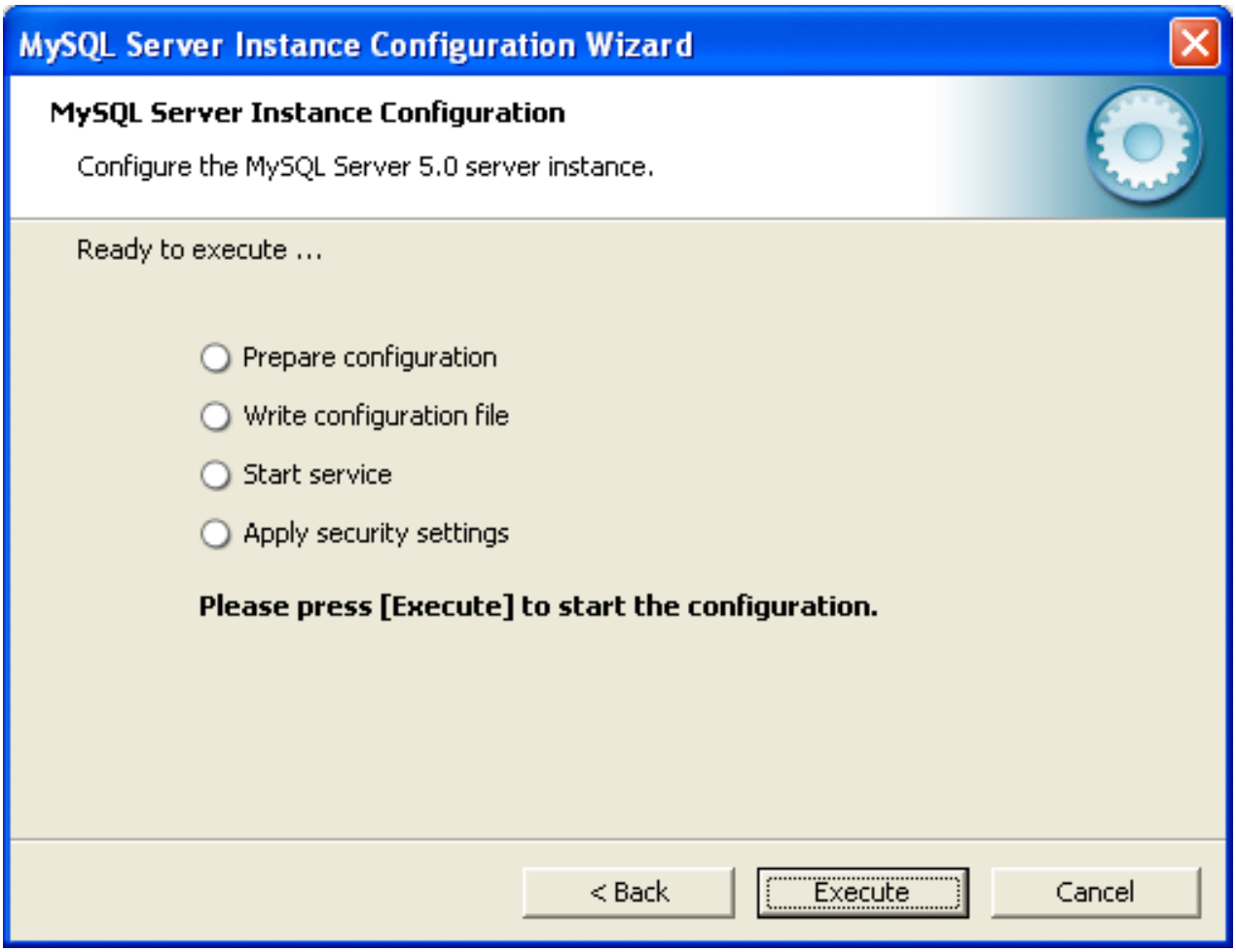
To permit `root` logins from across the network, check the box next to the **Enable root access from remote machines** option. This decreases the security of your `root` account.

To create an anonymous user account, check the box next to the **Create An Anonymous Account** option. Creating an anonymous account can decrease server security and cause login and permission difficulties. For this reason, it is not recommended.

2.10.3.12 The Confirmation Dialog

The final dialog in the MySQL Server Instance Configuration Wizard is the **Confirmation Dialog**. To start the configuration process, click the **Execute** button. To return to a previous dialog, click the **Back** button. To exit the MySQL Server Instance Configuration Wizard without configuring the server, click the **Cancel** button.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



After you click the **Execute** button, the MySQL Server Instance Configuration Wizard performs a series of tasks and displays the progress onscreen as the tasks are performed.

The MySQL Server Instance Configuration Wizard first determines configuration file options based on your choices using a template prepared by MySQL developers and engineers. This template is named `my-template.ini` and is located in your server installation directory.

The MySQL Configuration Wizard then writes these options to the corresponding configuration file.

If you chose to create a service for the MySQL server, the MySQL Server Instance Configuration Wizard creates and starts the service. If you are reconfiguring an existing service, the MySQL Server Instance Configuration Wizard restarts the service to apply your configuration changes.

If you chose to set a `root` password, the MySQL Configuration Wizard connects to the server, sets your new `root` password, and applies any other security settings you may have selected.

After the MySQL Server Instance Configuration Wizard has completed its tasks, it displays a summary. Click the **Finish** button to exit the MySQL Server Configuration Wizard.

2.10.4 Installing MySQL on Microsoft Windows Using a noinstall Zip Archive

Users who are installing from the Noinstall package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a Zip archive is as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.10.4.1 Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to Section 2.10.7, “Upgrading MySQL on Windows”, before beginning the upgrade process.
2. Make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server is installed in `C:\mysql`. The MySQL Installation Wizard installs MySQL under `C:\Program Files\MySQL`. If you do not install MySQL at `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See Section 2.10.4.2, “Creating an Option File”.
4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.10.4.2 Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (`C:\Program Files\MySQL\MySQL Server 5.0` and `C:\Program Files\MySQL\MySQL Server 5.0\data`).
- You need to tune the server settings.

When the MySQL server starts on Windows, it looks for option files in several locations, such as the Windows directory, `C:\`, and the MySQL installation directory (for the full list of locations, see Section 4.2.6, “Using Option Files”). The Windows directory typically is named something like `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
shell> echo %WINDIR%
```

MySQL looks for options in each location first in the `my.ini` file, and then in the `my.cnf` file. However, to avoid confusion, it is best if you use only one file. If your PC uses a boot loader where `C:` is not the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.

You can also make use of the example option files included with your MySQL distribution; see Section 5.1.2, “Server Configuration Defaults”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is in `E:\mydata\data`, you can create an option file containing a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Note that Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

The rules for use of backslash in option file values are given in Section 4.2.6, “Using Option Files”.

On Windows, the MySQL installer places the data directory directly under the directory where you install MySQL. If you would like to use a data directory in a different location, you should copy the entire contents of the `data` directory to the new location. For example, if MySQL is installed in `C:\Program Files\MySQL\MySQL Server 5.0`, the data directory is by default in `C:\Program Files\MySQL\MySQL Server 5.0\data`. If you want to use `E:\mydata` as the data directory instead, you must do two things:

1. Move the entire `data` directory and all of its contents from `C:\Program Files\MySQL\MySQL Server 5.0\data` to `E:\mydata`.
2. Use a `--datadir` option to specify the new data directory location each time you start the server.

2.10.4.3 Selecting a MySQL Server Type

The following table shows the available servers for Windows in MySQL 5.0.

Binary	Description
`mysqld-nt`	Optimized binary with named-pipe support
`mysqld`	Optimized binary without named-pipe support
`mysqld-debug`	Like `mysqld-nt`, but compiled with full debugging and automatic memory allocation checking

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

Each of the servers in a distribution support the same set of storage engines. The `SHOW ENGINES` statement displays which engines a given server supports.

All Windows MySQL 5.0 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows support named pipes as indicated in the following list. However, the default is to use TCP/IP regardless of platform. (Named pipes are slower than TCP/IP in many Windows configurations.)

Use of named pipes is subject to these conditions:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Named pipes are enabled only if you start the server with the `--enable-named-pipe` option. It is necessary to use this option explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used.
- Named-pipe connections are permitted only by the `mysqld-nt` and `mysqld-debug` servers.

Note

Most of the examples in this manual use `mysqld` as the server name. If you choose to use a different server, such as `mysqld-nt`, make the appropriate substitutions in the commands that are shown in the examples.

2.10.4.4 Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the `Noinstall` version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

The examples in these sections assume that MySQL is installed under the default location of `C:\Program Files\MySQL\MySQL Server 5.0`. Adjust the path names shown in the examples if you have MySQL installed in a different location.

Clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections.

MySQL for Windows also supports shared-memory connections if the server is started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=MEMORY` option.

For information about which server binary to run, see Section 2.10.4.3, “Selecting a MySQL Server Type”.

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --console
```

For a server that includes `InnoDB` support, you should see the messages similar to those following as it starts (the path names and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '5.0.96'  socket: ''  port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` option, the server writes diagnostic output to the error log in the data directory (`C:\Program Files\MySQL\MySQL Server 5.0\data` by default). The error log is the file with the `.err` extension.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.18.4, “Securing the Initial MySQL Accounts”.

2.10.4.5 Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the `mysqld` server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"
```

The path to `mysqld` may vary depending on the install location of MySQL on your system.

You can stop the MySQL server by executing this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If `mysqld` doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the `C:\Program Files\MySQL\MySQL Server 5.0\data` directory. It is the file with a suffix of `.err`. You can also try to start the server as `mysqld --console`; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start `mysqld` with the `--standalone` and `--debug` options. In this case, `mysqld` writes a log file `C:\mysqld.trace` that should contain the reason why `mysqld` doesn't start. See Section 21.3.3, “The DBUG Package”.

Use `mysqld --verbose --help` to display all the options that `mysqld` supports.

This documentation is for an older version. If you're

2.10.4.6 Customizing the PATH for MySQL Tools

To make it easier to invoke MySQL programs, you can add the path name of the MySQL `bin` directory to your Windows system `PATH` environment variable:

- On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
- Next select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
- Under **System Variables**, select **Path**, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL `bin` directory (for example, `C:\Program Files\MySQL\MySQL Server 5.0\bin`)

Note

There must be a semicolon separating this path from any values present in this field.

Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the `mysql` client, and all MySQL command-line utilities such as `mysqladmin` and `mysqldump`.

You should not add the MySQL `bin` directory to your Windows `PATH` if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system `PATH` by hand; accidental deletion or modification of any portion of the existing `PATH` value can leave you with a malfunctioning or even unusable system.

2.10.4.7 Starting MySQL as a Windows Service

On Windows, the recommended way to run MySQL is to install it as a Windows service, whereby MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using `NET` commands, or with the graphical `Services` utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

The `Services` utility (the Windows `Service Control Manager`) can be found in the Windows Control Panel (under **Administrative Tools** on Windows 2000, XP, Vista, and Server 2003). To avoid conflicts, it is advisable to close the `Services` utility while performing server installation or removal operations from the command line.

Installing the service

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin"
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
          -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL `bin` directory to your Windows system `PATH` environment variable:

- On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
- Next select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
- Under **System Variables**, select **Path**, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL `bin` directory (for example, `C:\Program Files\MySQL\MySQL Server 5.0\bin`), Note that there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the `mysql` client, and all MySQL command-line utilities such as `mysqladmin` and `mysqldump`.

  You should not add the MySQL `bin` directory to your Windows `PATH` if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system `PATH` by hand; accidental deletion or modification of any portion of the existing `PATH` value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used when installing the service:

- You can specify a service name immediately following the `--install` option. The default service name is `MySQL`.
- If a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=`*`file_name`* to specify the name of an option file from which the server should read options when it starts.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The use of a single option other than `--defaults-file` is possible but discouraged. `--defaults-file` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file. Also, in MySQL 5.0, use of an option different from `--defaults-file` is not supported until 5.0.3.

- As of MySQL 5.0.1, you can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the `[mysqld]` group and the group that has the same name as the service in the standard option files. This enables you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the service name for use by the server installed with that service name.
- If the service-installation command specifies a `--defaults-file` option after the service name, the server reads options the same way as described in the previous item, except that it reads options only from the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"
         --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

Note

On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. Otherwise, `mysqld.exe` will attempt to start the MySQL server.

You can also specify options as Start parameters in the Windows `Services` utility before you start the MySQL service.

Starting the service

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the `Services` utility, or by using a `NET START MySQL` command. The `NET` command is not case sensitive.

When run as a service, `mysqld` has no access to a console window, so no messages can be seen there. If `mysqld` does not start, check the error log to see whether the server wrote any messages there to indicate

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the cause of the problem. The error log is located in the MySQL data directory (for example, `C:\Program Files\MySQL\MySQL Server 5.0\data`). It is the file with a suffix of `.err`.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the `Services` utility, the `NET STOP MySQL` command, or the `mysqladmin shutdown` command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the `--install-manual` option rather than the `--install` option:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --install-manual
```

Removing the service

To remove a server that is installed as a service, first stop it if it is running by executing `NET STOP MySQL`. Then use the `--remove` option to remove it:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --remove
```

If `mysqld` is not running as a service, you can start it from the command line. For instructions, see Section 2.10.4.5, “Starting MySQL from the Windows Command Line”.

Please see Section 2.10.5, “Troubleshooting a MySQL Installation Under Windows”, if you encounter difficulties during installation.

For more information about stopping or removing a MySQL Windows service, see Section 5.5.2.2, “Starting Multiple MySQL Instances as Windows Services”.

2.10.4.8 Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow"
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow" -u root mysql
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" version status proc
shell> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql" test
```

If `mysqld` is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start `mysqld` with the `--skip-name-resolve` option and use only `localhost` and IP addresses in the `Host` column of the MySQL grant tables. (Be sure that an account exists that specifies an IP address or you may not be able to connect.)

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Note that if you have set a password for the `root` account, deleted the anonymous account, or created a new user account, then to connect to the MySQL server you must use the appropriate `-u` and `-p` options with the commands shown previously. See Section 4.2.2, “Connecting to the MySQL Server”.

For more information about `mysqlshow`, see Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.10.5 Troubleshooting a MySQL Installation Under Windows

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. The purpose of this section is to help you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 5.0\data`. See Section 5.4.1, “The Error Log”.

Another source of information regarding possible errors is the console messages displayed when the MySQL service is starting. Use the `NET START MySQL` command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See Section 2.10.4.7, “Starting MySQL as a Windows Service”.

The following examples show other common error messages you may encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, you may see these messages:

```
System error 1067 has occurred.
Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
```

  These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 5.0` and `C:\Program Files\MySQL\MySQL Server 5.0\data`, respectively).

  This situation may occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, there may be old and new configuration files that conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

  If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 5.0`, you need to ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. The `my.ini` file needs to be located in your Windows directory, typically `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable by issuing the following command from the command prompt:

```
shell> echo %WINDIR%
```

  An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

  Note that Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.0
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

The rules for use of backslash in option file values are given in Section 4.2.6, “Using Option Files”.

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See Section 2.10.4.2, “Creating an Option File”.

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Configuration Wizard, you may see this error:

```
Error: Cannot create Windows service for MySql. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This enables the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command-line:

```
shell> sc delete mysql
[SC] DeleteService SUCCESS
```

If the `sc` utility is not available for your version of Windows, download the `delsrv` utility from http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp and use the `delsrv mysql` syntax.

2.10.6 Windows Postinstallation Procedures

On Windows, you need not create the data directory and the grant tables. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory.

Regarding passwords, if you installed MySQL using the Windows Installation Wizard, you may have already assigned passwords to the accounts. (See Section 2.10.2.1, “Using the MySQL Installation Wizard”.) Otherwise, use the password-assignment procedure given in Section 2.18.4, “Securing the Initial MySQL Accounts”.

Before assigning passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see Section 2.10.4.4, “Starting the Server for the First Time”). You can also set up a MySQL service that runs automatically when Windows starts (see Section 2.10.4.7, “Starting MySQL as a Windows Service”).

These instructions assume that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

If you installed MySQL using the Windows installation Wizard (see Section 2.10.2.1, “Using the MySQL Installation Wizard”), the default installation directory is `C:\Program Files\MySQL\MySQL Server 5.0`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 5.0"
```

A common installation location for installation from a Zip package is `C:\mysql`:

```
C:\> cd C:\mysql
```

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your command interpreter to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Section 2.10.4.6, “Customizing the PATH for MySQL Tools”.

With the server running, issue the following commands to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
C:\> bin\mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

The list of installed databases may vary, but will always include the minimum of `mysql` and `information_schema`.

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you installed using the MSI packages and used the MySQL Server Instance Config Wizard, then the `root` user will have been created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options. (You will also need to use the `-u root` and `-p` options if you have already secured the initial MySQL accounts.) With `-p`, you will be prompted for the `root` password. For example:

```
C:\> bin\mysqlshow -u root -p
Enter password: (enter root password here)
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
C:\> bin\mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| host                      |
| proc                      |
| procs_priv                |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
C:\> bin\mysql -e "SELECT User, Host FROM mysql.user" mysql
+------+-----------+
| User | Host      |
+------+-----------+
| root | localhost |
+------+-----------+
```

For more information about `mysql` and `mysqlshow`, see Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”, and Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

If you are running a version of Windows that supports services, you can set up the MySQL server to run automatically when Windows starts. See Section 2.10.4.7, “Starting MySQL as a Windows Service”.

2.10.7 Upgrading MySQL on Windows

To upgrade MySQL on Windows, follow these steps:

1. Review Section 2.19.1, “Upgrading MySQL”, for additional information on upgrading MySQL that is not specific to Windows.
2. You should always back up your current MySQL installation before performing an upgrade. See Section 7.2, “Database Backup Methods”.
3. Download the latest Windows distribution of MySQL from http://dev.mysql.com/downloads/.
4. Before upgrading MySQL, you must stop the server. If the server is installed as a service, stop the service with the following command from the command prompt:

   ```
   shell> NET STOP MySQL
   ```

   If you are not running the MySQL server as a service, use `mysqladmin` to stop it. For example, before upgrading from MySQL 4.1 to 5.0, use `mysqladmin` from MySQL 4.1 as follows:

   ```
   shell> "C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqladmin" -u root shutdown
   ```

   Note

   If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and enter the password when prompted.

5. Before upgrading to MySQL 5.0 from a version previous to 4.1.5, or from a version of MySQL installed from a Zip archive to a version of MySQL installed with the MySQL Installation Wizard, you must first manually remove the previous installation and MySQL service (if the server is installed as a service).

   To remove the MySQL service, use the following command:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> C:\mysql\bin\mysqld --remove
```

If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

6. If you are using the MySQL Installation Wizard, start the wizard as described in Section 2.10.2.1, “Using the MySQL Installation Wizard”.
7. If you are installing MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql5`. Overwriting the existing installation is recommended.
8. If you were running MySQL as a Windows service and you had to remove the service earlier in this procedure, reinstall the service. (See Section 2.10.4.7, “Starting MySQL as a Windows Service”.)
9. Restart the server. For example, use `NET START MySQL` if you run MySQL as a service, or invoke `mysqld` directly otherwise.
10. As Administrator, run `mysql_upgrade` to check your tables, attempt to repair them if necessary, and update your grant tables if they have changed so that you can take advantage of any new capabilities. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.
11. If you encounter errors, see Section 2.10.5, “Troubleshooting a MySQL Installation Under Windows”.

2.10.8 Installing MySQL from Source on Windows

These instructions describe how to build binaries from source for MySQL 5.0 on Windows. Instructions are provided for building binaries from a standard source distribution or from the Bazaar tree that contains the latest development source.

Note

The instructions here are strictly for users who want to test MySQL on Microsoft Windows from the latest source distribution or from the Bazaar tree. For production use, we do not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by Oracle Corporation. Instructions for installing binary distributions are available in Section 2.10, “Installing MySQL on Microsoft Windows”.

To build MySQL on Windows from source, you must satisfy the following system, compiler, and resource requirements:

- Windows 2000, Windows XP, or newer version.

  Windows Vista is supported when using Visual Studio 2005 provided you have installed the following updates:

  - Microsoft Visual Studio 2005 Professional Edition - ENU Service Pack 1 (KB926601)
  - Security Update for Microsoft Visual Studio 2005 Professional Edition - ENU (KB937061)
  - Update for Microsoft Visual Studio 2005 Professional Edition - ENU (KB932232)

- To build from the standard source distribution, you will need `CMake`, which can be downloaded from http://www.cmake.org. After installing, modify your `PATH` environment variable to include the directory where `cmake` is located.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.
- If you are using Visual C++ 2005 Express Edition, you must also install an appropriate Platform SDK. More information and links to downloads for various Windows platforms is available from http://www.microsoft.com/downloads/details.aspx?familyid=0baf2b35-c656-4969-ace8-e4c0c0716adb.
- If you are compiling from a Bazaar tree or making changes to the parser, you need `bison` for Windows, which can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled "Complete package, excluding sources". After installing the package, modify your `PATH` environment variable to include the directory where `bison` is located.
- Cygwin might be necessary if you want to run the test script or package the compiled binaries and support files into a Zip archive. (Cygwin is needed only to test or package the distribution, not to build it.) Cygwin is available from http://cygwin.com.
- 3GB to 5GB of disk space.

There are three solutions available for building from the source code on Windows:

- Build from the standard MySQL source distribution. For this you will need `CMake` and Visual C++ Express Edition or Visual Studio. Using this method you can select the storage engines that are included in your build. To use this method, see Section 2.10.8.1, "Building MySQL from the Standard Source Distribution".
- Build from the MySQL Windows source distribution. The Windows source distribution includes ready-made Visual Studio solution files that enable support for all storage engines (except `NDB`). To build using using method you only need Visual C++ Express Edition or Visual Studio. To use this method, see Section 2.10.8.2, "Building MySQL from a Windows Source Distribution".
- Build directly from the Bazaar source repository. For this you will need `CMake`, Visual C++ Express Edition or Visual Studio, and `bison`. For this method you need to create the distribution on a Unix system and then copy the generated files to your Windows build environment. To use this method, see Section 2.10.8.5, "Creating a Windows Source Package from the Bazaar Repository".

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the `win32` mailing list. See Section 1.6.1, "MySQL Mailing Lists".

2.10.8.1 Building MySQL from the Standard Source Distribution

You can build MySQL on Windows by using a combination of `cmake` and Microsoft Visual Studio .NET 2003 (7.1), Microsoft Visual Studio 2005 (8.0) or Microsoft Visual C++ 2005 Express Edition. You must have the appropriate Microsoft Platform SDK installed.

Note

To compile from the source code using `CMake` you must use the standard source distribution (for example, `mysql-5.0.96.tar.gz`). You build from the same distribution as used to build MySQL on Unix, Linux and other platforms. Do *not* use the Windows Source distributions as they do not contain the necessary configuration script and other files.

Follow this procedure to build MySQL:

1. If you are installing from a packaged source distribution, create a work directory (for example, `C:\workdir`), and unpack the source distribution there using `WinZip` or another Windows tool that can read `.zip` files. This directory is the work directory in the following instructions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. If you are installing from a Bazaar tree, the root directory of that tree is the work directory in the following instructions.

3. Using a command shell, navigate to the work directory and run the following command:

```
C:\workdir>win\configure.js options
```

If you have associated the `.js` file extension with an application such as a text editor, then you may need to use the following command to force `configure.js` to be executed as a script:

```
C:\workdir>cscript win\configure.js options
```

These options are available for `configure.js`:

- `WITH_INNOBASE_STORAGE_ENGINE`: Enable the `InnoDB` storage engine.
- `WITH_PARTITION_STORAGE_ENGINE`: Enable user-defined partitioning.
- `WITH_ARCHIVE_STORAGE_ENGINE`: Enable the `ARCHIVE` storage engine.
- `WITH_BLACKHOLE_STORAGE_ENGINE`: Enable the `BLACKHOLE` storage engine.
- `WITH_EXAMPLE_STORAGE_ENGINE`: Enable the `EXAMPLE` storage engine.
- `WITH_FEDERATED_STORAGE_ENGINE`: Enable the `FEDERATED` storage engine.
- `MYSQL_SERVER_SUFFIX=suffix`: Server suffix, default none.
- `COMPILATION_COMMENT=comment`: Server comment, default "Source distribution".
- `MYSQL_TCP_PORT=port`: Server port, default 3306.
- `DISABLE_GRANT_OPTIONS`: Disables the the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld`. This option is available as of MySQL 5.0.36.

For example (type the command on one line):

```
C:\workdir>win\configure.js WITH_INNOBASE_STORAGE_ENGINE »
            WITH_PARTITION_STORAGE_ENGINE MYSQL_SERVER_SUFFIX=-pro
```

4. From the work directory, execute the `win\build-vs8.bat` or `win\build-vs71.bat` file, depending on the version of Visual Studio you have installed. The script invokes `CMake`, which generates the `mysql.sln` solution file you will need to build MySQL using Visual Studio..

   You can also use `win\build-vs8_x64.bat` to build the 64-bit version of MySQL. However, you cannot build the 64-bit version with Visual Studio Express Edition. You must use Visual Studio 2005 (8.0) or higher.

5. From the work directory, open the generated `mysql.sln` file with Visual Studio and select the proper configuration using the **Configuration** menu. The menu provides **Debug**, **Release**, **RelwithDebInfo**, **MinRelInfo** options. Then select **Solution** > **Build** to build the solution.

   The build process will take some time. Please be patient.

   Remember the configuration that you use in this step. It is important later when you run the test script because that script needs to know which configuration you used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6. You should test you build before installation. See Section 2.10.8.4, “Testing a Windows Source Build”.

7. To install, use the instructions in Section 2.10.8.3, “Installing MySQL from a Source Build on Windows”.

2.10.8.2 Building MySQL from a Windows Source Distribution

The Windows source distribution includes the necessary solution file and the `vcproj` files required to build each component. Using this method you are not able to select the storage engines that are included in your build.

Note

VC++ workspace files for MySQL 4.1 and above are compatible with Microsoft Visual Studio 7.1 and tested by us before each release.

Follow this procedure to build MySQL:

1. Create a work directory (for example, `C:\workdir`).

2. Unpack the source distribution in the aforementioned directory using `WinZip` or another Windows tool that can read `.zip` files.

3. Start Visual Studio .Net 2003 (7.1).

4. From the **File** menu, select **Open Solution...**.

5. Open the `mysql.sln` solution you find in the work directory.

6. From the **Build** menu, select **Configuration Manager...**.

7. In the **Active Solution Configuration** pop-up menu, select the configuration to use. You likely want to use one of **nt** (normal server), **Max nt** (more engines and features), or **Debug** configuration.

8. From the **Build** menu, select **Build Solution**.

9. Debug versions of the programs and libraries are placed in the `client_debug` and `lib_debug` directories. Release versions of the programs and libraries are placed in the `client_release` and `lib_release` directories.

10. You should test you build before installation. See Section 2.10.8.4, “Testing a Windows Source Build”.

11. To install, use the instructions in Section 2.10.8.3, “Installing MySQL from a Source Build on Windows”.

2.10.8.3 Installing MySQL from a Source Build on Windows

When you are satisfied that the program you have built is working correctly, stop the server. Now you can install the distribution. There are two ways to do this, either by using the supplied installation script or by copying the files individually by hand.

To use the script method you must have Cygwin installed as the script is a Shell script. To execute the installation process, run the `make_win_bin_dist` script in the `scripts` directory of the MySQL source distribution (see Section 4.4.2, “`make_win_bin_dist` — Package MySQL Distribution as Zip Archive”). This is a shell script, so you must have Cygwin installed if you want to use it. It creates a Zip archive of the built executables and support files that you can unpack to your desired installation location.

It is also possible to install MySQL by copying directories and files manually:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mkdir C:\mysql
shell> mkdir C:\mysql\bin
shell> mkdir C:\mysql\data
shell> mkdir C:\mysql\share
shell> mkdir C:\mysql\scripts
```

If you want to compile other clients and link them to MySQL, you should also create several additional directories:

```
shell> mkdir C:\mysql\include
shell> mkdir C:\mysql\lib
shell> mkdir C:\mysql\lib\debug
shell> mkdir C:\mysql\lib\opt
```

If you want to benchmark MySQL, create this directory:

```
shell> mkdir C:\mysql\sql-bench
```

Benchmarking requires Perl support. See Section 2.22, “Perl Installation Notes”.

2. From the work directory, copy into the `C:\mysql` directory the following directories:

```
shell> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
C:\workdir> xcopy scripts\*.* C:\mysql\scripts /E
C:\workdir> xcopy share\*.* C:\mysql\share /E
```

If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

If you want to benchmark MySQL, you should also do this:

```
C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
```

After installation, set up and start the server in the same way as for binary Windows distributions. See Section 2.10, “Installing MySQL on Microsoft Windows”.

2.10.8.4 Testing a Windows Source Build

You should test the server that you have built from source before using the distribution.

To test the server you need to run the built `mysqld`. By default, using the source build examples, the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data`. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their path names. You can either do this on the command line with the `--basedir` and `--datadir` options, or by placing appropriate options in an option file. (See Section 4.2.6,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

"Using Option Files".) If you have an existing data directory elsewhere that you want to use, you can specify its path name instead.

When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the `mysql` interactive command-line utility.

You can also run the standard test script, `mysql-test-run.pl`. This script is written in Perl, so you'll need either Cygwin or ActiveState Perl to run it. You may also need to install the modules required by the script. To run the test script, change location into the `mysql-test` directory under the work directory, set the `MTR_VS_CONFIG` environment variable to the configuration you selected earlier (or use the `--vs-config` option), and invoke `mysql-test-run.pl`. For example (using Cygwin and the `bash` shell):

```
shell> cd mysql-test
shell> export MTR_VS_CONFIG=debug
shell> ./mysql-test-run.pl --force --timer
shell> ./mysql-test-run.pl --force --timer --ps-protocol
```

2.10.8.5 Creating a Windows Source Package from the Bazaar Repository

To create a Windows source package from the current Bazaar source tree, use the instructions here. This procedure must be performed on a system running a Unix or Unix-like operating system because some of the configuration and build steps require tools that work only on Unix. For example, the following procedure is known to work well on Linux.

1. Copy the Bazaar source tree for MySQL 5.0. For instructions on how to do this, see Section 2.17.2, "Installing MySQL Using a Development Source Tree".

2. Configure and build the distribution so that you have a server binary to work with. One way to do this is to run the following command in the top-level directory of your source tree:

   ```
   shell> ./BUILD/compile-pentium-max
   ```

3. After making sure that the build process completed successfully, run the following utility script from top-level directory of your source tree:

   ```
   shell> ./scripts/make_win_src_distribution
   ```

   This script creates a Windows source package to be used on your Windows system.
   You can supply different options to the script based on your needs. See Section 4.4.3, "`make_win_src_distribution` — Create Source Distribution for Windows", for a list of permissible options.

   By default, `make_win_src_distribution` creates a Zip-format archive with the name `mysql-VERSION-win-src.zip`, where *`VERSION`* represents the version of your MySQL source tree.

4. Copy or upload the Windows source package that you have just created to your Windows machine. To compile it, use the instructions in Section 2.10.8.2, "Building MySQL from a Windows Source Distribution".

2.11 Installing MySQL on OS X

You can install MySQL on OS X 10.3.x ("Panther") or newer using a OS X binary package in DMG format instead of the binary tarball distribution. Please note that older versions of OS X (for example, 10.1.x or 10.2.x) are **not** supported by this package.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

When installing from the package version, you should also install the MySQL Preference Pane, which will enable you to control the startup and execution of your MySQL server from System Preferences.

To obtain MySQL, see Section 2.5, “How to Get MySQL”.

Note

Before proceeding with the installation, be sure to shut down all running MySQL server instances by using either the MySQL Manager Application (on OS X Server) or `mysqladmin shutdown` on the command line.

To actually install the MySQL DMG file, double-click the package icon. This launches the OS X Package Installer, which guides you through the installation of MySQL.

Due to a bug in the OS X package installer, you may see this error message in the destination disk selection dialog:

```
You cannot install this software on this disk. (null)
```

If this error occurs, simply click the `Go Back` button once to return to the previous screen. Then click `Continue` to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

The OS X MySQL DMG package installs itself into `/usr/local/mysql-`*`VERSION`* and also installs a symbolic link, `/usr/local/mysql`, that points to the new location. If a directory named `/usr/local/mysql` exists, it is renamed to `/usr/local/mysql.bak` first. Additionally, the installer creates the grant tables in the `mysql` database by executing `mysql_install_db`.

The installation layout is similar to that of a `tar` file binary distribution; all MySQL binaries are located in the directory `/usr/local/mysql/bin`. The MySQL socket file is created as `/tmp/mysql.sock` by default. See Section 2.7, “Installation Layouts”.

MySQL installation requires a OS X user account named `mysql`. A user account with this name should exist by default on OS X 10.2 and up.

If you are running OS X Server, a version of MySQL should already be installed. The following table shows the versions of MySQL that ship with OS X Server versions.

OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a

This manual section covers the installation of the official MySQL OS X DMG only. Make sure to read Apple's help information about installing MySQL: Run the “Help View” application, select “OS X Server” help, do a search for “MySQL,” and read the item entitled “Installing MySQL.”

For preinstalled versions of MySQL on OS X Server, note especially that you should start `mysqld` with `safe_mysqld` instead of `mysqld_safe` if MySQL is older than version 4.0.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you previously used Marc Liyanage's MySQL packages for OS X from http://www.entropy.ch, you can simply follow the update instructions for packages using the binary installation layout as given on his pages.

If you are upgrading from Marc's 3.23.x versions or from the OS X Server version of MySQL to the official MySQL DMG, you also need to convert the existing MySQL privilege tables to the current format, because some new security privileges have been added. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

If you want MySQL to start automatically during system startup, you also need to install the MySQL Startup Item. It is part of the OS X installation disk images as a separate installation package. Simply double-click the **MySQLStartupItem.pkg** icon and follow the instructions to install it. The Startup Item need be installed only once. There is no need to install it each time you upgrade the MySQL package later.

The Startup Item for MySQL is installed into `/Library/StartupItems/MySQLCOM`. (Before MySQL 4.1.2, the location was `/Library/StartupItems/MySQL`, but that collided with the MySQL Startup Item installed by OS X Server.) Startup Item installation adds a variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`. If you want to disable the automatic startup of MySQL, simply change this variable to `MYSQLCOM=-NO-`.

On OS X Server, the default MySQL installation uses the variable `MYSQL` in the `/etc/hostconfig` file. The MySQL Startup Item installer disables this variable by setting it to `MYSQL=-NO-`. This avoids boot time conflicts with the `MYSQLCOM` variable used by the MySQL Startup Item. However, it does not shut down a running MySQL server. You should do that yourself.

After the installation, you can start up MySQL by running the following commands in a terminal window. You must have administrator privileges to perform this task.

If you have installed the Startup Item, use this command:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

If you do not use the Startup Item, enter the following command sequence:

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
(Enter your password, if necessary)
(Press Control-Z)
shell> bg
(Press Control-D or enter "exit" to exit the shell)
```

You should be able to connect to the MySQL server, for example, by running `/usr/local/mysql/bin/mysql`.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.18.4, “Securing the Initial MySQL Accounts”.

You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

```
alias mysql=/usr/local/mysql/bin/mysql
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

For `tcsh`, use:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see Section 4.2.1, “Invoking MySQL Programs”.

If you are upgrading an existing installation, note that installing a new MySQL DMG does not remove the directory of an older installation. Unfortunately, the OS X Installer does not yet offer the functionality required to properly upgrade previously installed packages.

To use your existing databases with the new installation, you'll need to copy the contents of the old data directory to the new data directory. Make sure that neither the old server nor the new one is running when you do this. After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-`*`VERSION`*`.pkg`.

2.12 Installing MySQL on Linux Using RPM Packages

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages. The RPMs that we provide to the community should work on all versions of Linux that support RPM packages and use `glibc` 2.3. We also provide RPMs with binaries that are statically linked to a patched version of `glibc` 2.2, but only for the x86 (32-bit) architecture. To obtain RPM packages, see Section 2.5, “How to Get MySQL”.

For non-RPM Linux distributions, you can install MySQL using a `.tar.gz` package. See Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”.

We do provide some platform-specific RPMs; the difference between a platform-specific RPM and a generic RPM is that a platform-specific RPM is built on the targeted platform and is linked dynamically whereas a generic RPM is linked statically with LinuxThreads.

Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by us in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead. Because of these differences, RPM packages built by us check whether such RPMs built by other vendors are installed. If so, the RPM does not install and produces a message explaining this.

If you have problems with an RPM file (for example, if you receive the error `Sorry, the host '`*`xxxx`*`' could not be looked up`), see Section 2.20.1.2, “Linux Binary Distribution Notes”.

In most cases, you need to install only the `MySQL-server` and `MySQL-client` packages to get a functional MySQL installation. The other packages are not required for a standard installation.

For upgrades, if your installation was originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you get a dependency failure when trying to install MySQL packages (for example, `error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...`), you should also install the `MySQL-shared-compat` package, which includes the shared libraries for older releases for backward compatibility.

Some Linux distributions still ship with MySQL 3.23 and they usually link applications dynamically to save disk space. If these shared libraries are in a separate package (for example, `MySQL-shared`), it is sufficient to simply leave this package installed and just upgrade the MySQL server and client packages (which are statically linked and do not depend on the shared libraries). For distributions that include the shared libraries in the same package as the MySQL server (for example, Red Hat Linux), you could either install our 3.23 `MySQL-shared` RPM, or use the `MySQL-shared-compat` package instead. (Do not install both.)

The RPM packages shown in the following list are available. The names shown here use a suffix of `.glibc23.i386.rpm`, but particular packages can have different suffixes, as described later. Packages that have `community` in the names are Community Server builds, available from MySQL 5.0.27 on.

- `MySQL-server-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-server-community-`*`VERSION`*`.glibc23.i386.rpm`

  The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine.

- `MySQL-client-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-client-community-`*`VERSION`*`.glibc23.i386.rpm`

  The standard MySQL client programs. You probably always want to install this package.

- `MySQL-bench-`*`VERSION`*`.glibc23.i386.rpm`

  Tests and benchmarks. Requires Perl and the `DBI` and `DBD::mysql` modules.

- `MySQL-devel-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-devel-community-`*`VERSION`*`.glibc23.i386.rpm`

  The libraries and include files that are needed if to compile other MySQL clients, such as the Perl modules. Install this RPM if you intend to compile C API applications.

- `MySQL-debuginfo-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-community-debuginfo-`*`VERSION`*`.glibc23.i386.rpm`

  Debugging information. `debuginfo` RPMs are never needed to use MySQL software; this is true both for the server and for client programs. However, they contain additional information that might be needed by a debugger to analyze a crash.

- `MySQL-shared-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-shared-community-`*`VERSION`*`.glibc23.i386.rpm`

  The shared libraries (`libmysqlclient.so*`) that certain languages and applications need to dynamically load and use MySQL. It contains single-threaded and thread-safe libraries. Install this RPM if you intend to compile or run C API applications that depend on the shared client library. If you install this package, do not install the `MySQL-shared-compat` package.

- `MySQL-shared-compat-`*`VERSION`*`.glibc23.i386.rpm`

  The shared libraries for older releases, up to the current release. It contains single-threaded and thread-safe libraries. Install this package instead of `MySQL-shared` if you have applications installed that are

This documentation is for an older version. If you're

dynamically linked against older versions of MySQL but you want to upgrade to the current version without breaking the library dependencies.

- `MySQL-clustermanagement-community`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-clusterstorage-community`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-clustertools-community`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-clusterextra-community`*`VERSION`*`.glibc23.i386.rpm`

  Packages that contain additional files for MySQL Cluster installations. These are platform-specific RPMs, in contrast to the platform-independent `ndb-`*`xxx`* RPMs.

  Note

  The `MySQL-clustertools` RPM requires a working installation of perl and the `DBI` and `HTML::Template` packages. See Section 2.22, “Perl Installation Notes”, and Section 17.4.18, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”, for more information.

- `MySQL-ndb-management-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-ndb-storage-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-ndb-tools-`*`VERSION`*`.glibc23.i386.rpm`, `MySQL-ndb-extra-`*`VERSION`*`.glibc23.i386.rpm`

  Packages that contain additional files for MySQL Cluster installations. These are platform-independent RPMs, in contrast to the platform-specific `cluster`*`xxx`*`-community` RPMs.

- `MySQL-test-community-`*`VERSION`*`.glibc23.i386.rpm`

  The MySQL test suite.

- `MySQL-`*`VERSION`*`.src.rpm`

  The source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, SPARC).

The suffix of RPM package names (following the *`VERSION`* value) has the following syntax:

```
[.PLATFORM].CPU.rpm
```

The *`PLATFORM`* and *`CPU`* values indicate the type of system for which the package is built. *`PLATFORM`*, if present, indicates the platform, and *`CPU`* indicates the processor type or family.

If the *`PLATFORM`* value is missing (for example, `MySQL-server-`*`VERSION`*`.i386.rpm`), the package is statically linked against a version of `glibc` 2.2 that has been patched to handle larger numbers of threads with larger stack sizes than the stock library.

If *`PLATFORM`* is present, the package is dynamically linked against `glibc` 2.3 and the *`PLATFORM`* value indicates whether the package is platform independent or intended for a specific platform, as shown in the following table.

***`PLATFORM`* Value**	**Intended Use**
`glibc23`	Platform independent, should run on any Linux distribution that supports `glibc` 2.3
`rhel4`, `rhel5`	Red Hat Enterprise Linux 4 or 5
`sles10`	SuSE Linux Enterprise Server 10

The *`CPU`* value indicates the processor type or family for which the package is built.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

CPU Value	Intended Processor Type or Family
i386, i586, i686	Pentium processor or better, 32 bit
x86_64	64-bit x86 processor
ia64	Itanium (IA-64) processor

To see all files in an RPM package (for example, a MySQL-server RPM), run a command like this:

```
shell> rpm -qpl MySQL-server-VERSION.glibc23.i386.rpm
```

To perform a standard minimal installation, install the server and client RPMs:

```
shell> rpm -i MySQL-server-VERSION.glibc23.i386.rpm
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

To install only the client programs, install just the client RPM:

```
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. To learn more about this feature, see Section 2.6, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

The server RPM places data under the /var/lib/mysql directory. The RPM also creates a login account for a user named mysql (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in /etc/init.d/ to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you do not lose it when you install a newer RPM.) See Section 2.18.5, “Starting and Stopping MySQL Automatically”, for more information on how MySQL can be started automatically at system startup.

If the RPM files that you install include MySQL-server, the mysqld server should be up and running after installation. You should be able to start using MySQL.

If something goes wrong, you can find more information in the binary installation section. See Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.18.4, “Securing the Initial MySQL Accounts”.

During RPM installation, a user named mysql and a group named mysql are created on the system. This is done using the useradd, groupadd, and usermod commands. Those commands require appropriate administrative privileges, which is ensured for locally managed users and groups (as listed in the /etc/passwd and /etc/group files) by the RPM installation process being run by root.

If you log in as the mysql user, you may find that MySQL displays “Invalid (old?) table or database name” errors that mention .mysqlgui, lost+found, .mysqlgui, .bash_history, .fonts.cache-1, .lesshst, .mysql_history, .profile, .viminfo, and similar files created by MySQL or operating system utilities. You can safely ignore these error messages or remove the files or directories that cause them if you do not need them.

For nonlocal user management (LDAP, NIS, and so forth), the administrative tools may require additional authentication (such as a password), and will fail if the installing user does not provide this authentication.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Even if they fail, the RPM installation will not abort but succeed, and this is intentional. If they failed, some of the intended transfer of ownership may be missing, and it is recommended that the system administrator then manually ensures some appropriate user andgroup exists and manually transfers ownership following the actions in the RPM spec file.

2.13 Installing MySQL on Solaris

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, http://dev.mysql.com/downloads/mysql/5.0.html.

If you install MySQL using a binary tarball distribution on Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

You can install MySQL on Solaris using a binary package in PKG format instead of the binary tarball distribution. Before installing using the binary PKG format, you should create the `mysql` user and group, for example:

```
groupadd mysql
useradd -g mysql -s /bin/false mysql
```

Some basic PKG-handling commands follow:

- To add a package:

  ```
  pkgadd -d package_name.pkg
  ```

- To remove a package:

  ```
  pkgrm package_name
  ```

- To get a full list of installed packages:

  ```
  pkginfo
  ```

- To get detailed information for a package:

  ```
  pkginfo -l package_name
  ```

- To list the files belonging to a package:

  ```
  pkgchk -v package_name
  ```

- To get packaging information for an arbitrary file:

  ```
  pkgchk -l -p file_name
  ```

For additional information about installing MySQL on Solaris, see Section 2.20.3, “Solaris Notes”.

2.14 Installing MySQL on i5/OS

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The i5/OS POWER MySQL package was created in cooperation with IBM. MySQL works within the Portable Application Solution Environment (PASE) on the System i series of hardware and will also provide database services for the Zend Core for i5/OS.

MySQL for i5/OS is provided both as a `tar` file and as a save file (`.savf`) package that can be downloaded and installed directly without any additional installation steps required. To install MySQL using the `tar` file, see Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”.

MySQL is only supported on i5/OS V5R4 or later releases. The i5/OS PASE must be installed for MySQL to operate. You must be able to login as a user in `*SECOFR` class.

You should the installation notes and tips for i5/OS before starting installation. See i5/OS Installation Notes.

Before Installation:

Note

The installation package will use an existing configuration if you have previously installed MySQL (which is identified by looking for the file `/etc/my.cnf`). The values for the data directory (`DATADIR`) and owner of the MySQL files (`USRPRF`) specified during the installation will be ignored, and the values determined from the `/etc/my.cnf` will be used instead.

If you want to change these parameters during a new install, you should temporarily rename `/etc/my.cnf`, install MySQL using the new parameters you want to use, and then merge your previous `/etc/my.cnf` configuration settings with the new `/etc/my.cnf` file that is created during installation.

- You must have a user profile with PASE with suitable privileges. The user should be within the `*SECOFR` class, such as the `QSECOFR` user ID. You can use the `WRKUSRPRF` command to check your user profile.
- For network connections to MySQL, you must have TCP/IP enabled. You should also check the following:
  - Ensure that a name has defined for the system. Run the Configure TCP/IP (`CFGTCP`) command and select option 12 (Change TCP/IP domain information) to display this setting. Make sure that a value is listed in the Host name field.
  - Make sure that the system has a loopback entry which represents the `localhost` or `127.0.0.1`.
  - Ensure that the IP address of the IBM i machine is mapped correctly to the host name.

To install MySQL on i5/OS, follow these steps:

1. On the System i machine, create a save file that will be used to receive the downloaded installation save file. The file should be located within the General Purpose Library (`QGPL`):

   ```
   CRTSAVF FILE(QGPL/MYSQLINST) TESXT('MySQL Save file')
   ```

2. Download the MySQL installation save file in 32-bit (`mysql-5.0.82-i5os-power-32bit.savf`) or 64-bit (`mysql-5.0.82-i5os-power-64bit.savf`) from MySQL Downloads.

3. You need to FTP the downloaded `.savf` file directly into the `QGPL/MYSQLINST` file on the System i server. You can do this through FTP using the following steps after logging in to the System i machine:

   ```
   ftp> bin
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ftp> cd qgpl
ftp> put mysql-5.0.82-i5os-power.savf mysqlinst
```

4. Log into the System i server using a user in the *SECOFR class, such as the QSECOFR user ID.

5. You need to restore the installation library stored in the .savf save file:

```
RSTLIB MYSQLINST DEV(*SAVF) SAVF(QGPL/MYSQLINST) MBROPT(*ALL) ALWOBJDIF(*ALL)
```

Note

You can ignore the security changes-type message at the bottom of the installation panel.

6. Once you have finished restoring the MYSQLINST library, check that all the necessary objects for installation are on the system by using the Display Library (DSPLIB) command:

```
DSPLIB LIB(MYSQLINST)
```

7. You need to execute the installation command, MYSQLINST/INSMYSQL. You can specify three parameter settings during installation:

   - DIR('/QOpenSys/usr/local/mysql') sets the installation location for the MySQL files. The directory will be created if it does not already exist.
   - DATADIR('/QOpenSys/usr/local/mysql/data') sets the location of the directory that will be used to store the database files and binary logs. The default setting is /QOpenSys/usr/local/mysql/data. Note that if the installer detects an existing installation (due to the existence of /etc/my.cnf), then the existing setting will be used instead of the default.
   - USRPRF(*MYSQL*) sets the user profile that will own the files that are installed. The profile will be created if it does not already exist.

   Note

   You should choose an appropriate user for using the MySQL server installation. The user will be used whenever you need to do any administration on the MySQL server.

   Once you have set the appropriate parameters, you can begin the installation.

   The installation copies all the necessary files into a directory matching the DIR configuration value; sets the ownership on those files, sets up the MySQL environment and creates the MySQL configuration file (in /etc/my.cnf) completing all the steps in a typical binary installation process automatically. If this is a new installation of MySQL, or if the installer detects that this is a new version (because the /etc/my.cnf file does not exist), then the initial core MySQL databases will also be created during installation.

   Once the installation has been completed, you will get a notice advising you to set the password for the root user. For more information, Section 2.18, “Postinstallation Setup and Testing”.

8. Once the installation has completed, you can delete the installation file:

```
DLTLIB LIB(MYSQLINST)
```

Upgrading an existing MySQL instance

You need to execute the upgrade command, MYSQLINST/UPGMYSQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

You cannot use `MYSQLINST/UPGMYSQL` to upgrade between release series of MySQL (for example from 5.0 to 5.1). For information and advice on migrating between release series you can use the advice provided in Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”.

You must specify 6 parameters to perform an upgrade:

- `DIR('/QOpenSys/usr/local/')`: Sets the installation location for the MySQL files. The directory will be created if it does not already exist. This is the directory that the MySQL server will be installed into, inside a directory with a name matching the version and release. For example, if installing MySQL 5.0.82 with the `DIR` set to `/QOpenSys/usr/local/` would result in `/QOpenSys/usr/local/mysql-5.0.82-i5os-power64` and a symbolic link to this directory will be created in `/QOpenSys/usr/local/mysql`.
- `DATADIR('/QOpenSys/mysql/data')`: Sets the location of the directory that will be upgraded.
- `USRPRF('MYSQL')`: Sets the user profile that will own the files that are installed. The profile will be created if it does not already exist; if it is created as part of the upgrade process, it will be disabled initially. You may wish to enable this user profile so that it can be used to start the MySQL server later. It is best practice to use the one previously created during the first installation.
- `MYSQLUSR('root user')`: Any user account in the current MySQL server with `SUPER` privileges.
- `PASSWORD('root user password')`: The password for the above account. This is necessary as the upgrade starts the MySQL server to upgrade the tables and the password is need to be able to shutdown the MySQL server.
- `CURINST('path to previous install')`: The full path to the installation that is being upgraded. For example an installation in `/QOpenSys/usr/local/` will be `/QOpenSys/usr/local/mysql-5.1.30-i5os-power64`. Failure to specify this option may result in corruption of your existing data files.

For example:

```
MYSQLINST/UPGMYSQL DIR('/QOpenSys/usr/local/') DATADIR('/QOpenSys/mysql/data') »
     USERPRF(MYSQL) MYSQLUSR('root') PASSWORD('root') CURINST('/QOpenSys/usr/local/mysql-5.1.30-i5os-power
```

You should receive a Program Message indicating `UPGRADE SUCCESSFUL!` upon completion or an error message if there is a problem.You can view the upgrade programs progression and the error in the text file `upgrade.log` in the installation directory.

To start MySQL:

1. Log into the System i server using the user profile create or specified during installation. By default, this is `MYSQL`.

   Note

   You should start `mysqld_safe` using a user that in the PASE environment has the id=0 (the equivalent of the standard Unix `root` user). If you do not use a user with this ID then the system will be unable to change the user when executing `mysqld` as set using `--user` option. If this happens, `mysqld` may be unable to read the files located within the MySQL data directory and the execution will fail.

2. Enter the PASE environment using `call qp2term`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

3. Start the MySQL server by changing to the installation directory and running `mysqld_safe`, specifying the user name used to install the server. The installer conveniently installs a symbolic link to the installation directory (`mysql-5.0.42-i5os-power-32bit`) as `/opt/mysql/mysql`:

   ```
   > cd /opt/mysql/mysql
   > bin/mysqld_safe --user=mysql &
   ```

   You should see a message similar to the following:

   ```
   Starting mysqld daemon with databases »
        from /opt/mysql/mysql-enterprise-5.0.42-i5os-power-32bit/data
   ```

If you are having problems starting MySQL server, see Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”.

To stop MySQL:

1. Log into the System i server using the user profile create or specified during installation. By default, this is `MYSQL`.
2. Enter the PASE environment using `call qp2term`.
3. Stop the MySQL server by changing into the installation directory and running `mysqladmin`, specifying the user name used to install the server:

   ```
   > cd /opt/mysql/mysql
   > bin/mysqladmin -u root shutdown
   ```

   If the session that you started and stopped MySQL are the same, you may get the log output from `mysqld`:

   ```
      STOPPING server from pid file »
        /opt/mysql/mysql-enterprise-5.0.42-i5os-power-32bit/data/I5DBX.RCHLAND.IBM.COM.pid
      070718 10:34:20  mysqld ended
   ```

   If the sessions used to start and stop MySQL are different, you will not receive any confirmation of the shutdown.

Notes and tips

- A problem has been identified with the installation process on DBCS systems. If you are having problems install MySQL on a DBCS system, you need to change your job's coded character set identifier (`CSSID`) to 37 (`EBCDIC`) before executing the install command, `INSMYSQL`. To do this, determine your existing `CSSID` (using `DSPJOB` and selecting option 2), execute `CHGJOB CSSID(37)`, run `INSMYSQL` to install MySQL and then execute `CHGJOB` again with your original `CSSID.`
- If you want to use the Perl scripts that are included with MySQL, you need to download the iSeries Tools for Developers (5799-PTL). See http://www-03.ibm.com/servers/enable/site/porting/tools/.

2.15 Installing MySQL on NetWare

Porting MySQL to NetWare was an effort spearheaded by Novell. Novell customers should be pleased to note that NetWare 6.5 ships with bundled MySQL binaries, complete with an automatic commercial use license for all servers running that version of NetWare.

MySQL for NetWare is compiled using a combination of Metrowerks CodeWarrior for NetWare and special cross-compilation versions of the GNU autotools.

The latest binary packages for NetWare can be obtained at http://dev.mysql.com/downloads/. See Section 2.5, “How to Get MySQL”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To host MySQL, the NetWare server must meet these requirements:

- The latest Support Pack of NetWare 6.5 must be installed.
- The system must meet Novell's minimum requirements to run the respective version of NetWare.
- MySQL data and the program binaries must be installed on an NSS volume; traditional volumes are not supported.

To install MySQL for NetWare, use the following procedure:

1. If you are upgrading from a prior installation, stop the MySQL server. This is done from the server console, using the following command:

   ```
   SERVER:  mysqladmin -u root shutdown
   ```

   Note

   If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

2. Log on to the target server from a client machine with access to the location where you are installing MySQL.

3. Extract the binary package Zip file onto the server. Be sure to allow the paths in the Zip file to be used. It is safe to simply extract the file to `SYS:\`.

   If you are upgrading from a prior installation, you may need to copy the data directory (for example, `SYS:MYSQL\DATA`), as well as `my.cnf`, if you have customized it. You can then delete the old copy of MySQL.

4. You might want to rename the directory to something more consistent and easy to use. The examples in this manual use `SYS:MYSQL` to refer to the installation directory.

   Note that MySQL installation on NetWare does not detect if a version of MySQL is already installed outside the NetWare release. Therefore, if you have installed the latest MySQL version from the Web (for example, MySQL 4.1 or later) in `SYS:\MYSQL`, you must rename the folder before upgrading the NetWare server; otherwise, files in `SYS:\MySQL` are overwritten by the MySQL version present in NetWare Support Pack.

5. At the server console, add a search path for the directory containing the MySQL NLMs. For example:

   ```
   SERVER:  SEARCH ADD SYS:MYSQL\BIN
   ```

6. Initialize the data directory and the grant tables, if necessary, by executing `mysql_install_db` at the server console.

7. Start the MySQL server using `mysqld_safe` at the server console.

8. To finish the installation, you should also add the following commands to `autoexec.ncf`. For example, if your MySQL installation is in `SYS:MYSQL` and you want MySQL to start automatically, you could add these lines:

   ```
   #Starts the MySQL 5.0.x database server
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you are running MySQL on NetWare 6.0, we strongly suggest that you use the `--skip-external-locking` option on the command line:

```
#Starts the MySQL 5.0.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --skip-external-locking
```

It is also necessary to use `CHECK TABLE` and `REPAIR TABLE` instead of `myisamchk`, because `myisamchk` makes use of external locking. External locking is known to have problems on NetWare 6.0; the problem has been eliminated in NetWare 6.5. Note that the use of MySQL on Netware 6.0 is not officially supported.

`mysqld_safe` on NetWare provides a screen presence. When you unload (shut down) the `mysqld_safe` NLM, the screen does not go away by default. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the `--autoclose` option to `mysqld_safe`. For example:

```
#Starts the MySQL 5.0.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

The behavior of `mysqld_safe` on NetWare is described further in Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

9. When installing MySQL, either for the first time or upgrading from a previous version, download and install the latest and appropriate Perl module and PHP extensions for NetWare:

   - Perl: http://forge.novell.com/modules/xfcontent/downloads.php/perl/Modules/
   - PHP: http://forge.novell.com/modules/xfcontent/downloads.php/php/Modules/

If there was an existing installation of MySQL on the NetWare server, be sure to check for existing MySQL startup commands in `autoexec.ncf`, and edit or delete them as necessary.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.18, “Postinstallation Setup and Testing”.

2.16 Installing MySQL on Unix/Linux Using Generic Binaries

Oracle provides a set of binary distributions of MySQL. These include generic binary distributions in the form of compressed `tar` files (files with a `.tar.gz` extension) for a number of platforms, and binaries in platform-specific package formats for selected platforms.

This section covers the installation of MySQL from a compressed `tar` file binary distribution. For other platform-specific package formats, see the other platform-specific sections. For example, for Windows distributions, see Section 2.10, “Installing MySQL on Microsoft Windows”.

To obtain MySQL, see Section 2.5, “How to Get MySQL”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL compressed `tar` file binary distributions have names of the form `mysql-VERSION-OS.tar.gz`, where *`VERSION`* is a number (for example, `5.0.96`), and *`OS`* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

Warning

If you have previously installed MySQL using your operating system native package management system, such as `yum` or `apt-get`, you may experience problems installing using a native binary. Make sure your previous MySQL installation has been removed entirely (using your package management system), and that any additional files, such as old versions of your data files, have also been removed. You should also check for configuration files such as `/etc/my.cnf` or the `/etc/mysql` directory and delete them.

If you run into problems and need to file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

To install and use a MySQL binary distribution, the command sequence looks like this:

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
shell> cd /usr/local
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
# Next command is optional
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

Note

This procedure assumes that you have `root` (administrator) access to your system. Alternatively, you can prefix each command using the `sudo` (Linux) or `pfexec` (OpenSolaris) command.

Note

The procedure does not assign passwords to MySQL accounts. To do so, use the instructions in Section 2.18.4, “Securing the Initial MySQL Accounts”.

A more detailed version of the preceding description for installing a binary distribution follows.

Create a mysql User and Group

If your system does not already have a user and group for `mysqld` to run as, you may need to create one. The following commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

If your system does not already have a user and group to use for running `mysqld`, you may need to create one. The following commands add the `mysql` group and the `mysql` user. You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following

This documentation is for an older version. If you're

instructions. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
```

Note

Because the user is required only for ownership purposes, not login purposes, the `useradd` command uses the `-r` and `-s /bin/false` options to create a user that does not have login permissions to your server host. Omit these options if your `useradd` does not support them.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under `/usr/local`. The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.

```
shell> cd /usr/local
```

Obtain a distribution file using the instructions in Section 2.5, “How to Get MySQL”. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `tar` command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This enables you to refer more easily to the installation directory as `/usr/local/mysql`.

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
```

Perform Postinstallation Setup

The remainder of the installation process involves setting distribution ownership and access permissions, initializing the data directory, starting the MySQL server, and setting up the configuration file. For instructions, see Section 2.18, “Postinstallation Setup and Testing”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.17 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see http://www.mysql.com/support/supportedplatforms/database.html.

Before you proceed with an installation from source, check whether we produce a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”.

To obtain a source distribution for MySQL, see Section 2.5, “How to Get MySQL”. MySQL source distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-VERSION.tar.gz`, `mysql-VERSION.zip`, or `mysql-VERSION.rpm`, where *`VERSION`* is a number like `5.0.96`.

To perform a MySQL installation using the source code:

- To build MySQL from source on Unix-like systems, including Linux, commercial Unix, BSD, OS X and others using a `.tar.gz` or RPM-based source code distribution, see Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”.
- To build MySQL from source on Windows (Windows XP or newer required), see Section 2.10.8, “Installing MySQL from Source on Windows”.
- For information on building from one of our development trees, see Section 2.17.2, “Installing MySQL Using a Development Source Tree”.
- For information on using the `configure` command to specify the source build parameters, including links to platform specific parameters that you might need, see Section 2.17.3, “MySQL Source-Configuration Options”.

To install MySQL from source, the following system requirements must be satisfied:

- GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it (if you use a `.tar.gz` distribution), or `WinZip` or another tool that can read `.zip` files (if you use a `.zip` distribution).

  GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.
- A working ANSI C++ compiler. GCC 3.4.6 or later, Sun Studio 10 or later, Visual Studio 2005 or later, and many current vendor-supplied compilers are known to work.
- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or newer. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- `libtool` 1.5, available from http://www.gnu.org/software/libtool/. 1.5.24 or later is recommended.

If you are using a version of `gcc` recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. Also use `-felide-constructors` and `-fno-rtti` along with `-fno-exceptions`. When in doubt, do the following:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

If you run into problems and need to file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

2.17.1 Installing MySQL Using a Standard Source Distribution

To install MySQL from source, first configure, build, and install from a source package. Then follow the same postinstallation setup sequence as for a binary installation.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Section 2.12, “Installing MySQL on Linux Using RPM Packages”.

The sequence for installation from a compressed `tar` file source distribution is similar to the process for installing from a generic binary distribution that is detailed in Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”. For a MySQL `.tar.gz` source distribution, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -g mysql -s /bin/false mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
# Next command is optional
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

Note

This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.18, “Postinstallation Setup and Testing”, for postinstallation setup and testing.

A more detailed version of the preceding description for installing MySQL from a source distribution follows:

1. Add a login user and group for `mysqld` to run as:

This documentation is for an older version. If you're

```
shell> groupadd mysql
shell> useradd -g mysql -s /bin/false mysql
```

These commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Perform the following steps as the `mysql` user, except as noted.
3. Pick the directory under which you want to unpack the distribution and change location into it.
4. Obtain a distribution file using the instructions in Section 2.5, “How to Get MySQL”.
5. Unpack the distribution into the current directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

   ```
   shell> tar zxvf /path/to/mysql-VERSION.tar.gz
   ```

   This command creates a directory named `mysql-VERSION`.

   If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

6. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd mysql-VERSION
   ```

   Note that currently you must configure and build MySQL from this top-level directory. You cannot build it in a different directory.

7. Configure the release and compile everything:

   ```
   shell> ./configure --prefix=/usr/local/mysql
   shell> make
   ```

   When you run `configure`, you might want to specify other options. For example, if you need to debug `mysqld` or a MySQL client, run `configure` with the `--with-debug` option, and then recompile and link your clients with the new client library. See Section 21.3, “Debugging and Porting MySQL”.

   Run `./configure --help` for a list of options. Section 2.17.3, “MySQL Source-Configuration Options”, discusses some of the more useful options.

   If `configure` fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from `configure`. To file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

   If the compile fails, see Section 2.17.4, “Dealing with Problems Compiling MySQL”, for help.

8. Install the distribution:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> make install
```

You might need to run this command as `root`.

If you want to set up an option file, use one of those present in the `support-files` directory as a template. For example:

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

You might need to run this command as `root`.

If you want to configure support for `InnoDB` tables, you should edit the `/etc/my.cnf` file, removing the `#` character before the option lines that start with `innodb_...`, and modify the option values to be what you want. See Section 4.2.6, “Using Option Files”, and Section 14.2.1, “Configuring InnoDB”.

9. Change location into the installation directory:

```
shell> cd /usr/local/mysql
```

10. If you ran the `make install` command as `root`, the installed files will be owned by `root`. Ensure that the installation is accessible to `mysql` by executing the following commands as `root` in the installation directory:

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

11. If you have not installed MySQL before, you must create the MySQL data directory and initialize the grant tables:

```
shell> bin/mysql_install_db --user=mysql
```

If you run the command as `root`, include the `--user` option as shown. If you run the command while logged in as `mysql`, you can omit the `--user` option.

The command should create the data directory and its contents with `mysql` as the owner.

After using `mysql_install_db` to create the grant tables for MySQL, you must restart the server manually. The `mysqld_safe` command to do this is shown in a later step.

12. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql var
```

13. If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

14. If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself; see also Section 2.18.5, “Starting and Stopping MySQL Automatically”.

15. You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD::mysql` Perl modules. See Section 4.6.15, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”. For Perl module installation instructions, see Section 2.22, “Perl Installation Notes”.

After everything has been installed, test the distribution. To start the MySQL server, use the following command:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

If you run the command as `root`, you should use the `--user` option as shown. The option value is the name of the login account that you created in the first step to use for running the server. If you run the `mysqld_safe` command while logged in as that user, you can omit the `--user` option.

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

More information about `mysqld_safe` is given in Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

To make it more convenient to invoke programs installed in `/usr/local/mysql/bin`, you can add that directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. See Section 4.2.10, “Setting Environment Variables”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.18, “Postinstallation Setup and Testing”.

2.17.2 Installing MySQL Using a Development Source Tree

This section discusses how to install MySQL from the latest development source code.

To obtain the source tree, you must have Bazaar installed. The Bazaar VCS Web site has instructions for downloading and installing Bazaar on different platforms. Bazaar is supported on any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows, or OS X host.

MySQL development projects are hosted on Launchpad. MySQL projects, including MySQL Server, MySQL Workbench, and others are available from the Oracle/MySQL Engineering page. For the repositories related only to MySQL Server, see the MySQL Server page.

To build under Unix/Linux, you must have the following tools installed:

- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or newer. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- `autoconf` 2.58 (or newer), available from http://www.gnu.org/software/autoconf/.
- `automake` 1.8.1, available from http://www.gnu.org/software/automake/.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `libtool` 1.5, available from http://www.gnu.org/software/libtool/. 1.5.24 or later is recommended.
- `m4`, available from http://www.gnu.org/software/m4/.
- `bison`, available from http://www.gnu.org/software/bison/. You should use the latest version of `bison` where possible. Versions 1.75 and 2.1 are known to work. There have been reported problems with `bison` 1.875. If you experience problems, upgrade to a later, rather than earlier, version.

To build under Windows you must have Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.

Once the necessary tools are installed, create a local branch of the MySQL development tree on your machine using this procedure:

1. To obtain a copy of the MySQL source code, you must create a new Bazaar branch. If you do not already have a Bazaar repository directory set up, you must initialize a new directory:

   ```
   shell> mkdir mysql-server
   shell> bzr init-repo --trees mysql-server
   ```

   This is a one-time operation.

2. Assuming that you have an initialized repository directory, you can branch from the public MySQL server repositories to create a local source tree. To create a branch of a specific version:

   ```
   shell> cd mysql-server
   shell> bzr branch lp:mysql-server/5.0 mysql-5.0
   ```

   This is a one-time operation per source tree. You can branch the source trees for several versions of MySQL under the `mysql-server` directory.

3. The initial download will take some time to complete, depending on the speed of your connection. Please be patient. Once you have downloaded the first tree, additional trees should take significantly less time to download.

4. When building from the Bazaar branch, you may want to create a copy of your active branch so that you can make configuration and other changes without affecting the original branch contents. You can achieve this by branching from the original branch:

   ```
   shell> bzr branch mysql-5.0 mysql-5.0-build
   ```

5. To obtain changes made after you have set up the branch initially, update it using the `pull` option periodically. Use this command in the top-level directory of the local copy:

   ```
   shell> bzr pull
   ```

   You can examine the changeset comments for the tree by using the `log` option to `bzr`:

   ```
   shell> bzr log
   ```

   You can also browse changesets, comments, and source code online at the Launchpad MySQL Server page.

   If you see diffs (changes) or code that you have a question about, do not hesitate to send email to the MySQL `internals` mailing list. See Section 1.6.1, “MySQL Mailing Lists”. If you think you have a better idea on how to do something, send an email message to the list with a patch.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

After you have the local branch, you can build MySQL server from the source code. On Windows, the build process is different from Unix/Linux: see Section 2.10.8, “Installing MySQL from Source on Windows”.

On Unix/Linux, use the `autoconf` system to create the `configure` script so that you can configure the build environment before building. The following example shows the typical commands required to build MySQL from a source tree.

1. Change location to the top-level directory of the source tree; replace `mysql-5.0` with the appropriate directory name.

   ```
   shell> cd mysql-5.0
   ```

2. Prepare the source tree for configuration.

   You must separately configure the `BDB` and `InnoDB` storage engines. Run the following commands from the main source directory:

   ```
   shell> (cd bdb/dist; sh s_all)
   shell> (cd innobase; autoreconf --force --install)
   ```

   You can omit the previous commands if you do not require `BDB` or `InnoDB` support.

   Prepare the remainder of the source tree:

   ```
   shell> autoreconf --force --install
   ```

   As an alternative to the preceding `autoreconf` command, you can use `BUILD/autorun.sh`, which acts as a shortcut for the following sequence of commands:

   ```
   shell> aclocal; autoheader
   shell> libtoolize --automake --force
   shell> automake --force --add-missing; autoconf
   shell> (cd bdb/dist; sh s_all)
   shell> (cd innobase; aclocal; autoheader; autoconf; automake)
   ```

   If you get some strange errors during this stage, verify that you have the correct version of `libtool` installed.

3. Configure the source tree and compile MySQL:

   ```
   shell> ./configure  # Add your favorite options here
   shell> make
   ```

   For a description of some `configure` options, see Section 2.17.3, “MySQL Source-Configuration Options”.

   A collection of configuration scripts is located in the `BUILD/` subdirectory. For example, you may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific, or use another script that may be more appropriate. These scripts are provided on an “as-is” basis. They are not supported and their contents may change from release to release.

4. When the build is done, run `make install`. Be careful with this on a production machine; the installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `./configure` with values for the `--prefix`, `--with-tcp-`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`port`, and `--with-unix-socket-path` options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Section 5.5, “Running Multiple MySQL Instances on One Machine”.

5. Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See Section 21.1.2, “The MySQL Test Suite”.

6. If you have gotten to the `make` stage, but the distribution does not compile, please enter the problem into our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a `command not found` error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

2.17.3 MySQL Source-Configuration Options

The `configure` script provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `configure` command line. For a full list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

You can also affect `configure` using certain environment variables. See Section 2.21, “Environment Variables”.

The following table shows the available `configure` options.

Table 2.7 MySQL Source-Configuration Option Reference (`configure`)

Formats	Description	Default	Introduced	Removed
`--bindir`	User executables	`EPREFIX/bin`		
`--build`	Configure for building on BUILD	`guessed`		
`--cache-file`	Cache test results in FILE	`disabled`		
`--config-cache`	Alias for `--cache-file=config.cache'			
`--datadir`	Read-only architecture-independent data	`PREFIX/share`		
`--disable-FEATURE`	Do not include FEATURE			
`--disable-community-features`	Disable additional features provided by the community		5.0.82	
`--disable-dependency-tracking`	Disable dependency tracking			
`--disable-grant-options`	Disable GRANT options		5.0.34	
`--disable-largefile`	Omit support for large files			
`--disable-libtool-lock`	Disable libtool lock			
`--disable-profiling`	Build a version without query profiling code		5.0.37	5.0.45
`--enable-FEATURE`	Enable FEATURE			

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Formats	Description	Default	Introduced	Removed
`--enable-assembler`	Use assembler versions of some string functions if available			
`--enable-dependency-tracking`	Do not reject slow dependency extractors			
`--enable-fast-install`	Optimize for fast installation	`yes`		
`--enable-local-infile`	Enable LOCAL for LOAD DATA INFILE	`disabled`		
`--enable-shared`	Build shared libraries	`yes`		
`--enable-static`	Build static libraries	`yes`		
`--enable-thread-safe-client`	Compile the client with threads			
`--exec-prefix`	Install architecture-dependent files in EPREFIX			
`--help`	Display help message and exit			
`--host`	Cross-compile to build programs to run on HOST			
`--includedir`	C header files	`PREFIX/include`		
`--infodir`	Info documentation	`PREFIX/info`		
`--libdir`	Object code libraries	`EPREFIX/lib`		
`--libexecdir`	Program executables	`EPREFIX/libexec`		
`--localstatedir`	Modifiable single-machine data	`PREFIX/var`		
`--mandir`	man documentation	`PREFIX/man`		
`--no-create`	Do not create output files			
`--oldincludedir`	C header files for non-gcc	`/usr/include`		
`--prefix`	Install architecture-independent files in PREFIX			
`--program-prefix`	Prepend PREFIX to installed program names			
`--program-suffix`	Append SUFFIX to installed program names			
`--program-transform-name`	run sed PROGRAM on installed program names			
`--quiet`	Do not print `checking...' messages			
`--sbindir`	System administrative executables	`EPREFIX/sbin`		
`--sharedstatedir`	Modifiable architecture-independent data	`PREFIX/com`		
`--srcdir`	Find the sources in DIR	`configure directory or ..`		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Formats	Description	Default	Introduced	Removed
`--sysconfdir`	Read-only single-machine data	`PREFIX/etc`		
`--target`	Configure for building compilers for TARGET			
`--version`	Display version information and exit			
`--with-PACKAGE`	Use PACKAGE			
`--with-archive-storage-engine`	Enable the Archive Storage Engine	`no`		
`--with-berkeley-db`	Use BerkeleyDB located in DIR	`no`		
`--with-berkeley-db-includes`	Find Berkeley DB headers in DIR			
`--with-berkeley-db-libs`	Find Berkeley DB libraries in DIR			
`--with-big-tables`	Support tables with more than 4 G rows even on 32 bit platforms		5.0.4	
`--with-blackhole-storage-engine`	Enable the Blackhole Storage Engine	`no`	5.0.4	
`--with-charset`	Default character set			
`--with-client-ldflags`	Extra linking arguments for clients			
`--with-collation`	Default collation			
`--with-comment`	Comment about compilation environment			
`--with-csv-storage-engine`	Enable the CSV Storage Engine	`yes`		
`--with-darwin-mwcc`	Use Metrowerks CodeWarrior wrappers on OS X/Darwin		5.0.6	
`--with-embedded-privilege-control`	Build parts to check user's privileges (only affects embedded library)			
`--with-embedded-server`	Build the embedded server			
`--with-example-storage-engine`	Enable the Example Storage Engine	`no`		
`--with-extra-charsets`	Use charsets in addition to default			
`--with-gnu-ld`	Assume the C compiler uses GNU ld	`no`		
`--with-isam`	Enable the ISAM table type			5.0.2
`--with-lib-ccflags`	Extra CC options for libraries			

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Formats	Description	Default	Introduced	Removed
`--with-libwrap`	Compile in libwrap (tcp_wrappers) support			
`--with-low-memory`	Try to use less memory to compile to avoid memory limitations			
`--with-machine-type`	Set the machine type, like "powerpc"		5.0.44	
`--with-max-indexes`	Sets the maximum number of indexes per table	`64`		
`--with-mit-threads`	Always use included thread lib			5.0.4
`--with-mysqld-ldflags`	Extra linking arguments for mysqld			
`--with-mysqld-libs`	Extra libraries to link with for mysqld		5.0.44	
`--with-mysqld-user`	What user the mysqld daemon shall be run as			
`--with-mysqlfs`	Include the corba-based MySQL file system			5.0.3
`--with-mysqlmanager`	Build the mysqlmanager binary	`Build if server is built`		
`--with-named-curses-libs`	Use specified curses libraries			
`--with-named-thread-libs`	Use specified thread libraries			
`--with-ndb-ccflags`	Extra CC options for ndb compile		5.0.3	
`--with-ndb-docs`	Include the NDB Cluster ndbapi and mgmapi documentation			
`--with-ndb-port`	Port for NDB Cluster management server			
`--with-ndb-port-base`	Port for NDB Cluster management server		5.0.3	
`--with-ndb-sci`	Provide MySQL with a custom location of sci library			
`--with-ndb-shm`	Include the NDB Cluster shared memory transporter			5.0.2
`--with-ndb-test`	Include the NDB Cluster ndbapi test programs			
`--with-ndbcluster`	Include the NDB Cluster table handler	`no`		
`--with-openssl`	Include the OpenSSL support			
`--with-openssl-includes`	Find OpenSSL headers in DIR			

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Formats	Description	Default	Introduced	Removed
--with-openssl-libs	Find OpenSSL libraries in DIR			
--with-other-libc	Link against libc and other standard libraries installed in the specified nonstandard location			
--with-pic	Try to use only PIC/non-PIC objects	Use both		
--with-pstack	Use the pstack backtrace library			
--with-pthread	Force use of pthread library			
--with-raid	Enable RAID Support			5.0.3
--with-server-suffix	Append value to the version string			
--with-system-type	Set the system type, like "sun-solaris10"		5.0.44	
--with-tags	Include additional configurations	automatic		
--with-tcp-port	Which port to use for MySQL services	3306		
--with-unix-socket-path	Where to put the unix-domain socket			
--with-vio	Include the Virtual IO support			5.0.2
--with-yassl	Include the yaSSL support		5.0.6	
--with-zlib-dir	Provide MySQL with a custom location of compression library			
--without-PACKAGE	Do not use PACKAGE			
--without-bench	Skip building of the benchmark suite			
--without-debug	Build a production version without debugging code			
--without-docs	Skip building of the documentation			
--without-extra-tools	Skip building utilities in the tools directory			
--without-geometry	Do not build geometry-related parts			
--without-innodb	Do not include the InnoDB table handler			5.0.48
--without-libedit	Use system libedit instead of bundled copy			
--without-man	Skip building of the man pages			
--without-ndb-debug	Disable special ndb debug features		5.0.3	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Formats	Description	Default	Introduced	Removed
`--without-query-cache`	Do not build query cache			
`--without-readline`	Use system readline instead of bundled copy			
`--without-server`	Only build the client			
`--without-uca`	Skip building of the national Unicode collations		5.0.3	

Some of the `configure` options available are described here. For options that may be of use if you have difficulties building MySQL, see Section 2.17.4, “Dealing with Problems Compiling MySQL”.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `--prefix`, `--with-tcp-port`, and `with-unix-socket-path` options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir`, `--port`, and `--socket` options for `mysqld`.

- To compile just the MySQL client libraries and client programs and not the server, use the `--without-server` option:

```
shell> ./configure --without-server
```

  If you have no C++ compiler, some client programs such as `mysql` cannot be compiled because they require C++. In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` option. The compile step should still try to build all clients, but you can ignore any warnings about files such as `mysql.cc`. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- To build the embedded MySQL library (`libmysqld.a`), use the `--with-embedded-server` option.

- To place your log files and database directories elsewhere than under `/usr/local/var`, use a `configure` command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
          --localstatedir=/usr/local/mysql/data
```

  The first command changes the installation prefix so that everything is installed under `/usr/local/mysql` rather than the default of `/usr/local`. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `/usr/local/var`) and changes it to `/usr/local/mysql/data`.

  You can also specify the installation directory and data directory locations at server startup time by using the `--basedir` and `--datadir` options. These can be given on the command line or in an MySQL option file, although it is more common to use an option file. See Section 4.2.6, “Using Option Files”.

- The `--with-tcp-port` option specifies the port number on which the server listens for TCP/IP connections. The default is port 3306. To listen on a different port, use a `configure` command like this:

```
shell> ./configure --with-tcp-port=3307
```

- On Unix, if you want the MySQL socket file location to be somewhere other than the default location (normally in the directory `/tmp` or `/var/run`), use a `configure` command like this:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> ./configure \
          --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

The socket file name must be an absolute path name. You can also change the location of `mysql.sock` at server startup by using a MySQL option file. See Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”.

- To compile statically linked programs (for example, to make a binary distribution, to get better performance, or to work around problems with some Red Hat Linux distributions), run `configure` like this:

```
shell> ./configure --with-client-ldflags=-all-static \
          --with-mysqld-ldflags=-all-static
```

- If you are using `gcc` and do not have `libg++` or `libstdc++` installed, you can tell `configure` to use `gcc` as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

When you use `gcc` as your C++ compiler, it does not attempt to link in `libg++` or `libstdc++`. This may be a good thing to do even if you have those libraries installed. Some versions of them have caused strange problems for MySQL users in the past.

The following list indicates some compilers and environment variable settings that are commonly used with each one.

- `gcc` 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

- `gcc` 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

In most cases, you can get a reasonably optimized MySQL binary by using the options from the preceding list and adding the following options to the `configure` line:

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The full `configure` line would, in other words, be something like the following for all recent `gcc` versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at http://dev.mysql.com/downloads/ are all compiled with full optimization and should work well for most users. See Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”.

- If the build fails and produces errors about your compiler or linker not being able to create the shared library `libmysqlclient.so.N` (where `N` is a version number), you can work around this problem by

This documentation is for an older version. If you're
This documentation is for an older version. If you're

giving the `--disable-shared` option to `configure`. In this case, `configure` does not build a shared `libmysqlclient.so.N` library.

- By default, MySQL uses the `latin1` (cp1252 West European) character set. To change the default set, use the `--with-charset` option:

```
shell> ./configure --with-charset=CHARSET
```

`CHARSET` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`. (Additional character sets might be available. Check the output from `./configure --help` for the current list.)

The default collation may also be specified. MySQL uses the `latin1_swedish_ci` collation by default. To change this, use the `--with-collation` option:

```
shell> ./configure --with-collation=COLLATION
```

To change both the character set and the collation, use both the `--with-charset` and `--with-collation` options. The collation must be a legal collation for the character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.)

With the `configure` option `--with-extra-charsets=LIST`, you can define which additional character sets should be compiled into the server. *`LIST`* is one of the following:

- A list of character set names separated by spaces
- `complex` to include all character sets that can't be dynamically loaded
- `all` to include all character sets into the binaries

Clients that want to convert characters between the server and the client should use the `SET NAMES` statement. See Section 10.1.4, “Connection Character Sets and Collations”.

- To configure MySQL with debugging code, use the `--with-debug` option:

```
shell> ./configure --with-debug
```

This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See Section 21.3, “Debugging and Porting MySQL”.

As of MySQL 5.0.25, using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- If your client programs are using threads, you must compile a thread-safe version of the MySQL client library with the `--enable-thread-safe-client` configure option. This creates a `libmysqlclient_r` library with which you should link your threaded applications. See Section 20.6.4.2, “Writing C API Threaded Client Programs”.

- Some features require that the server be built with compression library support, such as the `COMPRESS()` and `UNCOMPRESS()` functions, and compression of the client/server protocol. The `--with-zlib-dir=no|bundled|DIR` option provides control over compression library support. The

This documentation is for an older version. If you're
This documentation is for an older version. If you're

value `no` explicitly disables compression support. `bundled` causes the `zlib` library bundled in the MySQL sources to be used. A *`DIR`* path name specifies the directory in which to find the compression library sources.

- It is possible to build MySQL with large table support using the `--with-big-tables` option, beginning with MySQL 5.0.4.

  This option causes the variables that store table row counts to be declared as `unsigned long long` rather than `unsigned long`. This enables tables to hold up to approximately 1.844E+19 ($(2^{32})^2$) rows rather than 2^{32} (~4.295E+09) rows. Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable this feature.

- Run `configure` with the `--disable-grant-options` option to cause the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld` to be disabled. For Windows, the `configure.js` script recognizes the `DISABLE_GRANT_OPTIONS` flag, which has the same effect. The capability is available as of MySQL 5.0.34.

- This option allows MySQL Community Server features to be enabled. Additional options may be required for individual features, such as `--enable-profiling` to enable statement profiling. This option was added in MySQL 5.0.82.

- In MySQL Community Server, this option enables the statement profiling capability exposed by the `SHOW PROFILE` and `SHOW PROFILES` statements. (See Section 13.7.5.29, “SHOW PROFILES Syntax”.) The option was added in MySQL 5.0.37.

- See Section 2.20, “Operating System-Specific Notes”, for options that pertain to particular operating systems.

- See Section 6.3.6.2, “Building MySQL with Support for Secure Connections”, for options that pertain to configuring MySQL to support secure (encrypted) connections.

2.17.4 Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using `gcc`. On other systems, warnings may occur due to differences in system include files. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.

- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before re-running `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run `make distclean`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following list describes some of the problems that have been found to occur most often when compiling MySQL:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

```
Internal compiler error: program cc1plus got fatal signal 11
Out of virtual memory
Virtual memory exhausted
```

  The problem is that `gcc` requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running `configure` with the `--with-low-memory` option:

```
shell> ./configure --with-low-memory
```

  This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

```
configure: error: installation or configuration problem:
C++ compiler cannot create executables.
```

  You might also observe problems during compilation related to `g++`, `libg++`, or `libstdc++`.

  One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler did not work. To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

```
shell> CXX="gcc -O3" ./configure
```

  This works because `gcc` compiles C++ source files as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

  Another way to fix these problems is to install `g++`, `libg++`, and `libstdc++`. However, do not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of `mysqld` without providing any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

- To define flags to be used by your C or C++ compilers, specify them using the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To see what flags you might need to specify, invoke `mysql_config` with the `--cflags` option.

- If you get errors such as those shown here when compiling `mysqld`, `configure` did not correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenced
     type of the pointer value ''length'' is ''unsigned long'',
     which is not compatible with ''int''.
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by `configure`). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (You must do this each time you run `configure` because `configure` regenerates `config.h`.)

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

```
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs.

GNU `make` 3.75 is known to work.

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

Versions of `bison` older than 1.75 may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- On Debian Linux 3.0, you need to install `gawk` instead of the default `mawk` if you want to compile MySQL with Berkeley DB support.
- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one, you probably do not have `g++` installed:

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer
without a cast
make[2]: *** [libmysql.lo] Error 1
```

  By default, the `configure` script attempts to determine the correct number of arguments by using `g++` (the GNU C++ compiler). This test yields incorrect results if `g++` is not installed. There are two ways to work around this problem:

  - Make sure that the GNU C++ `g++` is installed. On some Linux distributions, the required package is called `gpp`; on others, it is named `gcc-c++`.
  - Use `gcc` as your C++ compiler by setting the `CXX` environment variable to `gcc`:

```
export CXX="gcc"
```

  You must run `configure` again after making either of those changes.

For information about acquiring or updating tools, see the system requirements in Section 2.17, “Installing MySQL from Source”.

2.17.5 Compiling and Linking an Optimized mysqld Server

Most of the following tests were performed on Linux with the MySQL benchmarks, but they should give some indication for other operating systems and workloads.

You obtain the fastest executables when you link with `-static`.

By using better compiler and compilation options, you can obtain a 10% to 30% speed increase in applications. This is particularly important if you compile the MySQL server yourself.

When we tested both the Cygnus CodeFusion and Fujitsu compilers, neither was sufficiently bug-free to enable MySQL to be compiled with optimizations enabled.

The standard MySQL binary distributions are compiled with support for all character sets. When you compile MySQL yourself, you should include support only for the character sets that you are going to use. This is controlled by the `--with-charset` option to `configure`.

Here is a list of some measurements that we have made:

- If you link dynamically (without `-static`), the result is 13% slower on Linux. Note that you still can use a dynamically linked MySQL library for your client applications. It is the server that is most critical for performance.
- For a connection from a client to a server running on the same host, if you connect using TCP/IP rather than a Unix socket file, performance is 7.5% slower. (On Unix, if you connect to the host name `localhost`, MySQL uses a socket file by default.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- For TCP/IP connections from a client to a server, connecting to a remote server on another host is 8% to 11% slower than connecting to a server on the same host, even for connections faster than 100Mb/s Ethernet.
- When running our benchmark tests using secure connections (all data encrypted with internal SSL support) performance was 55% slower than with unencrypted connections.
- If you compile with `--with-debug=full`, most queries are 20% slower. Some queries may take substantially longer; for example, the MySQL benchmarks run 35% slower. If you use `--with-debug` (without `=full`), the speed decrease is only 15%. For a version of `mysqld` that has been compiled with `--with-debug=full`, you can disable memory checking at runtime by starting it with the `--skip-safemalloc` option. The execution speed should then be close to that obtained when configuring with `--with-debug`.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster than one compiled with `gcc` 3.2.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster in 32-bit mode than in 64-bit mode.
- Compiling with `gcc` 2.95.2 for UltraSPARC with the `-mcpu=v8 -Wa,-xarch=v8plusa` options gives 4% better performance.
- Compiling on Linux-x86 using `gcc` without frame pointers (`-fomit-frame-pointer` or `-fomit-frame-pointer -ffixed-ebp`) makes `mysqld` 1% to 4% faster.

2.18 Postinstallation Setup and Testing

This section discusses tasks that you should perform after installing MySQL:

- If necessary, initialize the data directory and create the MySQL grant tables. For some MySQL installation methods, data directory initialization may be done for you automatically:
  - Installation on Windows
  - Installation on Linux using a server RPM distribution.
  - Installation on OS X using a DMG distribution.

  For other platforms and installation types, including installation from generic binary and source distributions, you must initialize the data directory yourself. For instructions, see Section 2.18.1, “Initializing the Data Directory”.
- For instructions, see Section 2.18.2, “Starting the Server”, and Section 2.18.3, “Testing the Server”.
- Assign passwords to any initial accounts in the grant tables, if that was not already done during data directory initialization. Passwords prevent unauthorized access to the MySQL server. You may also wish to restrict access to test databases. For instructions, see Section 2.18.4, “Securing the Initial MySQL Accounts”.
- Optionally, arrange for the server to start and stop automatically when your system starts and stops. For instructions, see Section 2.18.5, “Starting and Stopping MySQL Automatically”.
- Optionally, populate time zone tables to enable recognition of named time zones. For instructions, see Section 10.6, “MySQL Server Time Zone Support”.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Section 6.2, “The MySQL Access Privilege System”, and Section 6.3, “MySQL User Account Management”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.18.1 Initializing the Data Directory

After installing MySQL, you must initialize the data directory, including the tables in the `mysql` system database. For some MySQL installation methods, data directory initialization may be done automatically, as described in Section 2.18, “Postinstallation Setup and Testing”. For other installation methods, including installation from generic binary and source distributions, you must initialize the data directory yourself.

This section describes how to initialize the data directory on Unix and Unix-like systems. (For Windows, see Section 2.10.6, “Windows Postinstallation Procedures”.) For some suggested commands that you can use to test whether the server is accessible and working properly, see Section 2.18.3, “Testing the Server”.

In the examples shown here, the server runs under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server. For information about creating the account, see Creating a `mysql` System User and Group, in Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”.

1. Change location into the top-level directory of your MySQL installation, represented here by *`BASEDIR`*:

   ```
   shell> cd BASEDIR
   ```

   `BASEDIR` is likely to be something like `/usr/local/mysql` or `/usr/local`. The following steps assume that you have changed location to this directory.

   You will find several files and subdirectories in the *`BASEDIR`* directory. The most important for installation purposes are the `bin` and `scripts` subdirectories, which contain the server as well as client and utility programs.

   For some distribution types, `mysqld` is installed in the `libexec` directory.

2. If necessary, ensure that the distribution contents are accessible to `mysql`. If you installed the distribution as `mysql`, no further action is required. If you installed the distribution as `root`, its contents will be owned by `root`. Change its ownership to `mysql` by executing the following commands as `root` in the installation directory. The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

   ```
   shell> chown -R mysql .
   shell> chgrp -R mysql .
   ```

3. If necessary, initialize the data directory, including the `mysql` database containing the initial MySQL grant tables that determine how users are permitted to connect to the server.

   Typically, data directory initialization need be done only the first time you install MySQL. If you are upgrading an existing installation, you should run `mysql_upgrade` instead (see Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”). However, the command that initializes the data directory does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

   The exact location of `mysql_install_db` depends on the layout for your given installation. To initialize the grant tables, use one of the following commands, depending on whether `mysql_install_db` is located in the `bin` or `scripts` directory:

   ```
   shell> bin/mysql_install_db --user=mysql
   shell> scripts/mysql_install_db --user=mysql
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this if you run `mysql_install_db` as `root`, include the `--user` option as shown. Otherwise, you should execute the program while logged in as `mysql`, in which case you can omit the `--user` option from the command.

The `mysql_install_db` command creates the server's data directory. Under the data directory, it creates directories for the `mysql` database that holds the grant tables and the `test` database that you can use to test MySQL. The program also creates privilege table entries for the initial account or accounts. `test_`. For a complete listing and description of the grant tables, see Section 6.2, “The MySQL Access Privilege System”.

It might be necessary to specify other options such as `--basedir` or `--datadir` if `mysql_install_db` does not identify the correct locations for the installation directory or data directory. For example:

```
shell> bin/mysql_install_db --user=mysql \
         --basedir=/opt/mysql/mysql \
         --datadir=/opt/mysql/mysql/data
```

If you do not want to have the `test` database, you can remove it after starting the server, using the instructions in Section 2.18.4, “Securing the Initial MySQL Accounts”.

If you have trouble with `mysql_install_db` at this point, see Section 2.18.1.1, “Problems Running mysql_install_db”.

4. After initializing the data directory, you can establish the final installation ownership settings. To leave the installation owned by `mysql`, no action is required here. Otherwise, most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory. For some distribution types, the data directory might be named `var` rather than `data`; adjust the second command accordingly.

   ```
   shell> chown -R root .
   shell> chown -R mysql data
   ```

   If the plugin directory (the directory named by the `plugin_dir` system variable) is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making the plugin directory read only to the server or by setting the `secure_file_priv` system variable at server startup to a directory where `SELECT` writes can be performed safely.

5. If you installed MySQL using a source distribution, you may want to optionally copy one of the provided configuration files from the `support-files` directory into your `/etc` directory. There are different sample configuration files for different use cases, server types, and CPU and RAM configurations. To use one of these standard files, copy it to `/etc/my.cnf`, or `/etc/mysql/my.cnf` and edit and check the configuration before starting your MySQL server for the first time.

   You can also create `my.cnf` yourself and place into it the options the server should use at startup. See Section 5.1.2, “Server Configuration Defaults”.

   If you do not copy one of the standard configuration files or create your own, the MySQL server starts with its default settings.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6. If you want MySQL to start automatically when you boot your machine, see Section 2.18.5, “Starting and Stopping MySQL Automatically”.

Data directory initialization creates time zone tables in the `mysql` database but does not populate them. To do so, use the instructions in Section 10.6, “MySQL Server Time Zone Support”.

2.18.1.1 Problems Running mysql_install_db

The purpose of the `mysql_install_db` program is to initialize the data directory, including the tables in the `mysql` system database. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

To re-create your privilege tables, first stop the `mysqld` server if it is running. Then rename the `mysql` directory under the data directory to save it, and run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
shell> mv data/mysql data/mysql.old
shell> bin/mysql_install_db --user=mysql
```

When you run `mysql_install_db`, you might encounter the following problems:

- **`mysql_install_db` fails to install the grant tables**

  You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

  ```
  Starting mysqld daemon with databases from XXXXXX
  mysqld ended
  ```

  In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message and should indicate why `mysqld` did not start. If you do not understand what happened, include the log when you post a bug report. See Section 1.7, “How to Report Bugs or Problems”.

- **There is a `mysqld` process running**

  This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run `mysql_install_db` at all because it needs to be run only once, when you first install MySQL.

- **Installing a second `mysqld` server does not work when one server is running**

  This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

  ```
  Can't start server: Bind on TCP/IP port:
  Address already in use
  Can't start server: Bind on unix socket...
  ```

  For instructions on setting up multiple servers, see Section 5.5, “Running Multiple MySQL Instances on One Machine”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- **You do not have write access to the `/tmp` directory**

  If you do not have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory), an error occurs when you run `mysql_install_db` or the `mysqld` server.

  You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting `mysql_install_db` or `mysqld`, where *`some_tmp_dir`* is the full path name to some directory for which you have write permission:

  ```
  shell> TMPDIR=/some_tmp_dir/
  shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
  shell> export TMPDIR MYSQL_UNIX_PORT
  ```

  Then you should be able to run `mysql_install_db` and start the server with these commands:

  ```
  shell> bin/mysql_install_db --user=mysql
  shell> bin/mysqld_safe --user=mysql &
  ```

  If `mysql_install_db` is located in the `scripts` directory, modify the first command to `scripts/mysql_install_db`.

  See Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”, and Section 2.21, “Environment Variables”.

There are some alternatives to running the `mysql_install_db` program provided in the MySQL distribution:

- If you want the initial privileges to be different from the standard defaults, use account-management statements such as `CREATE USER`, `GRANT`, and `REVOKE` to change the privileges *after* the grant tables have been set up. In other words, run `mysql_install_db`, and then use `mysql -u root mysql` to connect to the server as the MySQL `root` user so that you can issue the necessary statements. (See Section 13.7.1, “Account Management Statements”.)

  To install MySQL on several machines with the same privileges, put the `CREATE USER`, `GRANT`, and `REVOKE` statements in a file and execute the file as a script using `mysql` after running `mysql_install_db`. For example:

  ```
  shell> bin/mysql_install_db --user=mysql
  shell> bin/mysql -u root < your_script_file
  ```

  This enables you to avoid issuing the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you are just learning how to use `CREATE USER`, `GRANT`, and `REVOKE` and have made so many modifications after running `mysql_install_db` that you want to wipe out the tables and start over.

  To re-create the grant tables, stop the server if it is running and remove the `mysql` database directory. Then run `mysql_install_db` again.

2.18.2 Starting the Server

This section describes how start the server on Unix and Unix-like systems. (For Windows, see Section 2.10.4.4, “Starting the Server for the First Time”.) For some suggested commands that you can use to test whether the server is accessible and working properly, see Section 2.18.3, “Testing the Server”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Start the MySQL server like this:

```
shell> bin/mysqld_safe --user=mysql &
```

It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this if you run `mysqld_safe` as `root`, include the `--user` option as shown. Otherwise, execute the program while logged in as `mysql`, in which case you can omit the `--user` option from the command.

For further instructions for running MySQL as an unprivileged user, see Section 6.1.5, “How to Run MySQL as a Normal User”.

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

If the server is unable to access the data directory it starts or read the grant tables in the `mysql` database, it writes a message to its error log. Such problems can occur if you neglected to create the grant tables by initializing the data directory before proceeding to this step, or if you ran the command that initializes the data directory without the `--user` option. Remove the `data` directory and run the command with the `--user` option.

If you have other problems starting the server, see Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”. For more information about `mysqld_safe`, see Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD::mysql` Perl modules. See Section 4.6.15, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”. For Perl module installation instructions, see Section 2.22, “Perl Installation Notes”.

If you would like to use `mysqlaccess` and have the MySQL distribution in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `bin/mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

2.18.2.1 Troubleshooting Problems Starting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server. For additional suggestions for Windows systems, see Section 2.10.5, “Troubleshooting a MySQL Installation Under Windows”.

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start.
- Specify any special options needed by the storage engines you are using.
- Make sure that the server knows where to find the data directory.
- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must be set such that the server can read and modify them.
- Verify that the network interfaces the server wants to use are available.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Some storage engines have options that control their behavior. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `BDB`, `NDB`), be sure that you have them configured the way you want before starting the server:

- If you are using `InnoDB` tables, see Section 14.2.1, “Configuring InnoDB”.
- If you are using `BDB` (Berkeley DB) tables, see Section 14.5.3, “BDB Startup Options”.
- If you are using MySQL Cluster, see Section 17.3, “MySQL Cluster Configuration”.

Storage engines will use default option values if you specify none, but it is recommended that you review the available options and specify explicit values for those for which the defaults are not appropriate for your installation.

When the `mysqld` server starts, it changes location to the data directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

The data directory location is hardwired in when the server is compiled. This is where the server looks for the data directory by default. If the data directory is located somewhere else on your system, the server will not work properly. You can determine what the default path settings are by invoking `mysqld` with the `--verbose` and `--help` options.

If the default locations do not match the MySQL installation layout on your system, you can override them by specifying options to `mysqld` or `mysqld_safe` on the command line or in an option file.

To specify the location of the data directory explicitly, use the `--datadir` option. However, normally you can tell `mysqld` the location of the base directory under which MySQL is installed and it looks for the data directory there. You can do this with the `--basedir` option.

To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` and `--help` options. For example, if you change location into the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` as well, but `--verbose` and `--help` must be the last options.

Once you determine the path settings you want, start the server without `--verbose` and `--help`.

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

`host_name` is the name of the MySQL server host.

If you get `Errcode 13` (which means `Permission denied`) when starting `mysqld`, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

```
shell> ls -la /usr/local/mysql/var
```

If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

Even with correct ownership, MySQL might fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, reconfigure that software to enable `mysqld` to access the directories it uses during normal operation.

If the server fails to start up correctly, check the error log. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 5.0\data` on Windows, `/usr/local/mysql/data` for a Unix/Linux binary distribution, and `/usr/local/var` for a Unix/Linux source distribution). Look in the data directory for files with names of the form *`host_name`*`.err` and *`host_name`*`.log`, where *`host_name`* is the name of your server host. Then examine the last few lines of these files. You can use `tail` to display them:

```
shell> tail host_name.err
shell> tail host_name.log
```

The error log should contain information that indicates why the server could not start. For example, you might see something like this in the log:

```
000729 14:50:10  bdb:  Recovery function for LSN 1 27595 failed
000729 14:50:10  bdb:  warning: ./test/t1.db: No such file or directory
000729 14:50:10  Can't init databases
```

This means that you did not start `mysqld` with the `--bdb-no-recover` option and Berkeley DB found something wrong with its own log files when it tried to recover your databases. To be able to continue, you should move the old Berkeley DB log files from the database directory to some other place, where you can later examine them. The `BDB` log files are named in sequence beginning with `log.0000000001`, where the number increases over time.

If you are running `mysqld` with `BDB` table support and `mysqld` dumps core at startup, this could be due to problems with the `BDB` recovery log. In this case, you can try starting `mysqld` with `--bdb-no-recover`. If that helps, you should remove all `BDB` log files from the data directory and try starting `mysqld` again without the `--bdb-no-recover` option.

If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in Section 5.5, “Running Multiple MySQL Instances on One Machine”.)

If no other server is running, try to execute the command `telnet` *`your_host_name tcp_ip_port_number`*. (The default MySQL port number is 3306.) Then press Enter a couple of

This documentation is for an older version. If you're
This documentation is for an older version. If you're

times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. You will need to track down what program this is and disable it, or else tell `mysqld` to listen to a different port with the `--port` option. In this case, you will also need to specify the port number for client programs when connecting to the server using TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, you should make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1       localhost
```

If you cannot get `mysqld` to start, you can try to make a trace file to find the problem by using the `--debug` option. See Section 21.3.3, “The DBUG Package”.

2.18.3 Testing the Server

After the data directory is initialized and you have started the server, perform some simple tests to make sure that it works satisfactorily. This section assumes that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your shell (command interpreter) to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Section 4.2.10, “Setting Environment Variables”.

Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

If you cannot connect to the server, specify a `-u root` option to connect as `root`. If you have assigned a password for the `root` account already, you'll also need to specify `-p` on the command line and enter the password when prompted. For example:

```
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin  Ver 14.12 Distrib 5.0.96, for pc-linux-gnu on i686
...

Server version          5.0.96
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To see what else you can do with `mysqladmin`, invoke it with the `--help` option.

Verify that you can shut down the server (include a `-p` option if the `root` account has a password already):

```
shell> bin/mysqladmin -u root shutdown
```

Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql &
```

If `mysqld_safe` fails, see Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”.

Run some simple tests to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
shell> bin/mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

The list of installed databases may vary, but will always include the minimum of `mysql` and `information_schema`.

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
shell> bin/mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| proc                      |
| procs_priv                |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
shell> bin/mysql -e "SELECT User, Host FROM mysql.user" mysql
+------+-----------+
| User | Host      |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+-----------+
| root | localhost |
+------+-----------+
```

There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It requires the Perl DBI module that provides a database-independent interface to the various databases, and some other additional Perl modules:

```
DBI
DBD::mysql
Data::Dumper
Data::ShowTable
```

These modules can be obtained from CPAN (http://www.cpan.org/). See also Section 2.22.1, “Installing Perl on Unix”.

The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```
shell> cd sql-bench
shell> perl run-all-tests
```

If you do not have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the `sql-bench` benchmark directory.) In this case, you must first install the benchmark suite before you can use it. There are separate benchmark RPM files named `mysql-bench-VERSION.i386.rpm` that contain benchmark code and data.

If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

```
shell> mysql -vvf test < ./tests/auto_increment.tst
```

The expected result of the test can be found in the `./tests/auto_increment.res` file.

At this point, your server is running and you can access it. To tighten security if you have not yet assigned passwords to the initial account or accounts, follow the instructions in Section 2.18.4, “Securing the Initial MySQL Accounts”.

For more information about `mysql`, `mysqladmin`, and `mysqlshow`, see Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

2.18.4 Securing the Initial MySQL Accounts

The MySQL installation process involves initializing the data directory, including the `mysql` database containing the grant tables that define MySQL accounts. For details, see Section 2.18, “Postinstallation Setup and Testing”.

This section describes how to assign passwords to the initial accounts created during the MySQL installation procedure, if you have not already done so.

The `mysql.user` grant table defines the initial MySQL user accounts and their access privileges:

- Some accounts have the user name `root`. These are superuser accounts that have all privileges and can do anything. If these `root` accounts have empty passwords, anyone can connect to the MySQL server as `root` *without a password* and be granted all privileges.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- On Windows, `root` accounts are created that permit connections from the local host only. Connections can be made by specifying the host name `localhost` or the IP address `127.0.0.1`. If the user selects the **Enable root access from remote machines** option during installation, the Windows installer creates another `root` account that permits connections from any host.
- On Unix, each `root` account permits connections from the local host. Connections can be made by specifying the host name `localhost`, the IP address `127.0.0.1`, or the actual host name or IP address.

An attempt to connect to the host `127.0.0.1` normally resolves to the `localhost` account. However, this fails if the server is run with the `--skip-name-resolve` option, so the `127.0.0.1` account is useful in that case.

- If accounts for anonymous users were created, these have an empty user name. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.
  - On Windows, there is one anonymous account that permits connections from the local host. Connections can be made by specifying a host name of `localhost`. It has no global privileges. (Before MySQL 5.0.36, it has all global privileges, just like the `root` accounts.)
  - On Unix, each anonymous account permits connections from the local host. Connections can be made by specifying a host name of `localhost` for one of the accounts, or the actual host name or IP address for the other.

To display which accounts exist in the `mysql.user` table and check whether their passwords are empty, use the following statement:

```
mysql> SELECT User, Host, Password FROM mysql.user;
+------+--------------------+----------+
| User | Host               | Password |
+------+--------------------+----------+
| root | localhost          |          |
| root | myhost.example.com |          |
| root | 127.0.0.1          |          |
|      | localhost          |          |
|      | myhost.example.com |          |
+------+--------------------+----------+
```

This output indicates that there are several `root` and anonymous-user accounts, none of which have passwords. The output might differ on your system, but the presence of accounts with empty passwords means that your MySQL installation is unprotected until you do something about it:

- Assign a password to each MySQL `root` account that does not have one.
- To prevent clients from connecting as anonymous users without a password, either assign a password to each anonymous account or remove the accounts.

In addition, the `mysql.db` table contains rows that permit all accounts to access the `test` database and other databases with names that start with `test_`. This is true even for accounts that otherwise have no special privileges such as the default anonymous accounts. This is convenient for testing but inadvisable on production servers. Administrators who want database access restricted only to accounts that have permissions granted explicitly for that purpose should remove these `mysql.db` table rows.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the `root` accounts, then for the anonymous accounts. The instructions also cover how to remove anonymous accounts, should you prefer not to permit anonymous access at all, and describe how to remove permissive access to test databases. Replace *`new_password`* in the examples with the password that you

This documentation is for an older version. If you're
This documentation is for an older version. If you're

want to use. Replace *`host_name`* with the name of the server host. You can determine this name from the output of the preceding `SELECT` statement. For the output shown, *`host_name`* is `myhost.example.com`.

Note

For additional information about setting passwords, see Section 6.3.5, “Assigning Account Passwords”. If you forget your `root` password after setting it, see Section B.5.3.2, “How to Reset the Root Password”.

To set up additional accounts, see Section 6.3.2, “Adding User Accounts”.

You might want to defer setting the passwords until later, to avoid the need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for production purposes.

Assigning root Account Passwords

The `root` account passwords can be set several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the `UPDATE` statement
- Use the `mysqladmin` command-line client program

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue a `SET PASSWORD` statement for each `root` account listed in the `mysql.user` table.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('new_password');
```

The last statement is unnecessary if the `mysql.user` table has no `root` account with a host value of `%`.

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('new_password');
```

You can also use a single statement that assigns a password to all `root` accounts by using `UPDATE` to modify the `mysql.user` table directly. This method works on any platform:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('new_password')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

To assign passwords to the `root` accounts using `mysqladmin`, execute the following commands:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqladmin -u root password "new_password"
shell> mysqladmin -u root -h host_name password "new_password"
```

Those commands apply both to Windows and to Unix. The double quotation marks around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

The `mysqladmin` method of setting the `root` account passwords does not work for the `'root'@'127.0.0.1'` account. Use the `SET PASSWORD` method shown earlier.

After the `root` passwords have been set, you must supply the appropriate password whenever you connect as `root` to the server. For example, to shut down the server with `mysqladmin`, use this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

The `mysql` commands in the following instructions include a `-p` option based on the assumption that you have assigned the `root` account passwords using the preceding instructions and must specify that password when connecting to the server.

Assigning Anonymous Account Passwords

To assign passwords to the anonymous accounts, connect to the server as `root`, then use either `SET PASSWORD` or `UPDATE`.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('new_password');
```

To use `SET PASSWORD` on Unix, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('new_password');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('new_password');
```

To set the anonymous-user account passwords with a single `UPDATE` statement, do this (on any platform):

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> UPDATE mysql.user SET Password = PASSWORD('new_password')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

Removing Anonymous Accounts

If you prefer to remove any anonymous accounts rather than assigning them passwords, do so as follows on Windows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> DROP USER ''@'localhost';
```

On Unix, remove the anonymous accounts like this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER ''@'localhost';
mysql> DROP USER ''@'host_name';
```

Securing Test Databases

By default, the `mysql.db` table contains rows that permit access by any user to the `test` database and other databases with names that start with `test_`. (These rows have an empty `User` column value, which for access-checking purposes matches any user name.) This means that such databases can be used even by accounts that otherwise possess no privileges. If you want to remove any-user access to test databases, do so as follows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the privilege change remains unnoticed by the server until you restart it.

With the preceding change, only users who have global database privileges or privileges granted explicitly for the `test` database can use it. However, if you prefer that the database not exist at all, drop it:

```
mysql> DROP DATABASE test;
```

Note

On Windows, you can also perform the process described in this section using the Configuration Wizard (see Section 2.10.3.11, “The Security Options Dialog”). On other platforms, the MySQL distribution includes `mysql_secure_installation`, a command-line utility that automates much of the process of securing a MySQL installation.

2.18.5 Starting and Stopping MySQL Automatically

This section discusses methods for starting and stopping the MySQL server.

Generally, you start the `mysqld` server in one of these ways:

- Invoke `mysqld` directly. This works on any platform.
- On Windows, you can set up a MySQL service that runs automatically when Windows starts. See Section 2.10.4.7, “Starting MySQL as a Windows Service”.
- On Unix and Unix-like systems, you can invoke `mysqld_safe`, which tries to determine the proper options for `mysqld` and then runs it with those options. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.
- On systems that use System V-style run directories (that is, `/etc/init.d` and run-level specific directories), invoke `mysql.server`. This script is used primarily at system startup and shutdown. It usually is installed under the name `mysql`. The `mysql.server` script starts the server by invoking `mysqld_safe`. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- On OS X, install a separate MySQL Startup Item package to enable automatic MySQL startup at system startup. The Startup Item starts the server by invoking `mysql.server`. For details, see Section 2.11, “Installing MySQL on OS X”.

The `mysqld_safe` and `mysql.server` scripts and the OS X Startup Item can be used to start the server manually, or automatically at system startup time. `mysql.server` and the Startup Item also can be used to stop the server.

The following table shows which option groups the server and startup scripts read from option files.

Table 2.8 MySQL Startup Scripts and Supported Server Option Groups

Script	Option Groups
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-major_version]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`
`mysql.server`	`[mysqld]`, `[mysql.server]`, `[server]`

`[mysqld-major_version]` means that groups with names like `[mysqld-4.1]` and `[mysqld-5.0]` are read by servers having versions 4.1.x, 5.0.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead.

For more information on MySQL configuration files and their structure and contents, see Section 4.2.6, “Using Option Files”.

2.19 Upgrading or Downgrading MySQL

This section describes the steps to upgrade or downgrade a MySQL installation.

Upgrading is a common procedure, as you pick up bug fixes within the same MySQL release series or significant features between major MySQL releases. You perform this procedure first on some test systems to make sure everything works smoothly, and then on the production systems.

Downgrading is less common. Typically, you undo an upgrade because of some compatibility or performance issue that occurs on a production system, and was not uncovered during initial upgrade verification on the test systems. As with the upgrade procedure, perform and verify the downgrade procedure on some test systems first, before using it on a production system.

2.19.1 Upgrading MySQL

This section describes how to upgrade to a new MySQL version.

- Supported Upgrade Methods
- Supported Upgrade Paths
- Before You Begin
- Performing an In-place Upgrade
- Performing a Logical Upgrade
- Upgrade Troubleshooting

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Supported Upgrade Methods

- *In-place Upgrade*: Involves shutting down the old MySQL version, replacing the old MySQL binaries or packages with the new ones, restarting MySQL on the existing data directory, and running `mysql_upgrade`.
- *Logical Upgrade*: Involves exporting existing data from the old MySQL version using `mysqldump`, installing the new MySQL version, loading the dump file into the new MySQL version, and running `mysql_upgrade`.

Note

MySQL recommends a `mysqldump` upgrade when upgrading from a previous release. For example, use this method when upgrading from 4.1 to 5.0.

For in-place and logical upgrade procedures, see Performing an In-place Upgrade, and Performing a Logical Upgrade.

If you run MySQL Server on Windows, see Section 2.10.7, “Upgrading MySQL on Windows”.

Supported Upgrade Paths

Unless otherwise documented, the following upgrade paths are supported:

- Upgrading from a release series version to a newer release series version is supported. For example, upgrading from 5.0.95 to 5.0.96 is supported. Skipping release series versions is also supported. For example, upgrading from 5.0.92 to 5.0.96 is supported.
- Upgrading one release level is supported. For example, upgrading from 4.1 to 5.0 is supported. Upgrading to the latest release series version is recommended before upgrading to the next release level. For example, upgrade to the latest 4.1 release before upgrading to 5.0.
- Upgrading more than one release level is supported, but only if you upgrade one release level at a time. For example, if you currently are running MySQL 4.0 and wish to upgrade to a newer series, upgrade to MySQL 4.1 first before upgrading to MySQL 5.0, and so forth. For information on upgrading to MySQL 4.1 or earlier releases, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.
- Direct upgrades that skip a release level (for example, upgrading directly from MySQL 4.0 to 5.0) are not recommended or supported.

The following conditions apply to all upgrade paths:

- Upgrades between General Availability (GA) status releases are supported.
- Upgrades between milestone releases (or from a milestone release to a GA release) are not supported.
- For upgrades between versions of a MySQL release series that has reached GA status, you can move the MySQL format files and data files between different versions on systems with the same architecture. This is not necessarily true for upgrades between milestone releases. Use of milestone releases is at your own risk.

Before You Begin

Before upgrading, review the following information and perform the recommended steps:

- Before upgrading, protect your data by creating a backup of your current databases and log files. The backup should include the `mysql` database, which contains the MySQL system tables. See Section 7.2, “Database Backup Methods”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Review the Release Notes which provide information about features that are new in the MySQL 5.0 or differ from those found in earlier MySQL releases. Some of these changes may result in incompatibilities.
- Review Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”. This section describes changes that may require action before or after upgrading.
- Check Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”, to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are upgrading. If such changes have resulted in an incompatibility between MySQL versions, you will need to upgrade the affected tables using the instructions in Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”.
- If you use replication, see Section 16.4.3, “Upgrading a Replication Setup”, for information on upgrading your replication setup.
- If you use XA transactions with `InnoDB`, run `XA RECOVER` before upgrading to check for uncommitted XA transactions. If results are returned, either commit or rollback the XA transactions by issuing an `XA COMMIT` or `XA ROLLBACK` statement.
- If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.
- Rebuilding and reinstalling the Perl `DBD::mysql` module whenever you install or upgrade to a new release of MySQL is recommended. The same applies to other MySQL interfaces as well, such as PHP `mysql` extensions and the Python `MySQLdb` module.

Performing an In-place Upgrade

This section describes how to perform an in-place upgrade. Review Before you Begin before proceeding.

Note

If you upgrade an installation originally produced by installing multiple RPM packages, upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

To perform an in-place upgrade:

1. Review the changes described in Section 2.19.1.1, “Changes Affecting Upgrades to 5.0” for steps to be performed before upgrading.
2. If you use `InnoDB`, configure MySQL to perform a slow shutdown. For example:

   ```
   shell> bin/mysql -u root -ppassword --execute="set global innodb_fast_shutdown=0"
   ```

   With a slow shutdown, `InnoDB` performs a full purge and change buffer merge before shutting down, which ensures that data files are fully prepared in case of file format differences between releases.
3. Stop the old MySQL server.
4. Upgrade the MySQL binaries or packages in place (replace the 4.1 binaries with those from 5.0).
5. Start the MySQL 5.0 server using the existing data directory.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6. Run `mysql_upgrade`. For example:

```
shell> bin/mysql_upgrade -u root -ppassword
```

`mysql_upgrade` examines all tables in all databases for incompatibilities with the current version of MySQL Server. `mysql_upgrade` also upgrades the system tables so that you can take advantage of new privileges or capabilities.

Note

`mysql_upgrade` does not upgrade the contents of the help tables. For upgrade instructions, see Section 5.1.8, “Server-Side Help”.

Performing a Logical Upgrade

This section describes how to perform a logical upgrade. Review Before you Begin before proceeding.

To perform a logical upgrade:

1. Review the changes described in Section 2.19.1.1, “Changes Affecting Upgrades to 5.0” for steps to be performed before upgrading.
2. Export your existing data from the previous MySQL version:

```
shell> mysqldump --add-drop-table --add-drop-table
    -> --all-databases --force > data-for-upgrade.sql
```

Note

The `--all-databases` option includes all databases in the dump, including the `mysql` database that holds the system tables.

3. Stop the old MySQL server.
4. Install MySQL 5.0.
5. Initialize a new data directory:

```
shell> scripts/mysql_install_db --user=mysql --datadir=/path/to/5.0-datadir
```

6. Start the MySQL 5.0 server using the new data directory.
7. Load the previously created dump file into the new MySQL server. For example:

```
shell> bin/mysql -u root -ppassword --execute="source data-for-upgrade.sql" --force
```

8. Run `mysql_upgrade`. For example:

```
shell> bin/mysql_upgrade -u root -ppassword
```

`mysql_upgrade` examines all tables in all databases for incompatibilities with the current version of MySQL Server. `mysql_upgrade` also upgrades the system tables so that you can take advantage of new privileges or capabilities.

Note

`mysql_upgrade` does not upgrade the contents of the help tables. For upgrade instructions, see Section 5.1.8, “Server-Side Help”.

This documentation is for an older version. If you're

9. If you use `InnoDB`, configure MySQL to perform a slow shutdown. For example:

```
shell> bin/mysql -u root -ppassword --execute="set global innodb_fast_shutdown=0"
```

10. Shut down and restart the MySQL server to ensure a clean shutdown and startup.

Upgrade Troubleshooting

- If problems occur, such as that the new `mysqld` server does not start or that you cannot connect without a password, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.
- If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example from `libmysqlclient.so.15` to `libmysqlclient.so.16`.
- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See Section 9.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

2.19.1.1 Changes Affecting Upgrades to 5.0

Before upgrading to MySQL 5.0, review the changes described in this section to identify upgrade issues that apply to your current MySQL installation and applications.

Changes marked as either **Known issue** or **Incompatible change** are incompatibilities with earlier versions of MySQL, and may require your attention *before you upgrade*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If any upgrade issue applicable to your installation involves an incompatibility that requires special handling, follow the instructions given in the incompatibility description. Sometimes this involves dumping and reloading tables, or use of a statement such as `CHECK TABLE` or `REPAIR TABLE`.

For dump and reload instructions, see Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”. Any procedure that involves `REPAIR TABLE` with the `USE_FRM` option *must* be done before upgrading. Use of this statement with a version of MySQL different from the one used to create the table (that is, using it after upgrading) may damage the table. See Section 13.7.2.6, “REPAIR TABLE Syntax”.

Note

Several visible behaviors have changed between MySQL 4.1 and MySQL 5.0 to make MySQL more compatible with standard SQL. These changes may affect your applications.

- System Table Changes
- Server Changes

This documentation is for an older version. If you're

- SQL Changes
- C API Changes

System Table Changes

- **After upgrading a 5.0 installation to 5.0.10 or higher**, it is *necessary* to upgrade your grant tables. Otherwise, creating stored procedures and functions might not work. The procedure for doing this is described in Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

Server Changes

- MySQL 5.0.27 is the last version in MySQL 5.0 for which MySQL-Max binary distributions are provided, except for RPM distributions. For RPMs, MySQL 5.0.37 is the last release. After these versions, the features previously included in the `mysqld-max` server are included in `mysqld`.

  If you previously installed a MySQL-Max distribution that includes a server named `mysqld-max`, and then upgrade later to a non-Max version of MySQL, `mysqld_safe` still attempts to run the old `mysqld-max` server. If you perform such an upgrade, you should remove the old `mysqld-max` server manually to ensure that `mysqld_safe` runs the new `mysqld` server.

- **Incompatible change**: Character set or collation changes may require table indexes to be rebuilt. In MySQL 5.0, these occurred in version 5.0.48. For details, see Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.

- **Incompatible change**: `SHOW CREATE VIEW` displays view definitions using an `AS alias_name` clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. As of MySQL 5.0.52, aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of `SHOW CREATE VIEW` fail if any column alias exceeds 64 characters. This can cause problems for replication or loading dump files. For additional information and workarounds, see Section C.4, “Restrictions on Views”.

- **Incompatible change**: Beginning with MySQL 5.0.42, when a `DATE` value is compared with a `DATETIME` value, the `DATE` value is coerced to the `DATETIME` type by adding the time portion as `00:00:00`. Previously, the time portion of the `DATETIME` value was ignored, or the comparison could be performed as a string comparison. To mimic the old behavior, use the `CAST()` function to cause the comparison operands to be treated as previously. For example:

  ```
  date_col = CAST(NOW() AS DATE)
  ```

- **Incompatible change**: For `ENUM` columns that had enumeration values containing commas, the commas were mapped to `0xff` internally. However, this rendered the commas indistinguishable from true `0xff` characters in the values. This no longer occurs. However, the fix requires that you dump and reload any tables that have `ENUM` columns containing true `0xff` in their values: Dump the tables using `mysqldump` with the current server before upgrading from a version of MySQL 5.0 older than 5.0.36 to version 5.0.36 or newer.

- **Incompatible change**. For `BINARY` columns, the pad value and how it is handled has changed as of MySQL 5.0.15. The pad value for inserts now is `0x00` rather than space, and there is no stripping of the pad value for retrievals. For details, see Section 11.4.2, “The BINARY and VARBINARY Types”.

- **Incompatible change**: As of MySQL 5.0.13, `InnoDB` rolls back only the last statement on a transaction timeout. As of MySQL 5.0.32, a new option, `--innodb_rollback_on_timeout`, causes `InnoDB` to abort and roll back the entire transaction if a transaction timeout occurs (the same behavior as in MySQL 4.1).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- **Incompatible change**: The namespace for triggers changed in MySQL 5.0.10. Previously, trigger names had to be unique per table. Now they must be unique within the schema (database). An implication of this change is that `DROP TRIGGER` syntax now uses a schema name instead of a table name (schema name is optional and, if omitted, the current schema will be used).

  When upgrading from a version of MySQL 5 older than 5.0.10 to MySQL 5.0.10 or newer, you must drop all triggers and re-create them or `DROP TRIGGER` will not work after the upgrade. Here is a suggested procedure for doing this:

  1. Upgrade to MySQL 5.0.10 or later to be able to access trigger information in the `INFORMATION_SCHEMA.TRIGGERS` table. (This should work even for pre-5.0.10 triggers.)

  2. Dump all trigger definitions using the following `SELECT` statement:

     ```
     SELECT CONCAT('CREATE TRIGGER ', t.TRIGGER_SCHEMA, '.', t.TRIGGER_NAME,
                   ' ', t.ACTION_TIMING, ' ', t.EVENT_MANIPULATION, ' ON ',
                   t.EVENT_OBJECT_SCHEMA, '.', t.EVENT_OBJECT_TABLE,
                   ' FOR EACH ROW ', t.ACTION_STATEMENT, '//' )
     INTO OUTFILE '/tmp/triggers.sql'
     FROM INFORMATION_SCHEMA.TRIGGERS AS t;
     ```

     The statement uses `INTO OUTFILE`, so you must have the `FILE` privilege. The file will be created on the server host. Use a different file name if you like. To be 100% safe, inspect the trigger definitions in the `triggers.sql` file, and perhaps make a backup of the file.

  3. Stop the server and drop all triggers by removing all `.TRG` files in your database directories. Change location to your data directory and issue this command:

     ```
     shell> rm */*.TRG
     ```

  4. Start the server and re-create all triggers using the `triggers.sql` file:

     ```
     mysql> delimiter // ;
     mysql> source /tmp/triggers.sql //
     ```

  5. Use the `SHOW TRIGGERS` statement to check that all triggers were created successfully.

- **Incompatible change**: The indexing order for end-space in `TEXT` columns for `InnoDB` and `MyISAM` tables has changed. Starting from 5.0.3, `TEXT` indexes are compared as space-padded at the end (just as MySQL sorts `CHAR`, `VARCHAR` and `TEXT` fields). If you have an index on a `TEXT` column, you should run `CHECK TABLE` on it. If the check reports errors, rebuild the indexes: Dump and reload the table if it is an `InnoDB` table, or run `OPTIMIZE TABLE` or `REPAIR TABLE` if it is a `MyISAM` table.

- **Incompatible change**. As of MySQL 5.0.3, trailing spaces no longer are removed from values stored in `VARCHAR` and `VARBINARY` columns. The maximum lengths for `VARCHAR` and `VARBINARY` columns in MySQL 5.0.3 and later are 65,535 characters and 65,535 bytes, respectively.

  When a binary upgrade (file system-level copy of data files) to MySQL 5.0 is performed for a table with a `VARBINARY` column, the column is space-padded to the full permissible width of the column. This causes values in `VARBINARY` columns that do not occupy the full width of the column to include extra trailing spaces after the upgrade, which means that the data in the column is different.

  In addition, new rows inserted into a table upgraded in this way will be space padded to the full width of the column.

  This issue can be resolved as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

1. For each table containing VARBINARY columns, execute the following statement, where *tbl_name* is the name of the table and *engine_name* is the name of the storage engine currently used by *tbl_name*:

   ```
   ALTER TABLE tbl_name ENGINE=engine_name;
   ```

   In other words, if the table named mytable uses the MyISAM storage engine, then you would use this statement:

   ```
   ALTER TABLE mytable ENGINE=MYISAM;
   ```

   This rebuilds the table so that it uses the 5.0 VARBINARY format.

2. Then you must remove all trailing spaces from any VARBINARY column values. For each VARBINARY column *varbinary_column*, execute the following statement, where *tbl_name* is the name of the table containing the VARBINARY column:

   ```
   UPDATE tbl_name SET varbinary_column = RTRIM(varbinary_column);
   ```

   This is necessary and safe because trailing spaces are stripped before 5.0.3, meaning that any trailing spaces are erroneous.

This problem does not occur (and thus these two steps are not required) for tables upgraded using the recommended procedure of dumping tables prior to the upgrade and reloading them afterward.

Note

If you create a table with new VARCHAR or VARBINARY columns in MySQL 5.0.3 or later, the table will not be usable if you downgrade to a version older than 5.0.3. Dump the table with mysqldump before downgrading and reload it after downgrading.

- **Incompatible change**: The implementation of DECIMAL was changed in MySQL 5.0.3. You should make your applications aware of this change. For information about this change, and about possible incompatibilities with old applications, see Section 12.17, “Precision Math”, in particular, Section 12.17.2, “DECIMAL Data Type Characteristics”.

  DECIMAL columns are stored in a more efficient format. To convert a table to use the new DECIMAL type, you should do an ALTER TABLE on it. (The ALTER TABLE also will change the table's VARCHAR columns to use the new VARCHAR data type properties, described in a separate item.)

  A consequence of the change in handling of the DECIMAL and NUMERIC fixed-point data types is that the server is more strict to follow standard SQL. For example, a data type of DECIMAL(3,1) stores a maximum value of 99.9. Before MySQL 5.0.3, the server permitted larger numbers to be stored. That is, it stored a value such as 100.0 as 100.0. As of MySQL 5.0.3, the server clips 100.0 to the maximum permissible value of 99.9. If you have tables that were created before MySQL 5.0.3 and that contain floating-point data not strictly legal for the data type, you should alter the data types of those columns. For example:

  ```
  ALTER TABLE tbl_name MODIFY col_name DECIMAL(4,1);
  ```

  The behavior used by the server for DECIMAL columns in a table depends on the version of MySQL used to create the table. If your server is from MySQL 5.0.3 or higher, but you have DECIMAL columns

This documentation is for an older version. If you're

in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer `DECIMAL` format, dump them with `mysqldump` and reload them.

- **Incompatible change**: MySQL 5.0.3 and up uses precision math when calculating with `DECIMAL` and integer columns (64 decimal digits) and for rounding exact-value numbers. Rounding behavior is well-defined, not dependent on the implementation of the underlying C library. However, this might result in incompatibilities for applications that rely on the old behavior. (For example, inserting .5 into an `INT` column results in 1 as of MySQL 5.0.3, but might be 0 in older versions.) For more information about rounding behavior, see Section 12.17.4, “Rounding Behavior”, and Section 12.17.5, “Precision Math Examples”.

- **Incompatible change**: In very old versions of MySQL (prior to 4.1), the `TIMESTAMP` data type supported a display width, which was silenty ignored beginning with MySQL 4.1. This is deprecated in MySQL 5.1, and removed altogether in MySQL 5.5. These changes in behavior can lead to two problem scenarios when trying to use `TIMESTAMP(N)` columns with a MySQL 5.5 or later server:

  - When importing a dump file (for example, one created using `mysqldump`) created in a MySQL 5.0 or earlier server into a server from a newer release series, a `CREATE TABLE` or `ALTER TABLE` statement containing `TIMESTAMP(N)` causes the import to fail with a syntax error.

    To fix this problem, edit the dump file in a text editor to replace any instances of `TIMESTAMP(N)` with `TIMESTAMP` prior to importing the file. Be sure to use a plain text editor for this, and not a word processor; otherwise, the result is almost certain to be unusable for importing into the MySQL server.

  - When trying replicate any `CREATE TABLE` or `ALTER TABLE` statement containing `TIMESTAMP(N)` from a master MySQL server that supports the `TIMESTAMP(N)` syntax to a MySQL 5.5 or newer slave, the statement causes replication to fail. Similarly, when you try to restore from a binary log written by a server that supports `TIMESTAMP(N)` to a MySQL 5.5 or newer server, any `CREATE TABLE` or `ALTER TABLE` statement containing `TIMESTAMP(N)` causes the backup to fail. This holds true regardless of the logging format used by a MySQL 5.1 or newer server.

    It may be possible to fix such issues using a hex editor, by replacing any width arguments used with `TIMESTAMP`, and the parentheses containing them, with space characters (hexadecimal `20`). This can be made to work as long as checksums were not enabled when creating the binary log. Be sure to use a programmer's binary hex editor and not a regular text editor or word processor for this; otherwise, the result is almost certain to be a corrupted binary log file. To guard against accidental corruption of the binary log, you should always work on a copy of the file rather than the original.

  You should try to handle potential issues of these types proactively by updating with `ALTER TABLE` any `TIMESTAMP(N)` columns in your databases so that they use `TIMESTAMP` instead, before performing any upgrades.

- **Incompatible change**: `MyISAM` and `InnoDB` tables created with `DECIMAL` columns in MySQL 5.0.3 to 5.0.5 will appear corrupt after an upgrade to MySQL 5.0.6. (The same incompatibility will occur for these tables created in MySQL 5.0.6 after a downgrade to MySQL 5.0.3 to 5.0.5.) If you have such tables, check and repair them with `mysql_upgrade` after upgrading. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

- **Incompatible change**: For user-defined functions, exact-value decimal arguments such as `1.3` or `DECIMAL` column values were passed as `REAL_RESULT` values prior to MySQL 5.0.3. As of 5.0.3, they are passed as strings with a type of `DECIMAL_RESULT`. If you upgrade to 5.0.3 and find that your UDF now receives string values, use the initialization function to coerce the arguments to numbers as described in Section 21.2.2.3, “UDF Argument Processing”.

- **Incompatible change**: As of MySQL 5.0.3, the server by default no longer loads user-defined functions (UDFs) unless they have at least one auxiliary symbol (for example, an `xxx_init` or `xxx_deinit`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

symbol) defined in addition to the main function symbol. This behavior can be overridden with the `--allow-suspicious-udfs` option. See Section 21.2.2.6, “UDF Security Precautions”.

- **Incompatible change**: The update log has been removed in MySQL 5.0. If you had enabled it previously, enable the binary log instead.

- **Incompatible change**: Support for the `ISAM` storage engine has been removed in MySQL 5.0. If you have any `ISAM` tables, you should convert them *before* upgrading. For example, to convert an `ISAM` table to use the `MyISAM` storage engine, use this statement:

```
ALTER TABLE tbl_name ENGINE = MyISAM;
```

  Use a similar statement for every `ISAM` table in each of your databases.

- **Incompatible change**: Support for `RAID` options in `MyISAM` tables has been removed in MySQL 5.0. If you have tables that use these options, you should convert them before upgrading. One way to do this is to dump them with `mysqldump`, edit the dump file to remove the `RAID` options in the `CREATE TABLE` statements, and reload the dump file. Another possibility is to use `CREATE TABLE new_tbl ... SELECT raid_tbl` to create a new table from the `RAID` table. However, the `CREATE TABLE` part of the statement must contain sufficient information to re-create column attributes as well as indexes, or column attributes may be lost and indexes will not appear in the new table. See Section 13.1.10, “CREATE TABLE Syntax”.

  The `.MYD` files for `RAID` tables in a given database are stored under the database directory in subdirectories that have names consisting of two hex digits in the range from `00` to `ff`. After converting all tables that use `RAID` options, these `RAID`-related subdirectories still will exist but can be removed. Verify that they are empty, and then remove them manually. (If they are not empty, this indicates that there is some `RAID` table that has not been converted.)

- As of MySQL 5.0.25, the `lc_time_names` system variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. See Section 10.7, “MySQL Server Locale Support”.

- In MySQL 5.0.6, binary logging of stored routines and triggers was changed. This change has implications for security, replication, and data recovery, as discussed in Section 18.6, “Binary Logging of Stored Programs”.

- As of MySQL 5.0.28, `mysqld_safe` no longer implicitly invokes `mysqld-max` if it exists. Instead, it invokes `mysqld` unless a `--mysqld` or `--mysqld-version` option is given to specify another server explicitly. If you previously relied on the implicit invocation of `mysqld-max`, you should use an appropriate option now.

SQL Changes

- **Known issue**: Prior to MySQL 5.0.46, the parser accepted invalid code in SQL condition handlers, leading to server crashes or unexpected execution behavior in stored programs. Specifically, the parser permitted a condition handler to refer to labels for blocks that enclose the handler declaration. This was incorrect because block label scope does not include the code for handlers declared within the labeled block.

  As of 5.0.46, the parser rejects this invalid construct, but if you upgrade in place (without dumping and reloading your databases), existing handlers that contain the construct still are invalid *even if they appear to function as you expect* and should be rewritten.

This documentation is for an older version. If you're

To find affected handlers, use mysqldump to dump all stored procedures and functions, triggers, and events. Then attempt to reload them into an upgraded server. Handlers that contain illegal label references will be rejected.

For more information about condition handlers and writing them to avoid invalid jumps, see Section 13.6.7.2, “DECLARE ... HANDLER Syntax”.

- **Known issue**: The fix for Bug #23491 introduced a problem with `SHOW CREATE VIEW`, which is used by `mysqldump`. This causes an incompatibility when upgrading from versions affected by that bug fix (MySQL 5.0.40 through 5.0.43, MySQL 5.1.18 through 5.1.19): If you use `mysqldump` before upgrading from an affected version and reload the data after upgrading to a higher version, you must drop and recreate your views.

- **Incompatible change**: The parser accepted statements that contained `/* ... */` that were not properly closed with `*/`, such as `SELECT 1 /* + 2`. As of MySQL 5.0.50, statements that contain unclosed `/*`-comments now are rejected with a syntax error.

  This fix has the potential to cause incompatibilities. Because of Bug #26302, which caused the trailing `*/` to be truncated from comments in views, stored routines, triggers, and events, it is possible that objects of those types may have been stored with definitions that now will be rejected as syntactically invalid. Such objects should be dropped and re-created so that their definitions do not contain truncated comments. If a stored object definition contains only a single statement (does not use a `BEGIN ... END` block) and contains a comment within the statement, the comment should be moved to follow the statement or the object should be rewritten to use a `BEGIN ... END` block. For example, this statement:

  ```
  CREATE PROCEDURE p() SELECT 1 /* my comment */ ;
  ```

  Can be rewritten in either of these ways:

  ```
  CREATE PROCEDURE p() SELECT 1; /* my comment */
  CREATE PROCEDURE p() BEGIN SELECT 1 /* my comment */ ; END;
  ```

- **Incompatible change**: If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a database qualifier (that is, use *`db_name.func_name()`* syntax). See Section 9.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

- **Incompatible change**: As of MySQL 5.0.15, the `CHAR()` function returns a binary string rather than a string in the connection character set. An optional `USING charset_name` clause may be used to produce a result in a specific character set instead. Also, arguments larger than 256 produce multiple characters. They are no longer interpreted modulo 256 to produce a single character each. These changes may cause some incompatibilities:

  - `CHAR(ORD('A')) = 'a'` is no longer true:

    ```
    mysql> SELECT CHAR(ORD('A')) = 'a';
    +----------------------+
    | CHAR(ORD('A')) = 'a' |
    ```


This documentation is for an older version. If you're

```
+---------------------+
|                   0 |
+---------------------+
```

To perform a case-insensitive comparison, you can produce a result string in a nonbinary character set by adding a `USING` clause or converting the result:

```
mysql> SELECT CHAR(ORD('A') USING latin1) = 'a';
+----------------------------------+
| CHAR(ORD('A') USING latin1) = 'a' |
+----------------------------------+
|                                1 |
+----------------------------------+
mysql> SELECT CONVERT(CHAR(ORD('A')) USING latin1) = 'a';
+-------------------------------------------+
| CONVERT(CHAR(ORD('A')) USING latin1) = 'a' |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
```

- **Incompatible change**: Beginning with MySQL 5.0.12, natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard. The changes include elimination of redundant output columns for `NATURAL` joins and joins specified with a `USING` clause and proper ordering of output columns. The precedence of the comma operator also now is lower compared to `JOIN`, `LEFT JOIN`, and so forth.

  These changes make MySQL more compliant with standard SQL. However, they can result in different output columns for some joins. Also, some queries that appeared to work correctly prior to 5.0.12 must be rewritten to comply with the standard. For details about the scope of the changes and examples that show what query rewrites are necessary, see Section 13.2.8.2, “JOIN Syntax”.

- `CREATE TABLE ... SELECT CHAR(...)` produces a `VARBINARY` column, not a `VARCHAR` column. To produce a `VARCHAR` column, use `USING` or `CONVERT()` as just described to convert the `CHAR()` result into a nonbinary character set.

- Previously, the following statements inserted the value `0x00410041` (`'AA'` as a `ucs2` string) into the table:

  ```
  CREATE TABLE t (ucs2_column CHAR(2) CHARACTER SET ucs2);
  INSERT INTO t VALUES (CHAR(0x41,0x41));
  ```

  As of MySQL 5.0.15, the statements insert a single `ucs2` character with value `0x4141`.

- **Incompatible change**: By default, integer subtraction involving an unsigned value should produce an unsigned result. Tracking of the “unsignedness” of an expression was improved in MySQL 5.0.13. This means that, in some cases where an unsigned subtraction would have resulted in a signed integer, it now results in an unsigned integer. One context in which this difference manifests itself is when a subtraction involving an unsigned operand would be negative.

  Suppose that `i` is a `TINYINT UNSIGNED` column and has a value of 0. The server evaluates the following expression using 64-bit unsigned integer arithmetic with the following result:

  ```
  mysql> SELECT i - 1 FROM t;
  +----------------------+
  | i - 1                |
  +----------------------+
  | 18446744073709551615 |
  +----------------------+
  ```


This documentation is for an older version. If you're

If the expression is used in an `UPDATE t SET i = i - 1` statement, the expression is evaluated and the result assigned to `i` according to the usual rules for handling values outside the column range or 0 to 255. That is, the value is clipped to the nearest endpoint of the range. However, the result is version-specific:

- Before MySQL 5.0.13, the expression is evaluated but is treated as the equivalent 64-bit signed value (−1) for the assignment. The value of −1 is clipped to the nearest endpoint of the column range, resulting in a value of 0:

```
mysql> UPDATE t SET i = i - 1; SELECT i FROM t;
+------+
| i    |
+------+
|    0 |
+------+
```

- As of MySQL 5.0.13, the expression is evaluated and retains its unsigned attribute for the assignment. The value of 18446744073709551615 is clipped to the nearest endpoint of the column range, resulting in a value of 255:

```
mysql> UPDATE t SET i = i - 1; SELECT i FROM t;
+------+
| i    |
+------+
|  255 |
+------+
```

To get the older behavior, use `CAST()` to convert the expression result to a signed value:

```
UPDATE t SET i = CAST(i - 1 AS SIGNED);
```

Alternatively, set the `NO_UNSIGNED_SUBTRACTION` SQL mode. However, this will affect all integer subtractions involving unsigned values.

- **Incompatible change**: Before MySQL 5.0.12, `NOW()` and `SYSDATE()` return the same value (the time at which the statement in which the function occurs begins executing). As of MySQL 5.0.12, `SYSDATE()` returns the time at which it executes, which can differ from the value returned by `NOW()`. For information about the implications for binary logging, replication, and use of indexes, see the description for `SYSDATE()` in Section 12.7, “Date and Time Functions” and for `SET TIMESTAMP` in Section 13.7.4, “SET Syntax”. To restore the former behavior for `SYSDATE()` and cause it to be an alias for `NOW()`, start the server with the `--sysdate-is-now` option (available as of MySQL 5.0.20).

- **Incompatible change**: Before MySQL 5.0.13, `GREATEST(`*x*`,NULL)` and `LEAST(`*x*`,NULL)` return *x* when *x* is a non-`NULL` value. As of 5.0.13, both functions return `NULL` if any argument is `NULL`, the same as Oracle. This change can cause problems for applications that rely on the old behavior.

- **Incompatible change**: Before MySQL 5.0.8, conversion of `DATETIME` values to numeric form by adding zero produced a result in `YYYYMMDDHHMMSS` format. The result of `DATETIME+0` is now in `YYYYMMDDHHMMSS.000000` format.

- **Incompatible change**: In MySQL 5.0.6, the behavior of `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` has changed when the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values both are empty. Formerly, a column was read or written using the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. Now columns are read and written using a field width wide enough to hold all values in the field. However, data files written before this change was

This documentation is for an older version. If you're

made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 5.0.6 and up. This change also affects data files read by `mysqlimport` and written by `mysqldump --tab`, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`. For more information, see Section 13.2.6, “LOAD DATA INFILE Syntax”.

- **Incompatible change**: Before MySQL 5.0.2, `SHOW STATUS` returned global status values. The default as of 5.0.2 is to return session values, which is incompatible with previous versions. To issue a `SHOW STATUS` statement that will retrieve global status values for all versions of MySQL, write it like this:

```
SHOW /*!50002 GLOBAL */ STATUS;
```

- **Incompatible change**: User variables are not case sensitive in MySQL 5.0. In MySQL 4.1, `SET @x = 0; SET @X = 1; SELECT @x;` created two variables and returned `0`. In MySQL 5.0, it creates one variable and returns `1`. Replication setups that rely on the old behavior may be affected by this change.
- Some keywords may be reserved in MySQL 5.0 that were not reserved in MySQL 4.1. See Section 9.3, “Keywords and Reserved Words”.
- The `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` statements are deprecated. See Section 13.4.2.2, “LOAD DATA FROM MASTER Syntax”, for recommended alternatives.
- As of MySQL 5.0.25, `TIMESTAMP` columns that are `NOT NULL` now are reported that way by `SHOW COLUMNS` and `INFORMATION_SCHEMA`, rather than as `NULL`.
- Comparisons made between `FLOAT` or `DOUBLE` values that happened to work in MySQL 4.1 may not do so in 5.0. Values of these types are imprecise in all MySQL versions, and you are *strongly advised* to avoid such comparisons as `WHERE col_name=some_double`, *regardless of the MySQL version you are using*. See Section B.5.4.8, “Problems with Floating-Point Values”.
- As of MySQL 5.0.3, `BIT` is a separate data type, not a synonym for `TINYINT(1)`. See Section 11.1.1, “Numeric Type Overview”.
- MySQL 5.0.2 adds several SQL modes that enable stricter control over rejecting records that have invalid or missing values. See Section 5.1.7, “Server SQL Modes”, and Section 1.8.3.3, “Constraints on Invalid Data”. If you want to enable this control but continue to use MySQL's capability for storing incorrect dates such as `'2004-02-31'`, you should start the server with `--sql_mode="TRADITIONAL,ALLOW_INVALID_DATES"`.
- As of MySQL 5.0.2, the `SCHEMA` and `SCHEMAS` keywords are accepted as synonyms for `DATABASE` and `DATABASES`, respectively. (While “schemata” is grammatically correct and even appears in some MySQL 5.0 system database and table names, it cannot be used as a keyword.)

C API Changes

- **Incompatible change**: Because the MySQL 5.0 server has a new implementation of the `DECIMAL` data type, a problem may occur if the server is used by older clients that still are linked against MySQL 4.1 client libraries. If a client uses the binary client/server protocol to execute prepared statements that generate result sets containing numeric values, an error will be raised: `'Using unsupported buffer type: 246'`

  This error occurs because the 4.1 client libraries do not support the new `MYSQL_TYPE_NEWDECIMAL` type value added in 5.0. There is no way to disable the new `DECIMAL` data type on the server side. You can avoid the problem by relinking the application with the client libraries from MySQL 5.0.

- **Incompatible change**: The `ER_WARN_DATA_TRUNCATED` warning symbol was renamed to `WARN_DATA_TRUNCATED` in MySQL 5.0.3.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `reconnect` flag in the `MYSQL` structure is set to 0 by `mysql_real_connect()`. Only those client programs which did not explicitly set this flag to 0 or 1 after `mysql_real_connect()` experience a change. Having automatic reconnection enabled by default was considered too dangerous (due to the fact that table locks, temporary tables, user variables, and session variables are lost after reconnection).

2.19.2 Downgrading MySQL

This section describes how to downgrade to an older MySQL version.

- Supported Downgrade Methods
- Supported Downgrade Paths
- Before You Begin
- Performing an In-place Downgrade
- Performing a Logical Downgrade
- Downgrade Troubleshooting

Supported Downgrade Methods

Supported downgrade methods include:

- *In-place Downgrade*: Involves shutting down the new MySQL version, replacing the new MySQL binaries or packages with the old ones, and restarting the old MySQL version on the new data files. In-place downgrades are supported for downgrades between GA versions within the same release series. For example, in-place downgrades are supported for downgrades from 5.0.96 to 5.0.95.
- *Logical Downgrade*: Involves using `mysqldump` to dump all tables from the new MySQL version, and then loading the dump file into the old MySQL version. Logical downgrades are supported for downgrades between versions within the same release series and for downgrades between release levels. For example, logical downgrades are supported for downgrades from 5.0.95 to 5.0.95 and for downgrades from 5.0 to 4.1.

Supported Downgrade Paths

Unless otherwise documented, the following downgrade paths are supported:

- Downgrading from a release series version to an older release series version is supported using all downgrade methods. For example, downgrading from 5.0.96 to 5.0.95 is supported. Skipping release series versions is also supported. For example, downgrading from 5.0.96 to 5.0.92 is supported.
- Downgrading one release level is supported using the *logical downgrade* method. For example, downgrading from 5.0 to 4.1 is supported.
- Downgrading more than one release level is supported using the *logical downgrade* method, but only if you downgrade one release level at a time.

The following conditions apply to all downgrade paths:

- Downgrades between General Availability (GA) status releases are supported.
- Downgrades between milestone releases (or from a GA release to a milestone release) are not supported.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Before You Begin

Before downgrading, the following steps are recommended:

- Review the Release Notes for the MySQL version you are downgrading from to ensure that there are no features or fixes that you really need.
- Review Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”. This section describes changes that may require action before or after downgrading.

Note

The downgrade procedures described in the following sections assume you are downgrading with data files created or modified by the newer MySQL version. However, if you did not modify your data after upgrading, downgrading using backups taken *before* upgrading to the new MySQL version is recommended. Many of the changes described in Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0” that require action before or after downgrading are not applicable when downgrading using backups taken *before* upgrading to the new MySQL version.

- Always back up your current databases and log files before downgrading. The backup should include the `mysql` database, which contains the MySQL system tables. See Section 7.2, “Database Backup Methods”.
- Use of new features, new configuration options, or new configuration option values that are not supported by a previous release may cause downgrade errors or failures. Before downgrading, it is recommended that you reverse changes resulting from the use of new features and remove configuration settings that are not supported by the release you are downgrading to.
- Check Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”, to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are downgrading. If such changes have resulted in an incompatibility between MySQL versions, downgrade the affected tables using the instructions in Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”.
- If you use XA transactions with `InnoDB`, run `XA RECOVER` before downgrading to check for uncommitted XA transactions. If results are returned, either commit or rollback the XA transactions by issuing an `XA COMMIT` or `XA ROLLBACK` statement.

Performing an In-place Downgrade

In-place downgrades are supported for downgrades between GA status releases within the same release series. Review Before you Begin before proceeding.

To perform an in-place downgrade:

1. Review the changes described in Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0” for steps to be performed before downgrading.
2. If you use `InnoDB`, configure MySQL to perform a slow shutdown. For example:

   ```
   shell> bin/mysql -u root -ppassword --execute="set global innodb_fast_shutdown=0"
   ```

   With a slow shutdown, `InnoDB` performs a full purge and change buffer merge before shutting down, which ensures that data files are fully prepared in case of file format differences between releases.
3. Stop the newer MySQL server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4. Downgrade the MySQL binaries or packages in-place by replacing the newer binaries or packages with the older ones.
5. Start the older (downgraded) MySQL server.
6. Run `mysql_upgrade`. For example:

```
shell> bin/mysql_upgrade -u root -ppassword
```

Performing a Logical Downgrade

Logical downgrades are supported for downgrades between releases within the same release series and for downgrades to the previous release level. Only downgrades between General Availability (GA) status releases are supported. Review Before you Begin before proceeding.

To perform a logical downgrade:

1. Review the changes described in Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0” for steps to be performed before downgrading.
2. Dump all databases. For example:

```
shell> bin/mysqldump --add-drop-table --skip-routines --skip-triggers -u root -ppassword
    -> --all-databases --force > all_5_0_databases_dump.sql
```

3. Stop the newer MySQL server.
4. Initialize the older MySQL instance using an empty data directory. For example:

```
shell> scripts/mysql_install_db --user=mysql
```

5. Start the older MySQL server.
6. Load the dump file into the older MySQL server. For example:

```
shell> bin/mysql -u root -ppassword --execute="source all_5_0_databases_dump.sql" --force
```

7. Run `mysql_upgrade`. For example:

```
shell> bin/mysql_upgrade -u root -ppassword
```

8. If you use `InnoDB`, configure MySQL to perform a slow shutdown. For example:

```
shell> bin/mysql -u root -ppassword --execute="set global innodb_fast_shutdown=0"
```

9. Shut down and restart the MySQL server to ensure a clean shutdown and startup.

Downgrade Troubleshooting

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, use `mysqldump` to dump your tables before downgrading. After downgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables. For examples, see Section 2.19.5, “Copying MySQL Databases to Another Machine”.

A typical symptom of a downward-incompatible table format change when you downgrade is that you cannot open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

3. Dump any tables that were inaccessible to the older server by using `mysqldump` to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

2.19.2.1 Changes Affecting Downgrades from MySQL 5.0

Before downgrading from MySQL 5.0, review the changes described in this section. Some changes may require action before or after downgrading.

MySQL 4.1 does not support stored routines or triggers. If your databases contain stored routines or triggers, prevent them from being dumped when you use `mysqldump` by using the `--skip-routines` and `--skip-triggers` options. (See Section 4.5.4, “`mysqldump` — A Database Backup Program”.)

MySQL 4.1 does not support views. If your databases contain views, remove them with `DROP VIEW` before using `mysqldump`. (See Section 13.1.19, “DROP VIEW Syntax”.)

After downgrading from MySQL 5.0, you may see the following information in the `mysql.err` file:

```
Incorrect information in file: './mysql/user.frm'
```

In this case, you can do the following:

1. Start MySQL 5.0.4 (or newer).
2. Run `mysql_fix_privilege_tables`, which will change the `mysql.user` table to a format that both MySQL 4.1 and 5.0 can use.
3. Stop the MySQL server.
4. Start MySQL 4.1.

If the preceding procedure fails, you should be able to do the following instead:

1. Start MySQL 5.0.4 (or newer).
2. Run `mysqldump --opt --add-drop-table mysql > /tmp/mysql.dump`.
3. Stop the MySQL server.
4. Start MySQL 4.1 with the `--skip-grant-tables` option.
5. Run `mysql mysql < /tmp/mysql.dump`.
6. Run `mysqladmin flush-privileges`.

2.19.3 Checking Whether Tables or Indexes Must Be Rebuilt

A binary upgrade or downgrade is one that installs one version of MySQL “in place” over an existing version, without dumping and reloading tables:

1. Stop the server for the existing version if it is running.
2. Install a different version of MySQL. This is an upgrade if the new version is higher than the original version, a downgrade if the version is lower.
3. Start the server for the new version.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In many cases, the tables from the previous version of MySQL can be used without problem by the new version. However, sometimes changes occur that require tables or table indexes to be rebuilt, as described in this section. If you have tables that are affected by any of the issues described here, rebuild the tables or indexes as necessary using the instructions given in Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”.

Table Incompatibilities

After a binary upgrade to MySQL 5.1 from a MySQL 5.0 installation that contains `ARCHIVE` tables, accessing those tables causes the server to crash, even if you have run `mysql_upgrade` or `CHECK TABLE ... FOR UPGRADE`. To work around this problem, use `mysqldump` to dump all `ARCHIVE` tables before upgrading, and reload them into MySQL 5.1 after upgrading. The same problem occurs for binary downgrades from MySQL 5.1 to 5.0.

Index Incompatibilities

If you perform a binary upgrade without dumping and reloading tables, you cannot upgrade directly from MySQL 4.1 to 5.1 or higher. This occurs due to an incompatible change in the `MyISAM` table index format in MySQL 5.0. Upgrade from MySQL 4.1 to 5.0 and repair all `MyISAM` tables. Then upgrade from MySQL 5.0 to 5.1 and check and repair your tables.

Modifications to the handling of character sets or collations might change the character sort order, which causes the ordering of entries in any index that uses an affected character set or collation to be incorrect. Such changes result in several possible problems:

- Comparison results that differ from previous results
- Inability to find some index values due to misordered index entries
- Misordered `ORDER BY` results
- Tables that `CHECK TABLE` reports as being in need of repair

The solution to these problems is to rebuild any indexes that use an affected character set or collation, either by dropping and re-creating the indexes, or by dumping and reloading the entire table. In some cases, it is possible to alter affected columns to use a different collation. For information about rebuilding indexes, see Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”.

To check whether a table has indexes that must be rebuilt, consult the following list. It indicates which versions of MySQL introduced character set or collation changes that require indexes to be rebuilt. Each entry indicates the version in which the change occurred and the character sets or collations that the change affects. If the change is associated with a particular bug report, the bug number is given.

The list applies both for binary upgrades and downgrades. For example, Bug #27877 was fixed in MySQL 5.1.24, so it applies to upgrades from versions older than 5.1.24 to 5.1.24 or newer, and to downgrades from 5.1.24 or newer to versions older than 5.1.24.

In many cases, you can use `CHECK TABLE ... FOR UPGRADE` to identify tables for which index rebuilding is required. It will report this message:

```
Table upgrade required.
Please do "REPAIR TABLE `tbl_name`" to fix it!
```

In these cases, you can also use `mysqlcheck --check-upgrade` or `mysql_upgrade`, which execute `CHECK TABLE`. However, the use of `CHECK TABLE` applies only after upgrades, not downgrades. Also, `CHECK TABLE` is not applicable to all storage engines. For details about which storage engines `CHECK TABLE` supports, see Section 13.7.2.3, “CHECK TABLE Syntax”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

These changes cause index rebuilding to be necessary:

- MySQL 5.1.24 (Bug #27877)

  Affects indexes that use the `utf8_general_ci` or `ucs2_general_ci` collation for columns that contain `'ß'` LATIN SMALL LETTER SHARP S (German). The bug fix corrected an error in the original collations but introduced an incompatibility such that `'ß'` compares equal to characters with which it previously compared different.

  Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.30 (see Bug #40053).

  A workaround for this issue is implemented as of MySQL 5.1.62, 5.5.21, and 5.6.5. The workaround involves altering affected columns to use the `utf8_general_mysql500_ci` and `ucs2_general_mysql500_ci` collations, which preserve the original pre-5.1.24 ordering of `utf8_general_ci` and `ucs2_general_ci`.

- MySQL 5.0.48, 5.1.23 (Bug #27562)

  Affects indexes that use the `ascii_general_ci` collation for columns that contain any of these characters: `'`'` GRAVE ACCENT, `'['` LEFT SQUARE BRACKET, `'\'` REVERSE SOLIDUS, `']'` RIGHT SQUARE BRACKET, `'~'` TILDE

  Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.29 (see Bug #39585).

- MySQL 5.0.48, 5.1.21 (Bug #29461)

  Affects indexes for columns that use any of these character sets: `eucjpms`, `euc_kr`, `gb2312`, `latin7`, `macce`, `ujis`

  Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.29 (see Bug #39585).

2.19.4 Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild a table. This can be necessitated by changes to MySQL such as how data types are handled or changes to character set handling. For example, an error in a collation might have been corrected, necessitating a table rebuild to update the indexes for character columns that use the collation. (For examples, see Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.) It might also be that a table repair or upgrade should be done as indicated by a table check operation such as that performed by `CHECK TABLE`, `mysqlcheck`, or `mysql_upgrade`.

Methods for rebuilding a table include dumping and reloading it, or using `ALTER TABLE` or `REPAIR TABLE`.

Note

If you are rebuilding tables because a different version of MySQL will not handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
shell> mysqldump db_name t1 > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
shell> mysqldump db_name > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` option:

```
shell> mysqldump --all-databases > dump.sql
shell> mysql < dump.sql
```

To rebuild a table with `ALTER TABLE`, use a “null” alteration; that is, an `ALTER TABLE` statement that “changes” the table to use the storage engine that it already has. For example, if `t1` is a `MyISAM` table, use this statement:

```
mysql> ALTER TABLE t1 ENGINE = MyISAM;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

If you must rebuild a table because a table checking operation indicates that the table is corrupt or needs an upgrade, you can use `REPAIR TABLE` if that statement supports the table's storage engine. For example, to repair a `MyISAM` table, use this statement:

```
mysql> REPAIR TABLE t1;
```

For storage engines such as `InnoDB` that `REPAIR TABLE` does not support, use `mysqldump` to create a dump file and `mysql` to reload the file, as described earlier.

For specifics about which storage engines `REPAIR TABLE` supports, see Section 13.7.2.6, “REPAIR TABLE Syntax”.

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases` or `--all-databases` option to repair all tables in specific databases or all databases, respectively:

```
shell> mysqlcheck --repair --databases db_name ...
shell> mysqlcheck --repair --all-databases
```

For incompatibilities introduced in MySQL 5.1.24 by the fix for Bug #27877 that corrected the `utf8_general_ci` and `ucs2_general_ci` collations, a workaround is implemented as of MySQL 5.1.62, 5.5.21, and 5.6.5. Upgrade to one of those versions, then convert each affected table using one of the following methods. In each case, the workaround altering affected columns to use the `utf8_general_mysql500_ci` and `ucs2_general_mysql500_ci` collations, which preserve the original pre-5.1.24 ordering of `utf8_general_ci` and `ucs2_general_ci`.

- To convert an affected table after a binary upgrade that leaves the table files in place, alter the table to use the new collation. Suppose that the table `t1` contains one or more problematic `utf8` columns. To convert the table at the table level, use a statement like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ALTER TABLE t1
CONVERT TO CHARACTER SET utf8 COLLATE utf8_general_mysql500_ci;
```

To apply the change on a column-specific basis, use a statement like this (be sure to repeat the column definition as originally specified except for the `COLLATE` clause):

```
ALTER TABLE t1
MODIFY c1 CHAR(N) CHARACTER SET utf8 COLLATE utf8_general_mysql500_ci;
```

- To upgrade the table using a dump and reload procedure, dump the table using `mysqldump`, modify the `CREATE TABLE` statement in the dump file to use the new collation, and reload the table.

After making the appropriate changes, `CHECK TABLE` should report no error.

2.19.5 Copying MySQL Databases to Another Machine

You can copy the `.frm`, `.MYI`, and `.MYD` files for `MyISAM` tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See Section 14.1, “The MyISAM Storage Engine”.

In cases where you need to transfer databases between different architectures, you can use `mysqldump` to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the `mysql` client.

Use `mysqldump --help` to see what options are available.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For large tables, this is much faster than simply using `mysqldump`. In the following commands, *`DUMPDIR`* represents the full path name of the directory you use to store the output from `mysqldump`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the *`DUMPDIR`* directory to some corresponding directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.20 Operating System-Specific Notes

2.20.1 Linux Notes

This section discusses issues that have been found to occur on Linux. The first few subsections describe general operating system-related issues, problems that can occur when using binary or source distributions, and postinstallation issues. The remaining subsections discuss problems that occur with Linux on specific platforms.

Note that most of these problems occur on older versions of Linux. If you are running a recent version, you may see none of them.

2.20.1.1 Linux Operating System Notes

MySQL needs at least Linux version 2.0.

Warning

We have seen some strange problems with Linux 2.2.14 and MySQL on SMP systems. We also have reports from some MySQL users that they have encountered serious stability problems using MySQL with kernel 2.2.14. If you are using this kernel, you should upgrade to 2.2.19 (or newer) or to a 2.4 kernel. If you have a multiple-CPU box, you should seriously consider using 2.4 because it gives you a significant speed boost. Your system should be more stable.

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

2.20.1.2 Linux Binary Distribution Notes

The Linux-Intel binary and RPM releases of MySQL are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

The binary release is linked with `-static`, which means you do not normally need to worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with

This documentation is for an older version. If you're

`-static` is slightly larger than a dynamically linked program, but also slightly faster (3% to 5%). However, one problem with a statically linked program is that you can't use user-defined functions (UDFs). If you are going to write or use UDFs (this is something for C or C++ programmers only), you must compile MySQL yourself using dynamic linking.

A known issue with binary distributions is that on older Linux systems that use `libc` (such as Red Hat 4.x or Slackware), you get some (nonfatal) issues with host name resolution. If your system uses `libc` rather than `glibc2`, you probably will encounter some difficulties with host name resolution and `getpwnam()`. This happens because `glibc` (unfortunately) depends on some external libraries to implement host name resolution and `getpwent()`, even when compiled with `-static`. These problems manifest themselves in two ways:

- You may see the following error message when you run `mysql_install_db`:

  ```
  Sorry, the host 'xxxx' could not be looked up
  ```

  You can deal with this by executing `mysql_install_db --force`, which does not execute the `resolveip` test in `mysql_install_db`. The downside is that you cannot use host names in the grant tables: except for `localhost`, you must use IP addresses instead. If you are using an old version of MySQL that does not support `--force`, you must manually remove the `resolveip` test in `mysql_install_db` using a text editor.

- You also may see the following error when you try to run `mysqld` with the `--user` option:

  ```
  getpwnam: No such file or directory
  ```

  To work around this problem, start `mysqld` by using the `su` command rather than by specifying the `--user` option. This causes the system itself to change the user ID of the `mysqld` process so that `mysqld` need not do so.

Another solution, which solves both problems, is not to use a binary distribution. Obtain a MySQL source distribution (in RPM or `.tar.gz` format) and install that instead.

On some Linux 2.2 versions, you may get the error `Resource temporarily unavailable` when clients make a great many new connections to a `mysqld` server over TCP/IP. The problem is that Linux has a delay between the time that you close a TCP/IP socket and the time that the system actually frees it. There is room for only a finite number of TCP/IP slots, so you encounter the resource-unavailable error if clients attempt too many new TCP/IP connections over a short period of time. For example, you may see the error when you run the MySQL `test-connect` benchmark over TCP/IP.

We have inquired about this problem a few times on different Linux mailing lists but have never been able to find a suitable resolution. The only known “fix” is for clients to use persistent connections, or, if you are running the database server and clients on the same machine, to use Unix socket file connections rather than TCP/IP connections.

2.20.1.3 Linux Source Distribution Notes

The following notes regarding `glibc` apply only to the situation when you build MySQL yourself. If you are running Linux on an x86 machine, in most cases it is much better for you to use our binary. We link our binaries against the best patched version of `glibc` we can find and with the best compiler options, in an attempt to make it suitable for a high-load server. For a typical user, even for setups with a lot of concurrent connections or tables exceeding the 2GB limit, our binary is the best choice in most cases. After reading the following text, if you are in doubt about what to do, try our binary first to determine whether it meets your needs. If you discover that it is not good enough, you may want to try your own build. In that case, we would appreciate a note about it so that we can build a better binary next time.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL uses LinuxThreads on Linux. If you are using an old Linux version that doesn't have `glibc2`, you must install LinuxThreads before trying to compile MySQL. You can obtain LinuxThreads from http://dev.mysql.com/downloads/os-linux.html.

Note that `glibc` versions before and including version 2.1.1 have a fatal bug in `pthread_mutex_timedwait()` handling, which is used when `INSERT DELAYED` statements are issued. Do not use `INSERT DELAYED` before upgrading `glibc`.

Note that Linux kernel and the LinuxThread library can by default handle a maximum of 1,024 threads. If you plan to have more than 1,000 concurrent connections, you need to make some changes to LinuxThreads, as follows:

- Increase `PTHREAD_THREADS_MAX` in `sysdeps/unix/sysv/linux/bits/local_lim.h` to 4096 and decrease `STACK_SIZE` in `linuxthreads/internals.h` to 256KB. The paths are relative to the root of `glibc`. (Note that MySQL is not stable with 600 to 1000 connections if `STACK_SIZE` is the default of 2MB.)
- Recompile LinuxThreads to produce a new `libpthread.a` library, and relink MySQL against it.

There is another issue that greatly hurts MySQL performance, especially on SMP systems. The mutex implementation in LinuxThreads in `glibc` 2.1 is very poor for programs with many threads that hold the mutex only for a short time. This produces a paradoxical result: If you link MySQL against an unmodified LinuxThreads, removing processors from an SMP actually improves MySQL performance in many cases. We have made a patch available for `glibc` 2.1.3 to correct this behavior (http://dev.mysql.com/Downloads/Linux/linuxthreads-2.1-patch).

With `glibc` 2.2.2, MySQL uses the adaptive mutex, which is much better than even the patched one in `glibc` 2.1.3. Be warned, however, that under some conditions, the current mutex code in `glibc` 2.2.2 overspins, which hurts MySQL performance. The likelihood that this condition occurs can be reduced by re-nicing the `mysqld` process to the highest priority. We have also been able to correct the overspin behavior with a patch, available at http://dev.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch. It combines the correction of overspin, maximum number of threads, and stack spacing all in one. You need to apply it in the `linuxthreads` directory with `patch -p0 </tmp/linuxthreads-2.2.2.patch`. We hope it is included in some form in future releases of `glibc` 2.2. In any case, if you link against `glibc` 2.2.2, you still need to correct `STACK_SIZE` and `PTHREAD_THREADS_MAX`. We hope that the defaults is corrected to some more acceptable values for high-load MySQL setup in the future, so that the commands needed to produce your own build can be reduced to `./configure; make; make install`.

If you use these patches to build a special static version of `libpthread.a`, use it only for statically linking against MySQL. We know that these patches are safe for MySQL and significantly improve its performance, but we cannot say anything about their effects on other applications. If you link other applications that require LinuxThreads against the patched static version of the library, or build a patched shared version and install it on your system, you do so at your own risk.

If you experience any strange problems during the installation of MySQL, or with some common utilities hanging, it is very likely that they are either library or compiler related. If this is the case, using our binary resolves them.

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).

This documentation is for an older version. If you're

- Copy `libmysqclient.so` to `/usr/lib`.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler (`fcc/FCC`), you may have some problems compiling MySQL because the Linux header files are very `gcc` oriented. The following `configure` line should work with `fcc/FCC`:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
    -DCONST=const -DNO_STRTOLL_PROTO" \
CXX=FCC CXXFLAGS="-O -K fast -K lib \
    -K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE \
    -DCONST=const -Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
    '-D_EXTERN_INLINE=static __inline'" \
./configure \
    --prefix=/usr/local/mysql --enable-assembler \
    --with-mysqld-ldflags=-all-static --disable-shared \
    --with-low-memory
```

2.20.1.4 Linux Postinstallation Notes

`mysql.server` can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

If MySQL cannot open enough files or connections, it may be that you have not configured Linux to handle enough files.

In Linux 2.2 and onward, you can check the number of allocated file handles as follows:

```
shell> cat /proc/sys/fs/file-max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

If you have more than 16MB of memory, you should add something like the following to your init scripts (for example, `/etc/init.d/boot.local` on SuSE Linux):

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

You can also run the `echo` commands from the command line as `root`, but these settings are lost the next time your computer restarts.

Alternatively, you can set these parameters on startup by using the `sysctl` tool, which is used by many Linux distributions (including SuSE Linux 8.0 and later). Put the following values into a file named `/etc/sysctl.conf`:

```
# Increase some values for MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

You should also add the following to `/etc/my.cnf`:

```
[mysqld_safe]
open-files-limit=8192
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This should enable a server limit of 8,192 for the combined number of connections and open files.

The `STACK_SIZE` constant in LinuxThreads controls the spacing of thread stacks in the address space. It needs to be large enough so that there is plenty of room for each individual thread stack, but small enough to keep the stack of some threads from running into the global `mysqld` data. Unfortunately, as we have experimentally discovered, the Linux implementation of `mmap()` successfully unmaps a mapped region if you ask it to map out an address currently in use, zeroing out the data on the entire page instead of returning an error. So, the safety of `mysqld` or any other threaded application depends on the "gentlemanly" behavior of the code that creates threads. The user must take measures to make sure that the number of running threads at any given time is sufficiently low for thread stacks to stay away from the global heap. With `mysqld`, you should enforce this behavior by setting a reasonable value for the `max_connections` variable.

If you build MySQL yourself, you can patch LinuxThreads for better stack use. See Section 2.20.1.3, "Linux Source Distribution Notes". If you do not want to patch LinuxThreads, you should set `max_connections` to a value no higher than 500. It should be even less if you have a large key buffer, large heap tables, or some other things that make `mysqld` allocate a lot of memory, or if you are running a 2.2 kernel with a 2GB patch. If you are using our binary or RPM version, you can safely set `max_connections` at 1500, assuming no large key buffer or heap tables with lots of data. The more you reduce `STACK_SIZE` in LinuxThreads the more threads you can safely create. Values between 128KB and 256KB are recommended.

If you use a lot of concurrent connections, you may suffer from a "feature" in the 2.2 kernel that attempts to prevent fork bomb attacks by penalizing a process for forking or cloning a child. This causes MySQL not to scale well as you increase the number of concurrent clients. On single-CPU systems, we have seen this manifest as very slow thread creation; it may take a long time to connect to MySQL (as long as one minute), and it may take just as long to shut it down. On multiple-CPU systems, we have observed a gradual drop in query speed as the number of clients increases. In the process of trying to find a solution, we have received a kernel patch from one of our users who claimed it helped for his site. This patch is available at http://dev.mysql.com/Downloads/Patches/linux-fork.patch. We have done rather extensive testing of this patch on both development and production systems. It has significantly improved MySQL performance without causing any problems and is recommended for users who still run high-load servers on 2.2 kernels.

This issue has been fixed in the 2.4 kernel, so if you are not satisfied with the current performance of your system, rather than patching your 2.2 kernel, it might be easier to upgrade to 2.4. On SMP systems, upgrading also gives you a nice SMP boost in addition to fixing the fairness bug.

We have tested MySQL on the 2.4 kernel on a two-CPU machine and found MySQL scales *much* better. There was virtually no slowdown on query throughput all the way up to 1,000 clients, and the MySQL scaling factor (computed as the ratio of maximum throughput to the throughput for one client) was 180%. We have observed similar results on a four-CPU system: Virtually no slowdown as the number of clients was increased up to 1,000, and a 300% scaling factor. Based on these results, for a high-load SMP server using a 2.2 kernel, it is definitely recommended to upgrade to the 2.4 kernel at this point.

We have discovered that it is essential to run the `mysqld` process with the highest possible priority on the 2.4 kernel to achieve maximum performance. This can be done by adding a `renice -20 $$` command to `mysqld_safe`. In our testing on a four-CPU machine, increasing the priority resulted in a 60% throughput increase with 400 clients.

If you see a dead `mysqld` server process with `ps`, this usually means that you have found a bug in MySQL or you have a corrupted table. See Section B.5.3.3, "What to Do If MySQL Keeps Crashing".

To get a core dump on Linux if `mysqld` dies with a `SIGSEGV` signal, you can start `mysqld` with the `--core-file` option. Note that you also probably need to raise the core file size by adding `ulimit -`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`c 1000000` to `mysqld_safe` or starting `mysqld_safe` with `--core-file-size=1000000`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

2.20.1.5 Linux x86 Notes

MySQL requires `libc` 5.4.12 or newer. It is known to work with `libc` 5.4.46. `glibc` 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from Red Hat, so if you have problems, check whether there are any updates. The `glibc` 2.0.7-19 and 2.0.7-29 RPMs are known to work.

If you are using Red Hat 8.0 or a new `glibc` 2.2.x library, you may see `mysqld` die in `gethostbyaddr()`. This happens because the new `glibc` library requires a stack size greater than 128KB for this call. To fix the problem, start `mysqld` with the `--thread-stack=192K` option. This stack size is the default on MySQL 4.0.10 and above, so you should not see the problem.

If you are using `gcc` 3.0 and above to compile MySQL, you must install the `libstdc++v3` library before compiling MySQL; if you do not do this, you get an error about a missing `__cxa_pure_virtual` symbol during linking.

On some older Linux distributions, `configure` may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says. Add an extra underscore to the `_P` macro name that has only one underscore, and then try again.

You may get some warnings when compiling. Those shown here can be ignored:

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `-1' to
`long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `-1' to
`long unsigned int'
```

If `mysqld` always dumps core when it starts, the problem may be that you have an old `/lib/libc.a`. Try renaming it, and then remove `sql/mysqld` and do a new `make install` and try again. This problem has been reported on some Slackware installations.

If you get the following error when linking `mysqld`, it means that your `libg++.a` is not installed correctly:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using `libg++.a` by running `configure` like this:

```
shell> CXX=gcc ./configure
```

2.20.1.6 Linux SPARC Notes

In some implementations, `readdir_r()` is broken. The symptom is that the `SHOW DATABASES` statement always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from `config.h` after configuring and before compiling.

This documentation is for an older version. If you're

2.20.1.7 Linux Alpha Notes

We have tested MySQL 5.0 on Alpha with our benchmarks and test suite, and it appears to work well.

We currently build the MySQL binary packages on SuSE Linux 7.0 for AXP, kernel 2.4.4-SMP, Compaq C compiler (V6.2-505) and Compaq C++ compiler (V6.3-006) on a Compaq DS20 machine with an Alpha EV6 processor.

You can find the preceding compilers at http://www.support.compaq.com/alpha-tools/. By using these compilers rather than `gcc`, we get about 9% to 14% better MySQL performance.

For MySQL on Alpha, we use the `-arch generic` flag to our compile options, which ensures that the binary runs on all Alpha processors. We also compile statically to avoid library problems. The `configure` command looks like this:

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \
./configure --prefix=/usr/local/mysql --disable-shared \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared
```

Some known problems when running MySQL on Linux-Alpha:

- Debugging threaded applications like MySQL does not work with `gdb 4.18`. You should use `gdb` 5.1 instead.
- If you try linking `mysqld` statically when using `gcc`, the resulting image dumps core at startup time. In other words, *do not* use `--with-mysqld-ldflags=-all-static` with `gcc`.

2.20.1.8 Linux PowerPC Notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

2.20.1.9 Linux MIPS Notes

To get MySQL to work on Qube2 (Linux Mips), you need the newest `glibc` libraries. `glibc-2.0.7-29C2` is known to work. You must also use `gcc` 2.95.2 or newer).

2.20.1.10 Linux IA-64 Notes

To get MySQL to compile on Linux IA-64, we use the following `configure` command for building with `gcc` 2.96:

```
CC=gcc \
CFLAGS="-O3 -fno-omit-frame-pointer" \
CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
    ./configure --prefix=/usr/local/mysql \
    "--with-comment=Official MySQL binary" \
    --with-extra-charsets=complex
```

On IA-64, the MySQL client binaries use shared libraries. This means that if you install our binary distribution at a location other than `/usr/local/mysql`, you need to add the path of the directory where you have `libmysqlclient.so` installed either to the `/etc/ld.so.conf` file or to the value of your `LD_LIBRARY_PATH` environment variable.

See Section 20.6.4.1, “Building C API Client Programs”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.20.1.11 SELinux Notes

RHEL4 comes with SELinux, which supports tighter access control for processes. If SELinux is enabled (`SELINUX` in `/etc/selinux/config` is set to `enforcing`, `SELINUXTYPE` is set to either `targeted` or `strict`), you might encounter problems installing Oracle Corporation RPM packages.

Red Hat has an update that solves this. It involves an update of the “security policy” specification to handle the install structure of the RPMs provided by Oracle Corporation. For further information, see https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=167551 and http://rhn.redhat.com/errata/RHBA-2006-0049.html.

2.20.2 OS X Notes

On OS X, `tar` cannot handle long file names. If you need to unpack a `.tar.gz` distribution, use `gnutar` instead.

2.20.2.1 OS X 10.x (Darwin)

MySQL should work without major problems on OS X 10.x (Darwin).

Known issues:

- If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `mysqld`. This runs all threads with the same priority. On OS X, this gives better performance, at least until Apple fixes its thread scheduler.
- The connection times (`wait_timeout`, `interactive_timeout` and `net_read_timeout`) values are not honored.

  This is probably a signal handling problem in the thread library where the signal doesn't break a pending read and we hope that a future update to the thread libraries will fix this.

Our binary for OS X is compiled on Darwin 6.3 with the following `configure` line:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
    ./configure --prefix=/usr/local/mysql \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --enable-local-infile --disable-shared
```

See Section 2.11, “Installing MySQL on OS X”.

2.20.2.2 OS X Server 1.2 (Rhapsody)

For current versions of OS X Server, no operating system changes are necessary before compiling MySQL. Compiling for the Server platform is the same as for the client version of OS X.

For older versions (OS X Server 1.2, a.k.a. Rhapsody), you must first install a pthread package before trying to configure MySQL.

See Section 2.11, “Installing MySQL on OS X”.

2.20.3 Solaris Notes

For information about installing MySQL on Solaris using PKG distributions, see Section 2.13, “Installing MySQL on Solaris”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

On Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

If you get the following error from `configure`, it means that you have something wrong with your compiler installation:

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

In this case, you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is `gcc` 2.95.2 or 3.2. You can find this at http://gcc.gnu.org/. Note that `gcc` 2.8.1 does not work reliably on SPARC.

The recommended `configure` line when using `gcc` 2.95.2 is:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
    --enable-assembler
```

If you have an UltraSPARC system, you can get 4% better performance by adding `-mcpu=v8 -Wa,-xarch=v8plusa` to the `CFLAGS` and `CXXFLAGS` environment variables.

If you have Sun's Forte 5.0 (or newer) compiler, you can run `configure` like this:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit binary with Sun's Forte compiler, use the following configuration options:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit Solaris binary using `gcc`, add `-m64` to `CFLAGS` and `CXXFLAGS` and remove `--enable-assembler` from the `configure` line.

In the MySQL benchmarks, we obtained a 4% speed increase on UltraSPARC when using Forte 5.0 in 32-bit mode, as compared to using `gcc` 3.2 with the `-mcpu` flag.

If you create a 64-bit `mysqld` binary, it is 4% slower than the 32-bit binary, but can handle more threads and memory.

When using Solaris 10 for x86_64, you should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the `InnoDB` storage engine on this platform.

If you get a problem with `fdatasync` or `sched_yield`, you can fix this by adding `LIBS=-lrt` to the `configure` line

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For compilers older than WorkShop 5.3, you might have to edit the `configure` script. Change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

To this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `-Xc` option, the Sun compiler can't compile with the Solaris `pthread.h` header file. This is a Sun bug (broken compiler or broken include file).

If `mysqld` issues the following error message when you run it, you have tried to compile MySQL with the Sun compiler without enabling the `-mt` multi-thread option:

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `-mt` to `CFLAGS` and `CXXFLAGS` and recompile.

If you are using the SFW version of `gcc` (which comes with Solaris 8), you must add `/opt/sfw/lib` to the environment variable `LD_LIBRARY_PATH` before running `configure`.

If you are using the `gcc` available from `sunfreeware.com`, you may have many problems. To avoid this, you should recompile `gcc` and GNU `binutils` on the machine where you are running them.

If you get the following error when compiling MySQL with `gcc`, it means that your `gcc` is not configured for your version of Solaris:

```
shell> gcc -O3 -g -O2 -DDBUG_OFF  -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

The proper thing to do in this case is to get the newest version of `gcc` and compile it with your current `gcc` compiler. At least for Solaris 2.5, almost all binary versions of `gcc` have old, unusable include files that break all programs that use threads, and possibly other programs as well.

Solaris does not provide static versions of all system libraries (`libpthreads` and `libdl`), so you cannot compile MySQL with `--static`. If you try to do so, you get one of the following errors:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqclient.so` to `/usr/lib`.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you have problems with `configure` trying to link with `-lz` when you do not have `zlib` installed, you have two options:

- If you want to be able to use the compressed communication protocol, you need to get and install `zlib` from `ftp.gnu.org`.
- Run `configure` with the `--with-named-z-libs=no` option when building MySQL.

If you are using `gcc` and have problems with loading user-defined functions (UDFs) into MySQL, try adding `-lgcc` to the link line for the UDF.

If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--back_log=50` option as a workaround for this. (Use `-O back_log=50` before MySQL 4.)

To configure the generation of core files on Solaris you should use the `coreadm` command. Because of the security implications of generating a core on a `setuid()` application, by default, Solaris does not support core files on `setuid()` programs. However, you can modify this behavior using `coreadm`. If you enable `setuid()` core files for the current user, they will be generated using the mode 600 and owned by the superuser.

2.20.3.1 Solaris 2.7/2.8 Notes

Normally, you can use a Solaris 2.6 binary on Solaris 2.7 and 2.8. Most of the Solaris 2.6 issues also apply for Solaris 2.7 and 2.8.

MySQL should be able to detect new versions of Solaris automatically and enable workarounds for the following problems.

Solaris 2.7 / 2.8 has some bugs in the include files. You may see the following error when you use `gcc`:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

If this occurs, you can fix the problem by copying `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and changing line 41 from this:

```
#if     !defined(lint) && !defined(__lint)
```

To this:

```
#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

Alternatively, you can edit `/usr/include/widec.h` directly. Either way, after you make the fix, you should remove `config.cache` and run `configure` again.

If you get the following errors when you run `make`, it is because `configure` didn't detect the `curses.h` file (probably because of the error in `/usr/include/widec.h`):

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this problem is to do one of the following:

1. Configure with `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure`.
2. Edit `/usr/include/widec.h` as indicated in the preceding discussion and re-run `configure`.
3. Remove the `#define HAVE_TERM` line from the `config.h` file and run `make` again.

If your linker cannot find `-lz` when linking client programs, the problem is probably that your `libz.so` file is installed in `/usr/local/lib`. You can fix this problem by one of the following methods:

- Add `/usr/local/lib` to `LD_LIBRARY_PATH`.
- Add a link to `libz.so` from `/lib`.
- If you are using Solaris 8, you can install the optional `zlib` from your Solaris 8 CD distribution.
- Run `configure` with the `--with-named-z-libs=no` option when building MySQL.

2.20.3.2 Solaris x86 Notes

On Solaris 8 on x86, `mysqld` dumps core if you remove the debug symbols using `strip`.

If you are using `gcc` on Solaris x86 and you experience problems with core dumps under load, you should use the following `configure` command:

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti -DHAVE_CURSES_H" \
./configure --prefix=/usr/local/mysql
```

This avoids problems with the `libstdc++` library and with C++ exceptions.

If this doesn't help, you should compile a debug version and run it with a trace file or under `gdb`. See Section 21.3, “Debugging and Porting MySQL”.

2.20.4 BSD Notes

This section provides information about using MySQL on variants of BSD Unix.

2.20.4.1 FreeBSD Notes

FreeBSD 4.x or newer is recommended for running MySQL, because the thread package is much more integrated. To get a secure and stable system, you should use only FreeBSD kernels that are marked `-RELEASE`.

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

It is recommended you use MIT-pthreads on FreeBSD 2.x, and native threads on FreeBSD 3 and up. It is possible to run with native threads on some late 2.2.x versions, but you may encounter problems shutting down `mysqld`.

Unfortunately, certain function calls on FreeBSD are not yet fully thread-safe. Most notably, this includes the `gethostbyname()` function, which is used by MySQL to convert host names into IP addresses. Under certain circumstances, the `mysqld` process suddenly causes 100% CPU load and is unresponsive. If you encounter this problem, try to start MySQL using the `--skip-name-resolve` option.

Alternatively, you can link MySQL on FreeBSD 4.x against the LinuxThreads library, which avoids a few of the problems that the native FreeBSD thread implementation has. For a very good comparison of LinuxThreads versus native threads, see Jeremy Zawodny's article *FreeBSD or Linux for your MySQL Server?* at http://jeremy.zawodny.com/blog/archives/000697.html.

Known problem when using LinuxThreads on FreeBSD is:

- The connection times (`wait_timeout`, `interactive_timeout` and `net_read_timeout`) values are not honored. The symptom is that persistent connections can hang for a very long time without getting closed down and that a 'kill' for a thread will not take affect until the thread does it a new command

  This is probably a signal handling problem in the thread library where the signal doesn't break a pending read. This is supposed to be fixed in FreeBSD 5.0

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

The recommended way to compile and install MySQL on FreeBSD with `gcc` (2.95.2 and up) is:

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \
    CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions \
    -felide-constructors -fno-strength-reduce" \
    ./configure --prefix=/usr/local/mysql --enable-assembler
gmake
gmake install
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
bin/mysqld_safe &
```

Be sure that your name resolver setup is correct. Otherwise, you may experience resolver delays or failures when connecting to `mysqld`. Also make sure that the `localhost` entry in the `/etc/hosts` file is correct. The file should start with a line similar to this:

```
127.0.0.1       localhost localhost.your.domain
```

FreeBSD is known to have a very low default file handle limit. See Section B.5.2.18, “File Not Found and Similar Errors”. Start the server by using the `--open-files-limit` option for `mysqld_safe`, or raise the limits for the `mysqld` user in `/etc/login.conf` and rebuild it with `cap_mkdb /etc/login.conf`. Also be sure that you set the appropriate class for this user in the password file if you are not using the default (use `chpass mysqld-user-name`). See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

This documentation is for an older version. If you're

FreeBSD limits the size of a process to 512MB, even if you have much more RAM available on the system. So you may get an error such as this:

```
Out of memory (Needed 16391 bytes)
```

In current versions of FreeBSD (at least 4.x and greater), you may increase this limit by adding the following entries to the `/boot/loader.conf` file and rebooting the machine (these are not settings that can be changed at run time with the `sysctl` command):

```
kern.maxdsiz="1073741824" # 1GB
kern.dfldsiz="1073741824" # 1GB
kern.maxssiz="134217728" # 128MB
```

For older versions of FreeBSD, you must recompile your kernel to change the maximum data segment size for a process. In this case, you should look at the `MAXDSIZ` option in the `LINT` config file for more information.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 2.21, “Environment Variables”.

2.20.4.2 NetBSD Notes

To compile on NetBSD, you need GNU `make`. Otherwise, the build process fails when `make` tries to run `lint` on C++ files.

2.20.4.3 OpenBSD 2.5 Notes

On OpenBSD 2.5, you can compile MySQL with native threads with the following options:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.20.4.4 BSD/OS Version 2.x Notes

If you get the following error when compiling MySQL, your `ulimit` value for virtual memory is too low:

```
item_func.h: In method
`Item_func_ge::Item_func_ge(const Item_func_ge &)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using `ulimit -v 80000` and run `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

If you are using `gcc`, you may also use have to use the `--with-low-memory` flag for `configure` to be able to compile `sql_yacc.cc`.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 2.21, “Environment Variables”.

2.20.4.5 BSD/OS Version 3.x Notes

Upgrade to BSD/OS 3.1. If that is not possible, install BSDIpatch M300-038.

Use the following command when configuring MySQL:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
env CXX=shlicc++ CC=shlicc2 \
./configure \
    --prefix=/usr/local/mysql \
    --localstatedir=/var/mysql \
    --without-perl \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure \
    --prefix=/usr/local/mysql \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `mysqld`. This runs all threads with the same priority. On BSDI 3.1, this gives better performance, at least until BSDI fixes its thread scheduler.

If you get the error `virtual memory exhausted` while compiling, you should try using `ulimit -v 80000` and running `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

2.20.4.6 BSD/OS Version 4.x Notes

BSDI 4.x has some thread-related bugs. If you want to use MySQL on this, you should install all thread-related patches. At least M400-023 should be installed.

On some BSDI 4.x systems, you may get problems with shared libraries. The symptom is that you can't execute any client programs, for example, `mysqladmin`. In this case, you need to reconfigure not to use shared libraries with the `--disable-shared` option to configure.

Some customers have had problems on BSDI 4.0.1 that the `mysqld` binary after a while can't open tables. This occurs because some library/system-related bug causes `mysqld` to change current directory without having asked for that to happen.

The fix is to either upgrade MySQL to at least version 3.23.34 or, after running `configure`, remove the line `#define HAVE_REALPATH` from `config.h` before running `make`.

Note that this means that you can't symbolically link a database directories to another database directory or symbolic link a table to another database on BSDI. (Making a symbolic link to another disk is okay).

2.20.5 Other Unix Notes

2.20.5.1 HP-UX Version 10.20 Notes

If you install MySQL using a binary tarball distribution on HP-UX, you may run into trouble even before you get the MySQL distribution unpacked, as the HP-UX `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

There are a couple of small problems when compiling MySQL on HP-UX. Use `gcc` instead of the HP-UX native compiler, because `gcc` produces better code.

Use `gcc` 2.95 on HP-UX. Do not use high optimization flags (such as `-O6`) because they may not be safe on HP-UX.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following `configure` line should work with `gcc` 2.95:

```
CFLAGS="-I/opt/dce/include -fpic" \
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \
-fno-rtti" \
CXX=gcc \
./configure --with-pthread \
    --with-named-thread-libs='-ldce' \
    --prefix=/usr/local/mysql --disable-shared
```

The following `configure` line should work with `gcc` 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors \
    -fno-exceptions -fno-rtti -O3 -fPIC" \
./configure --prefix=/usr/local/mysql \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --enable-local-infile  --with-pthread \
    --with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC
    --disable-shared
```

2.20.5.2 HP-UX Version 11.x Notes

If you install MySQL using a binary tarball distribution on HP-UX, you may run into trouble even before you get the MySQL distribution unpacked, as the HP-UX `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

Because of some critical bugs in the standard HP-UX libraries, you should install the following patches before trying to run MySQL on HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

This solves the problem of getting `EWOULDBLOCK` from `recv()` and `EBADF` from `accept()` in threaded applications.

If you are using `gcc` 2.95.1 on an unpatched HP-UX 11.x system, you may get the following error:

```
In file included from /usr/include/unistd.h:11,
                 from ../include/global.h:125,
                 from mysql_priv.h:15,
                 from item.cc:19:
/usr/include/sys/unistd.h:184: declaration of C function ...
/usr/include/sys/pthread.h:440: previous declaration ...
In file included from item.h:306,
                 from mysql_priv.h:158,
                 from item.cc:19:
```

The problem is that HP-UX does not define `pthreads_atfork()` consistently. It has conflicting prototypes in `/usr/include/sys/unistd.h`:184 and `/usr/include/sys/pthread.h`:440.

One solution is to copy `/usr/include/sys/unistd.h` into `mysql/include` and edit `unistd.h` and change it to match the definition in `pthread.h`. Look for this line:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
                                          void (*child)());
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Change it to look like this:

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                                          void (*child)(void));
```

After making the change, the following `configure` line should work:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

If you are using HP-UX compiler, you can use the following command (which has been tested with `cc` B.11.11.04):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
    --with-extra-character-set=complex
```

You can ignore any errors of the following type:

```
aCC: warning 901: unknown option: `-3': use +help for online
documentation
```

If you get the following error from `configure`, verify that you do not have the path to the K&R compiler before the path to the HP-UX C and C++ compiler:

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

Another reason for not being able to compile is that you didn't define the `+DD64` flags as just described.

Another possibility for HP-UX 11 is to use the MySQL binaries provided at http://dev.mysql.com/downloads/, which we have built and tested ourselves. We have also received reports that the HP-UX 10.20 binaries supplied by MySQL can be run successfully on HP-UX 11. If you encounter problems, you should be sure to check your HP-UX patch level.

2.20.5.3 IBM-AIX notes

Automatic detection of `xlC` is missing from Autoconf, so a number of variables need to be set before running `configure`. The following example uses the IBM compiler:

```
export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "
export CXX="xlC_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS

./configure --prefix=/usr/local \
                --localstatedir=/var/mysql \
                --sbindir='/usr/local/bin' \
                --libexecdir='/usr/local/bin' \
                --enable-thread-safe-client \
                --enable-large-files
```

The preceding options are used to compile the MySQL distribution that can be found at http://www-frec.bull.com/.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you change the `-O3` to `-O2` in the preceding `configure` line, you must also remove the `-qstrict` option. This is a limitation in the IBM C compiler.

If you are using `gcc` to compile MySQL, you *must* use the `-fno-exceptions` flag, because the exception handling in `gcc` is not thread-safe! There are also some known problems with IBM's assembler that may cause it to generate bad code when used with `gcc`.

Use the following `configure` line with `gcc` 2.95 on AIX:

```
CC="gcc -pipe -mcpu=power -Wa,-many" \
CXX="gcc -pipe -mcpu=power -Wa,-many" \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory
```

The `-Wa,-many` option is necessary for the compile to be successful. IBM is aware of this problem but is in no hurry to fix it because of the workaround that is available. We do not know if the `-fno-exceptions` is required with `gcc` 2.95, but because MySQL doesn't use exceptions and the option generates faster code, you should always use it with `gcc`.

If you get a problem with assembler code, try changing the `-mcpu=xxx` option to match your CPU. Typically `power2`, `power`, or `powerpc` may need to be used. Alternatively, you might need to use `604` or `604e`. We are not positive but suspect that `power` would likely be safe most of the time, even on a power2 machine.

If you do not know what your CPU is, execute a `uname -m` command. It produces a string that looks like `000514676700`, with a format of `xxyyyyyymmss` where `xx` and `ss` are always `00`, `yyyyyy` is a unique system ID and `mm` is the ID of the CPU Planar. A chart of these values can be found at http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm.

This gives you a machine type and a machine model you can use to determine what type of CPU you have.

If you have problems with threads on AIX 5.3, you should upgrade AIX 5.3 to technology level 7 (5300-07).

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring as follows:

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti \
-DDONT_USE_THR_ALARM" \
./configure --prefix=/usr/local/mysql --with-debug \
    --with-low-memory
```

This doesn't affect the performance of MySQL, but has the side effect that you can't kill clients that are “sleeping” on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

On some versions of AIX, linking with `libbind.a` makes `getservbyname()` dump core. This is an AIX bug and should be reported to IBM.

For AIX 4.2.1 and `gcc`, you have to make the following changes.

After configuring, edit `config.h` and `include/my_config.h` and change the line that says this:

```
#define HAVE_SNPRINTF 1
```

to this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
#undef HAVE_SNPRINTF
```

And finally, in `mysqld.cc`, you need to add a prototype for `initgroups()`.

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

For 32-bit binaries, if you need to allocate a lot of memory to the `mysqld` process, it is not enough to just use `ulimit -d unlimited`. You may also have to modify `mysqld_safe` to add a line something like this:

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

You can find more information about using a lot of memory at http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm.

Users of AIX 4.3 should use `gmake` instead of the `make` utility included with AIX.

As of AIX 4.1, the C compiler has been unbundled from AIX as a separate product. `gcc` 3.3.2 can be obtained here: ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/gcc/

The steps for compiling MySQL on AIX with `gcc` 3.3.2 are similar to those for using `gcc` 2.95 (in particular, the need to edit `config.h` and `my_config.h` after running `configure`). However, before running `configure`, you should also patch the `curses.h` file as follows:

```
/opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h.ORIG
       Mon Dec 26 02:17:28 2005
--- /opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h
Mon Dec 26 02:40:13 2005
***************
*** 2023,2029 ****


  #endif /* _AIX32_CURSES */
! #if defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus) || defined
(__STRICT_ANSI__)
  extern int delwin (WINDOW *);
  extern int endwin (void);
  extern int getcurx (WINDOW *);
--- 2023,2029 ----


  #endif /* _AIX32_CURSES */
! #if 0 && (defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus)
|| defined
(__STRICT_ANSI__))
  extern int delwin (WINDOW *);
  extern int endwin (void);
  extern int getcurx (WINDOW *);
```

2.20.5.4 SunOS 4 Notes

On SunOS 4, MIT-pthreads is needed to compile MySQL. This in turn means you need GNU `make`.

Some SunOS 4 systems have problems with dynamic libraries and `libtool`. You can use the following `configure` line to avoid this problem:

```
```


This documentation is for an older version. If you're

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These can be ignored.

When compiling `mysqld`, there are some `implicit declaration of function` warnings. These can be ignored.

2.20.5.5 Alpha-DEC-UNIX Notes (Tru64)

If you are using `egcs` 1.1.2 on Digital Unix, you should upgrade to `gcc` 2.95.2, because `egcs` on DEC has some serious bugs!

When compiling threaded programs under Digital Unix, the documentation recommends using the `-pthread` option for `cc` and `cxx` and the `-lmach -lexc` libraries (in addition to `-lpthread`). You should run `configure` something like this:

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

When compiling `mysqld`, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

You can safely ignore these warnings. They occur because `configure` can detect only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a `SIGHUP` signal.) If so, try starting the server like this:

```
nohup mysqld [options] &
```

`nohup` causes the command following it to ignore any `SIGHUP` signal sent from the terminal. Alternatively, start the server by running `mysqld_safe`, which invokes `mysqld` using `nohup` for you. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

If you get a problem when compiling `mysys/get_opt.c`, just remove the `#define _NO_PROTO` line from the start of that file.

If you are using Compaq's CC compiler, the following `configure` line should work:

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
        -speculate all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
          -speculate all -arch host -noexceptions -nortti"
export CC CFLAGS CXX CXXFLAGS
./configure \
    --prefix=/usr/local/mysql \
    --with-low-memory \
    --enable-large-files \
    --enable-shared=yes \
    --with-named-thread-libs="-lpthread -lmach -lexc -lc"
gnumake
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you get a problem with `libtool` when compiling with shared libraries as just shown, when linking `mysql`, you should be able to get around this by issuing these commands:

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread  -O3 -DDBUG_OFF \
    -O4 -ansi_alias -ansi_args -fast -inline speed \
    -speculate all \ -arch host  -DUNDEF_HAVE_GETHOSTBYNAME_R \
    -o mysql  mysql.o readline.o sql_string.o completion_hash.o \
    ../readline/libreadline.a -lcurses \
    ../libmysql/.libs/libmysqlclient.so  -lm
cd ..
gnumake
gnumake install
scripts/mysql_install_db
```

2.20.5.6 Alpha-DEC-OSF/1 Notes

If you have problems compiling and have DEC `CC` and `gcc` installed, try running `configure` like this:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

If you get problems with the `c_asm.h` file, you can create and use a 'dummy' `c_asm.h` file with:

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Note that the following problems with the `ld` program can be fixed by downloading the latest DEC (Compaq) patch kit from: http://ftp.support.compaq.com/public/unix/.

On OSF/1 V4.0D and compiler "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)," the compiler had some strange behavior (undefined `asm` symbols). `/bin/ld` also appears to be broken (problems with `_exit undefined` errors occurring while linking `mysqld`). On this system, we have managed to compile MySQL with the following `configure` line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

With the Digital compiler "C++ V6.1-029," the following should work:

```
CC=cc -pthread
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
       -speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
         -speculate all -arch host -noexceptions -nortti
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql \
            --with-mysqld-ldflags=-all-static --disable-shared \
            --with-named-thread-libs="-lmach -lexc -lc"
```

In some versions of OSF/1, the `alloca()` function is broken. Fix this by removing the line in `config.h` that defines `'HAVE_ALLOCA'`.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`configure` uses the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using `gcc`, you can also try running `configure` like this:

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring with:

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

This does not affect the performance of MySQL, but has the side effect that you can't kill clients that are "sleeping" on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

With `gcc` 2.95.2, you may encounter the following compile error:

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

To fix this, you should change to the `sql` directory and do a cut-and-paste of the last `gcc` line, but change `-O3` to `-O0` (or add `-O0` immediately after `gcc` if you do not have any `-O` option on your compile line). After this is done, you can just change back to the top-level directory and run `make` again.

2.20.5.7 SGI Irix Notes

Note

As of MySQL 5.0, we do not provide binaries for Irix any more.

If you are using Irix 6.5.3 or newer, `mysqld` is able to create threads only if you run it as a user that has `CAP_SCHED_MGT` privileges (such as `root`) or give the `mysqld` server this privilege with the following shell command:

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

You may have to undefine some symbols in `config.h` after running `configure` and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the `mysqld` server dies on some `SELECT` statements, remove the lines from `config.h` that define `HAVE_ALLOC` and `HAVE_ALLOCA_H`. If `mysqladmin create` doesn't work, remove the line from `config.h` that defines `HAVE_READDIR_R`. You may have to remove the `HAVE_TERM_H` line as well.

SGI recommends that you install all the patches on this page as a set: http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest `rld` rollup, and the latest `libc` rollup.

You definitely need all the POSIX patches on this page, for pthreads support:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling `mysql.cc`:

```
"/usr/include/curses.h", line 82: error(1084):
invalid combination of type
```

Type the following in the top-level directory of your MySQL source tree:

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

There have also been reports of scheduling problems. If only one thread is running, performance is slow. Avoid this by starting another client. This may lead to a two-to-tenfold increase in execution speed thereafter for the other thread. This is a poorly understood problem with Irix threads; you may have to improvise to find solutions until this can be fixed.

If you are compiling with `gcc`, you can use the following `configure` command:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
    --with-named-thread-libs=-lpthread
```

On Irix 6.5.11 with native Irix C and C++ compilers ver. 7.3.1.2, the following is reported to work

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARG:platform=IP22 -I/usr/local/include \
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARG:platform=IP22 \
-I/usr/local/include -L/usr/local/lib' \
./configure --prefix=/usr/local/mysql --with-innodb --with-berkeley-db \
    --with-libwrap=/usr/local \
    --with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.20.5.8 SCO UNIX and OpenServer 5.0.x Notes

The current port is tested only on `sco3.2v5.0.5`, `sco3.2v5.0.6`, and `sco3.2v5.0.7` systems. There has also been progress on a port to `sco3.2v4.2`. Open Server 5.0.8 (Legend) has native threads and permits files greater than 2GB. The current maximum file size is 2GB.

We have been able to compile MySQL with the following `configure` command on OpenServer with `gcc` 2.95.3.

```
CC=gcc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=gcc CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-innodb \
    --with-openssl --with-vio --with-extra-charsets=complex
```

`gcc` is available at ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj.

This development system requires the OpenServer Execution Environment Supplement oss646B on OpenServer 5.0.6 and oss656B and the OpenSource libraries found in gwxlibs. All OpenSource tools are in the `opensrc` directory. They are available at ftp://ftp.sco.com/pub/openserver5/opensrc/.

Use the latest production release of MySQL.

SCO provides operating system patches at ftp://ftp.sco.com/pub/openserver5 for OpenServer 5.0.[0-6] and ftp://ftp.sco.com/pub/openserverv5/507 for OpenServer 5.0.7.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenServer for OpenServer 5.0.x.

The maximum file size on an OpenServer 5.0.x system is 2GB.

The total memory which can be allocated for streams buffers, clists, and lock records cannot exceed 60MB on OpenServer 5.0.x.

Streams buffers are allocated in units of 4096 byte pages, clists are 70 bytes each, and lock records are 64 bytes each, so:

```
(NSTRPAGES * 4096) + (NCLIST * 70) + (MAX_FLCKREC * 64) <= 62914560
```

Follow this procedure to configure the Database Services option. If you are unsure whether an application requires this, see the documentation provided with the application.

1. Log in as `root`.
2. Enable the SUDS driver by editing the `/etc/conf/sdevice.d/suds` file. Change the `N` in the second field to a `Y`.
3. Use `mkdev aio` or the Hardware/Kernel Manager to enable support for asynchronous I/O and relink the kernel. To enable users to lock down memory for use with this type of I/O, update the aiomemlock(F) file. This file should be updated to include the names of users that can use AIO and the maximum amounts of memory they can lock down.
4. Many applications use setuid binaries so that you need to specify only a single user. See the documentation provided with the application to determine whether this is the case for your application.

After you complete this process, reboot the system to create a new kernel incorporating these changes.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

```
Value           Default         Min             Max
-----           -------         ---             ---
NBUF            0               24              450000
NHBUF           0               32              524288
NMPBUF          0               12              512
MAX_INODE       0               100             64000
MAX_FILE        0               100             64000
CTBUFSIZE       128             0               256
MAX_PROC        0               50              16000
MAX_REGION      0               500             160000
NCLIST          170             120             16640
MAXUP           100             15              16000
NOFILES         110             60              11000
NHINODE         128             64              8192
NAUTOUP         10              0               60
NGROUPS         8               0               128
BDFLUSHR        30              1               300
MAX_FLCKREC     0               50              16000
PUTBUFSZ        8000            2000            20000
MAXSLICE        100             25              100
ULIMIT          4194303         2048            4194303
* Streams Parameters
NSTREAM         64              1               32768
NSTRPUSH        9               9               9
NMUXLINK        192             1               4096
STRMSGSZ        16384           4096            524288
STRCTLSZ        1024            1024            1024
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
STRMAXBLK       524288          4096            524288
NSTRPAGES       500             0               8000
STRSPLITFRAC    80              50              100
NLOG            3               3               3
NUMSP           64              1               256
NUMTIM          16              1               8192
NUMTRW          16              1               8192
* Semaphore Parameters
SEMMAP          10              10              8192
SEMMNI          10              10              8192
SEMMNS          60              60              8192
SEMMNU          30              10              8192
SEMMSL          25              25              150
SEMOPM          10              10              1024
SEMUME          10              10              25
SEMVMX          32767           32767           32767
SEMAEM          16384           16384           16384
* Shared Memory Parameters
SHMMAX          524288          131072          2147483647
SHMMIN          1               1               1
SHMMNI          100             100             2000
FILE            0               100             64000
NMOUNT          0               4               256
NPROC           0               50              16000
NREGION         0               500             160000
```

Set these values as follows:

- `NOFILES` should be 4096 or 2048.
- `MAXUP` should be 2048.

To make changes to the kernel, use the `idtune` *`name parameter`* command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. For example, to change `SEMMS` to `200`, execute this command as `root`:

```
# /etc/conf/bin/idtune SEMMNS 200
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

To tune the system, the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `NOFILES` and `MAXUP` should be set to at least 2048.
- `MAXPROC` should be set to at least 3000/4000 (depends on number of users) or more.
- The following formulas are recommended to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

  ```
  SEMMSL = 13
  ```

  13 is what has been found to be the best for both Progress and MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SEMMNS = SEMMSL * number of db servers to be run on the system
```

Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

```
SEMMNU = SEMMNS
```

Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

You need to at least install the SCO OpenServer Linker and Application Development Libraries or the OpenServer Development System to use `gcc`. You cannot use the GCC Dev system without installing one of these.

You should get the FSU Pthreads package and install it first. This can be found at http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz. You can also get a precompiled package from ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz.

FSU Pthreads can be compiled with SCO Unix 4.2 with tcpip, or using OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0) with the SCO Development System installed using a good port of GCC 2.5.x. For ODT or OS 3.0, you need a good port of GCC 2.5.x. There are a lot of problems without a good port. The port for this product requires the SCO Unix Development system. Without it, you are missing the libraries and the linker that is needed. You also need `SCO-3.2v4.2-includes.tar.gz`. This file contains the changes to the SCO Development include files that are needed to get MySQL to build. You need to replace the existing system include files with these modified header files. They can be obtained from ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

To build FSU Pthreads on your system, all you should need to do is run GNU `make`. The `Makefile` in FSU-threads-3.14.tar.gz is set up to make FSU-threads.

You can run `./configure` in the `threads/src` directory and select the SCO OpenServer option. This command copies `Makefile.SCO5` to `Makefile`. Then run `make`.

To install in the default `/usr/include` directory, log in as `root`, and then `cd` to the `thread/src` directory and run `make install`.

Remember that you must use GNU `make` to build MySQL.

Note

If you do not start `mysqld_safe` as `root`, you should get only the default 110 open files per process. `mysqld` writes a note about this in the log file.

With SCO 3.2V4.2, you should use FSU Pthreads version 3.14 or newer. The following `configure` command should work:

```
CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \
./configure \
    --prefix=/usr/local/mysql \
    --with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \
    --with-named-curses-libs="-lcurses"
```

You may have problems with some include files. In this case, you can find new SCO-specific include files at ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

You should unpack this file in the `include` directory of your MySQL source tree.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SCO development notes:

- MySQL should automatically detect FSU Pthreads and link `mysqld` with `-lgthreads -lsocket -lgthreads`.
- The SCO development libraries are re-entrant in FSU Pthreads. SCO claims that its library functions are re-entrant, so they must be re-entrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make re-entrant libraries.
- FSU Pthreads (at least the version at ftp://ftp.zenez.com) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that `gmalloc.o` is included in `libgthreads.a` and `libgthreads.so`.
- In FSU Pthreads, the following system calls are pthreads-aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()`, and `wait()`.
- The CSSA-2001-SCO.35.2 (the patch is listed in custom as erg711905-dscr_remap security patch (version 2.0.0)) breaks FSU threads and makes `mysqld` unstable. You have to remove this one if you want to run `mysqld` on an OpenServer 5.0.6 machine.
- If you use SCO OpenServer 5, you may need to recompile FSU pthreads with `-DDRAFT7` in `CFLAGS`. Otherwise, `InnoDB` may hang at a `mysqld` startup.
- SCO provides operating system patches at ftp://ftp.sco.com/pub/openserver5 for OpenServer 5.0.x.
- SCO provides security fixes and `libsocket.so.2` at ftp://ftp.sco.com/pub/security/OpenServer and ftp://ftp.sco.com/pub/security/sse for OpenServer 5.0.x.
- Pre-OSR506 security fixes. Also, the `telnetd` fix at ftp://stage.caldera.com/pub/security/openserver/ or ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/ as both `libsocket.so.2` and `libresolv.so.1` with instructions for installing on pre-OSR506 systems.

  It is probably a good idea to install these patches before trying to compile/use MySQL.

Beginning with Legend/OpenServer 6.0.0, there are native threads and no 2GB file size limit.

2.20.5.9 SCO OpenServer 6.0.x Notes

OpenServer 6 includes these key improvements:

- Larger file support up to 1 TB
- Multiprocessor support increased from 4 to 32 processors
- Increased memory support up to 64GB
- Extending the power of UnixWare into OpenServer 6
- Dramatic performance improvement

OpenServer 6.0.0 commands are organized as follows:

- `/bin` is for commands that behave exactly the same as on OpenServer 5.0.x.
- `/u95/bin` is for commands that have better standards conformance, for example Large File System (LFS) support.
- `/udk/bin` is for commands that behave the same as on UnixWare 7.1.4. The default is for the LFS support.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following is a guide to setting `PATH` on OpenServer 6. If the user wants the traditional OpenServer 5.0.x then `PATH` should be `/bin` first. If the user wants LFS support, the path should be `/u95/bin:/bin`. If the user wants UnixWare 7 support first, the path would be `/udk/bin:/u95/bin:/bin:`.

Use the latest production release of MySQL. Should you choose to use an older release of MySQL on OpenServer 6.0.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

MySQL distribution files with names of the following form are `tar` archives of media are tar archives of media images suitable for installation with the SCO Software Manager (`/etc/custom`) on SCO OpenServer 6:

```
mysql-PRODUCT-5.0.96-sco-osr6-i686.VOLS.tar
```

A distribution where *`PRODUCT`* is `pro-cert` is the Commercially licensed MySQL Pro Certified server. A distribution where *`PRODUCT`* is `pro-gpl-cert` is the MySQL Pro Certified server licensed under the terms of the General Public License (GPL).

Select whichever distribution you wish to install and, after download, extract the `tar` archive into an empty directory. For example:

```
shell> mkdir /tmp/mysql-pro
shell> cd /tmp/mysql-pro
shell> tar xf /tmp/mysql-pro-cert-5.0.96-sco-osr6-i686.VOLS.tar
```

Prior to installation, back up your data in accordance with the procedures outlined in Section 2.19.1, “Upgrading MySQL”.

Remove any previously installed `pkgadd` version of MySQL:

```
shell> pkginfo mysql 2>&1 > /dev/null && pkgrm mysql
```

Install MySQL Pro from media images using the SCO Software Manager:

```
shell> /etc/custom -p SCO:MySQL -i -z /tmp/mysql-pro
```

Alternatively, the SCO Software Manager can be displayed graphically by clicking the `Software Manager` icon on the desktop, selecting `Software -> Install New`, selecting the host, selecting `Media Images` for the Media Device, and entering `/tmp/mysql-pro` as the Image Directory.

After installation, run `mkdev mysql` as the `root` user to configure your newly installed MySQL Pro Certified server.

Note

The installation procedure for VOLS packages does not create the `mysql` user and group that the package uses by default. You should either create the `mysql` user and group, or else select a different user and group using an option in `mkdev mysql`.

If you wish to configure your MySQL Pro server to interface with the Apache Web server using PHP, download and install the PHP update from SCO at ftp://ftp.sco.com/pub/updates/OpenServer/SCOSA-2006.17/.

We have been able to compile MySQL with the following `configure` command on OpenServer 6.0.x:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CC=cc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=CC CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db \
    --with-extra-charsets=complex \
    --build=i686-unknown-sysv5SCO_SV6.0.0
```

If you use `gcc`, you must use `gcc` 2.95.3 or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

The version of Berkeley DB that comes with either UnixWare 7.1.4 or OpenServer 6.0.0 is not used when building MySQL. MySQL instead uses its own version of Berkeley DB. The `configure` command needs to build both a static and a dynamic library in *`src_directory`*`/bdb/build_unix/`, but it does not with MySQL's own `BDB` version. The workaround is as follows.

1. Configure as normal for MySQL.
2. `cd bdb/build_unix/`
3. `cp -p Makefile Makefile.sav`
4. Use same options and run `../dist/configure`.
5. Run `gmake`.
6. `cp -p Makefile.sav Makefile`
7. Change location to the top source directory and run `gmake`.

This enables both the shared and dynamic libraries to be made and work.

SCO provides OpenServer 6 operating system patches at ftp://ftp.sco.com/pub/openserver6.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenServer.

By default, the maximum file size on a OpenServer 6.0.0 system is 1TB. Some operating system utilities have a limitation of 2GB. The maximum possible file size on UnixWare 7 is 1TB with VXFS or HTFS.

OpenServer 6 can be configured for large file support (file sizes greater than 2GB) by tuning the UNIX kernel.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
```

To make changes to the kernel, use the `idtune` *`name parameter`* command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. To set the kernel values, execute the following commands as `root`:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

To tune the system, the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `SFNOLIM` and `HFNOLIM` should be at maximum 2048.
- `NPROC` should be set to at least 3000/4000 (depends on number of users).
- The following formulas are recommended to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

  ```
  SEMMSL = 13
  ```

  13 is what has been found to be the best for both Progress and MySQL.

  ```
  SEMMNS = SEMMSL * number of db servers to be run on the system
  ```

  Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

  ```
  SEMMNU = SEMMNS
  ```

  Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

2.20.5.10 SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes

Use the latest production release of MySQL. Should you choose to use an older release of MySQL on UnixWare 7.1.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

We have been able to compile MySQL with the following `configure` command on UnixWare 7.1.x:

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db=./bdb \
    --with-innodb --with-openssl --with-extra-charsets=complex
```

If you want to use `gcc`, you must use `gcc` 2.95.3 or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

The version of Berkeley DB that comes with either UnixWare 7.1.4 or OpenServer 6.0.0 is not used when building MySQL. MySQL instead uses its own version of Berkeley DB. The `configure` command needs to build both a static and a dynamic library in *`src_directory`*`/bdb/build_unix/`, but it does not with MySQL's own `BDB` version. The workaround is as follows.

1. Configure as normal for MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. `cd bdb/build_unix/`
3. `cp -p Makefile Makefile.sav`
4. Use same options and run `../dist/configure`.
5. Run `gmake`.
6. `cp -p Makefile.sav Makefile`
7. Change to top source directory and run `gmake`.

This enables both the shared and dynamic libraries to be made and work.

SCO provides operating system patches at ftp://ftp.sco.com/pub/unixware7 for UnixWare 7.1.1, ftp://ftp.sco.com/pub/unixware7/713/ for UnixWare 7.1.3, ftp://ftp.sco.com/pub/unixware7/714/ for UnixWare 7.1.4, and ftp://ftp.sco.com/pub/openunix8 for OpenUNIX 8.0.0.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenUNIX for OpenUNIX and ftp://ftp.sco.com/pub/security/UnixWare for UnixWare.

The UnixWare 7 file size limit is 1 TB with VXFS. Some OS utilities have a limitation of 2GB.

On UnixWare 7.1.4 you do not need to do anything to get large file support, but to enable large file support on prior versions of UnixWare 7.1.x, run `fsadm`.

```
# fsadm -Fvxfs -o largefiles /
# fsadm /         * Note
# ulimit unlimited
# /etc/conf/bin/idtune SFSZLIM 0x7FFFFFFF     ** Note
# /etc/conf/bin/idtune HFSZLIM 0x7FFFFFFF     ** Note
# /etc/conf/bin/idbuild -B

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.
```

Reboot the system using `shutdown`.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
```

To make changes to the kernel, use the `idtune` *`name parameter`* command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. To set the kernel values, execute the following commands as `root`:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
# /etc/conf/bin/idbuild -B && init 6
```

To tune the system, the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `SFNOLIM` and `HFNOLIM` should be at maximum 2048.
- `NPROC` should be set to at least 3000/4000 (depends on number of users).
- The following formulas are recommended to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

  ```
  SEMMSL = 13
  ```

  13 is what has been found to be the best for both Progress and MySQL.

  ```
  SEMMNS = SEMMSL * number of db servers to be run on the system
  ```

  Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

  ```
  SEMMNU = SEMMNS
  ```

  Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

2.20.6 OS/2 Notes

Note

We no longer test builds on OS/2. The notes in this section are provided for your information but may not work on your system.

MySQL uses quite a few open files. Because of this, you should add something like the following to your `CONFIG.SYS` file:

```
SET EMXOPT=-c -n -h1024
```

If you do not do this, you may encounter the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using MySQL with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the Pthreads library. MySQL must be installed on a partition with a type that supports long file names, such as HPFS, FAT32, and so on.

The `INSTALL.CMD` script must be run from OS/2's own `CMD.EXE` and may not work with replacement shells such as `4OS2.EXE`.

The `scripts/mysql-install-db` script has been renamed. It is called `install.cmd` and is a REXX script, which sets up the default MySQL security settings and creates the WorkPlace Shell icons for MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrtl -I../include -I../regex -I.. \
    -o example udf_example.c -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Note

Due to limitations in OS/2, UDF module name stems must not exceed eight characters. Modules are stored in the `/mysql2/udf` directory; the `safe-mysqld.cmd` script puts this directory in the `BEGINLIBPATH` environment variable. When using UDF modules, specified extensions are ignored---it is assumed to be `.udf`. For example, in Unix, the shared module might be named `example.so` and you would load a function from it like this:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';
```

In OS/2, the module would be named `example.udf`, but you would not specify the module extension:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.21 Environment Variables

This section lists environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables. In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See Section 4.2.6, “Using Option Files”.

Variable	Description
`CXX`	The name of your C++ compiler (for running `configure`).
`CC`	The name of your C compiler (for running `configure`).
`CFLAGS`	Flags for your C compiler (for running `configure`).
`CXXFLAGS`	Flags for your C++ compiler (for running `configure`).
`DBI_USER`	The default user name for Perl DBI.
`DBI_TRACE`	Trace options for Perl DBI.
`HOME`	The default path for the `mysql` history file is `$HOME/.mysql_history`.
`LD_RUN_PATH`	Used to specify the location of `libmysqlclient.so`.
`MYSQL_DEBUG`	Debug trace options when debugging.
`MYSQL_GROUP_SUFFIX`	Option group suffix value (like specifying `--defaults-group-suffix`).
`MYSQL_HISTFILE`	The path to the `mysql` history file. If this variable is set, its value overrides the default for `$HOME/.mysql_history`.
`MYSQL_HOME`	The path to the directory in which the server-specific `my.cnf` file resides (as of MySQL 5.0.3).
`MYSQL_HOST`	The default host name used by the `mysql` command-line client.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable	Description
MYSQL_PS1	The command prompt to use in the mysql command-line client.
MYSQL_PWD	The default password when connecting to mysqld. Using this is insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_UNIX_PORT	The default Unix socket file name; used for connections to localhost.
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory in which temporary files are created.
TZ	This should be set to your local time zone. See Section B.5.3.7, “Time Zone Problems”.
UMASK	The user-file creation mode when creating files. See note following table.
UMASK_DIR	The user-directory creation mode when creating directories. See note following table.
USER	The default user name on Windows and NetWare when connecting to mysqld.

For information about the mysql history file, see Section 4.5.1.3, “mysql Logging”.

The default UMASK and UMASK_DIR values are 0660 and 0700, respectively. MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero. For example, setting UMASK=0600 is equivalent to UMASK=384 because 0600 octal is 384 decimal.

The UMASK and UMASK_DIR variables, despite their names, are used as modes, not masks:

- If UMASK is set, mysqld uses ($UMASK | 0600) as the mode for file creation, so that newly created files have a mode in the range from 0600 to 0666 (all values octal).
- If UMASK_DIR is set, mysqld uses ($UMASK_DIR | 0700) as the base mode for directory creation, which then is AND-ed with ~(~$UMASK & 0666), so that newly created directories have a mode in the range from 0700 to 0777 (all values octal). The AND operation may remove read and write permissions from the directory mode, but not execute permissions.

2.22 Perl Installation Notes

The Perl DBI module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI, you must install the DBI module, as well as a DataBase Driver (DBD) module for each type of database server you want to access. For MySQL, this driver is the DBD::mysql module.

Perl, and the DBD::MySQL module for DBI must be installed if you want to run the MySQL benchmark scripts; see Section 8.13.2, “The MySQL Benchmark Suite”. They are also required for the MySQL Cluster ndb_size.pl utility; see Section 17.4.18, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”.

Note

Perl support is not included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState ppm program on Windows. The following sections describe how to do this.

The DBI/DBD interface requires Perl 5.6.0, and 5.6.1 or later is preferred. DBI *does not work* if you have an older version of Perl. You should use DBD::mysql 4.009 or higher. Although earlier versions are available, they do not support the full functionality of MySQL 5.0.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2.22.1 Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. If you install MySQL from RPM files on Linux, be sure to install the developer RPM as well. The client programs are in the client RPM, but client programming support is in the developer RPM.

The files you need for Perl support can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

   ```
   shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd DBI-VERSION
   ```

3. Build the distribution and compile everything:

   ```
   shell> perl Makefile.PL
   shell> make
   shell> make test
   shell> make install
   ```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL. This ensures that the latest versions of the MySQL client libraries are installed correctly.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://servers.digitaldaze.com/extensions/perl/modules.html#modules

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Look under the heading “Installing New Modules that Require Locally Installed Modules.”

2.22.2 Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.
2. Open a console window.
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

   ```
   C:\> set HTTP_proxy=my.proxy.com:3128
   ```

4. Start the PPM program:

   ```
   C:\> C:\perl\bin\ppm.pl
   ```

5. If you have not previously done so, install `DBI`:

   ```
   ppm> install DBI
   ```

6. If this succeeds, run the following command:

   ```
   ppm> install DBD-mysql
   ```

This procedure should work with ActiveState Perl 5.6 or newer.

If you cannot get the procedure to work, you should install the ODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.22.3 Problems Using the Perl DBI/DBD Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Compile the `DBD::mysql` distribution with `perl Makefile.PL -static -config` rather than `perl Makefile.PL`.
- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD::mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the path name of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD::mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the path name of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

You may see the following error from `DBD::mysql` when you run the tests:

```
t/00base............install_driver(mysql) failed:
Can't load '../blib/arch/auto/DBD/mysql/mysql.so' for module DBD::mysql:
../blib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

This means that you need to include the `-lz` compression library on the link line. That can be done by changing the following line in the file `lib/DBD/mysql/Install.pm`:

```
$sysliblist .= " -lm";
```

Change that line to:

```
$sysliblist .= " -lm -lz";
```

After this, you *must* run `make realclean` and then proceed with the installation from the beginning.

If you want to install DBI on SCO, you have to edit the `Makefile` in DBI-*xxx* and each subdirectory. Note that the following assumes `gcc` 2.95.2 or newer:

```
OLD:                                NEW:
CC = cc                             CC = gcc
CCCDLFLAGS = -KPIC -W1,-Bexport     CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport            CCDLFLAGS =

LD = ld                             LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib     LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib    LDFLAGS = -L/usr/local/lib

LD = ld                             LD = gcc -G -fpic
OPTIMISE = -Od                      OPTIMISE = -O1

OLD:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

NEW:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

These changes are necessary because the Perl dynaloader does not load the `DBI` modules if they were compiled with `icc` or `cc`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you want to use the Perl module on a system that does not support dynamic linking (such as SCO), you can generate a static version of Perl that includes `DBI` and `DBD::mysql`. The way this works is that you generate a version of Perl with the `DBI` code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the `DBD` code linked in, and install that.

On SCO, you must have the following environment variables set:

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

Or:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
    /usr/skunk/man:
```

First, create a Perl that includes a statically linked `DBI` module by running these commands in the directory where your `DBI` distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Then, you must install the new Perl. The output of `make perl` indicates the exact `make` command you need to execute to perform the installation. On SCO, this is `make -f Makefile.aperl inst_perl MAP_TARGET=perl`.

Next, use the just-created Perl to create another Perl that also includes a statically linked `DBD::mysql` by running these commands in the directory where your `DBD::mysql` distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Finally, you should install this new Perl. Again, the output of `make perl` indicates the command to use.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 3 Tutorial

Table of Contents

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as Chapter 5, *MySQL Server Administration*.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1 Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you will

This documentation is for an older version. If you're

This documentation is for an older version. If you're

also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

`host` and `user` represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `********` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.0.96-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter SQL statements.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`, it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Chapter 2, *Installing and Upgrading MySQL* that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Section B.5.2, “Common Errors When Using MySQL Programs”.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control+D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2 Entering Queries

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering queries, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple query that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+---------------+--------------+
| VERSION()     | CURRENT_DATE |
+---------------+--------------+
| 5.0.7-beta-Max | 2005-07-11   |
+---------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A query normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a query, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another query.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+---------------+
| VERSION()     |
+---------------+
| 5.0.7-beta-Max |
+---------------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2005-07-11 17:59:36 |
+---------------------+
1 row in set (0.00 sec)
```

A query need not be given all on a single line, so lengthy queries that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2005-07-11   |
+---------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a query that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new query.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
`mysql>`	Ready for new query
`->`	Waiting for next line of multiple-line query
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (“'”)
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (“"”)
`` `> ``	Waiting for next line, waiting for completion of an identifier that began with a backtick (“`”)
`/*>`	Waiting for next line, waiting for completion of a comment that began with `/*`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In the MySQL 5.0 series, the `/*>` prompt was implemented in MySQL 5.0.6.

Multiple-line statements commonly occur by accident when you intend to issue a query on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either "`'`" or "`"`" characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a "`'`" or "`"`" quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the query. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new query.

The `` `> `` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current query.

3.3 Creating and Using a Database

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Once you know how to enter SQL statements, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It is available in both compressed `tar` file and Zip formats at http://dev.mysql.com/doc/.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See Section 13.7.5.11, “SHOW DATABASES Syntax”.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

`USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a statement like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as `ERROR 1044 (42000): Access denied for user 'micah'@'localhost' to database 'menagerie'` when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see Section 6.2, “The MySQL Access Privilege System”.

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this statement:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Important

`menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-pmypassword`, not as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

Note

You can see at any time which database is currently selected using `SELECT DATABASE()`.

3.3.2 Creating a Table

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Creating the database is the easy part, but at this point it is empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you.

Note

Prior to MySQL 5.0.3, the upper limit was 255.) If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+--------------------+
| Tables in menagerie |
+--------------------+
| pet                |
+--------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see Chapter 11, *Data Types*.

3.3.3 Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”, for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4 Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` indicates what you want to see. This can be a list of columns, or `*` to indicate “all columns.” *`which_table`* indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it is present, *`conditions_to_satisfy`* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1 Selecting All Data

The simplest form of `SELECT` retrieves everything from a table:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of `SELECT` is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  The `UPDATE` changes only the record in question and does not require you to reload the table.

3.3.4.2 Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, and so forth. The query result is the same.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born during or after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` and `OR` may be intermixed, although `AND` has higher precedence than `OR`. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
    -> OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
    -> ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5 Date Calculations

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, use the `TIMESTAMPDIFF()` function. Its arguments are the unit in which you want the result expressed, and the two date for which to take the difference. The following query shows, for each pet, the birth date, the current date, and the age in years. An *alias* (`age`) is used to make the final output column label more meaningful.

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
    -> TIMESTAMPDIFF(YEAR,birth,death) AS age
    -> FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with NULL Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to `0` if it is currently `12`:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()` returns a number between `1` and `12`. And `MOD(something,12)` returns a number between `0` and `11`. So the addition has to be after the `MOD()`, otherwise we would go from November (`11`) to January (`1`).

3.3.4.6 Working with NULL Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values.

To test for `NULL`, use the `IS NULL` and `IS NOT NULL` operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

You cannot use arithmetic comparison operators such as `=`, `<`, or `<>` to test for `NULL`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Because the result of any arithmetic comparison with `NULL` is also `NULL`, you cannot obtain any meaningful results from such comparisons.

In MySQL, `0` or `NULL` means false and anything else means true. The default truth value from a boolean operation is `1`.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death <> NULL`.

Two `NULL` values are regarded as equal in a `GROUP BY`.

When doing an `ORDER BY`, `NULL` values are presented first if you do `ORDER BY ... ASC` and last if you do `ORDER BY ... DESC`.

A common error when working with `NULL` is to assume that it is not possible to insert a zero or an empty string into a column defined as `NOT NULL`, but this is not the case. These are in fact values, whereas `NULL` means “not having a value.” You can test this easily enough by using `IS [NOT] NULL` as shown:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+--------------+-----------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+--------------+-----------+----------------+
|         0 |            1 |          0 |              1 |
+-----------+--------------+-----------+----------------+
```

Thus it is entirely possible to insert a zero or empty string into a `NOT NULL` column, as these are in fact `NOT NULL`. See Section B.5.4.3, “Problems with NULL Values”.

3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep`, and `sed`.

SQL pattern matching enables you to use “_” to match any single character and “%” to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. You do not use `=` or `<>` when you use SQL patterns; use the `LIKE` or `NOT LIKE` comparison operators instead.

To find names beginning with “b”:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with “fy”:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a “w”:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the “_” pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP` and `NOT REGEXP` operators (or `RLIKE` and `NOT RLIKE`, which are synonyms).

The following list describes some characteristics of extended regular expressions:

- "`.`" matches any single character.
- A character class "`[...]`" matches any character within the brackets. For example, "`[abc]`" matches "`a`", "`b`", or "`c`". To name a range of characters, use a dash. "`[a-z]`" matches any letter, whereas "`[0-9]`" matches any digit.
- "`*`" matches zero or more instances of the thing preceding it. For example, "`x*`" matches any number of "`x`" characters, "`[0-9]*`" matches any number of digits, and "`.*`" matches any number of anything.
- A `REGEXP` pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a `LIKE` pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use "`^`" at the beginning or "`$`" at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` queries shown previously are rewritten here to use `REGEXP`.

To find names beginning with "`b`", use "`^`" to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

If you really want to force a `REGEXP` comparison to be case sensitive, use the `BINARY` keyword to make one of the strings a binary string. This query matches only lowercase "`b`" at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with "`fy`", use "`$`" to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a "`w`", use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use “`^`” and “`$`” to match the beginning and end of the name, and five instances of “`.`” in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the `{`*n*`}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

Section 12.5.2, “Regular Expressions”, provides more information about the syntax for regular expressions.

3.3.4.8 Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Harold |        2 |
+--------+----------+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = 'dog' OR species = 'cat'
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

If you name columns to select in addition to the `COUNT()` value, a `GROUP BY` clause should be present that names those same columns. Otherwise, the following occurs:

- If the `ONLY_FULL_GROUP_BY` SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

- If `ONLY_FULL_GROUP_BY` is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is indeterminate. The server is free to select the value from any row:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Harold |        8 |
+--------+----------+
1 row in set (0.00 sec)
```

See also Section 12.16.3, “MySQL Handling of GROUP BY”.

3.3.4.9 Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
    -> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
    -> remark
    -> FROM pet INNER JOIN event
    ->   ON pet.name = event.name
    -> WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses `ON` clause to match up records in the two tables based on the `name` values.

  The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the `ON` clause fails.

- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1 INNER JOIN pet AS p2
    ->   ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you have not yet selected any database, the result is `NULL`.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this statement:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_menagerie |
+---------------------+
| event               |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| pet                 |
+---------------------+
```

The name of the column in the output produced by this statement is always `Tables_in_db_name`, where *db_name* is the name of the database. See Section 13.7.5.34, “SHOW TABLES Syntax”, for more information.

If you want to find out about the structure of a table, the `DESCRIBE` statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns: If a column was created with the `AUTO_INCREMENT` option, the value will be `auto_increment` rather than empty.

`DESC` is a short form of `DESCRIBE`. See Section 13.8.1, “DESCRIBE Syntax”, for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See Section 13.7.5.9, “SHOW CREATE TABLE Syntax”.

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them. See Section 13.7.5.18, “SHOW INDEX Syntax”, for more about this statement.

3.5 Using mysql in Batch Mode

In the previous sections, you used `mysql` interactively to enter statements and view the results. You can also run `mysql` in batch mode. To do this, put the statements you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line statements or multiple-statement sequences. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

```
shell> mysql < batch-file | more
```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so that they can also run the statements.
- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the statements that are executed, use `mysql -v`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See Section 4.5.1.5, “Executing SQL Statements from a Text File”, for more information.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

3.6 Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database named `test`).

You can create and populate the example table with these statements:

```
CREATE TABLE shop (
    article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
    price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1 The Maximum Value for a Column

“What is the highest item number?”

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

This documentation is for an older version. If you're

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Other solutions are to use a `LEFT JOIN` or to sort all rows descending by price and get only the first row using the MySQL-specific `LIMIT` clause:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);

+---------+--------+-------+
| article | dealer | price |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The preceding example uses a correlated subquery, which can be inefficient (see Section 13.2.9.7, “Correlated Subqueries”). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause or a `LEFT JOIN`.

Uncorrelated subquery:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price;
```

`LEFT JOIN`:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and the `s2` rows values will be `NULL`. See Section 13.2.8.2, “JOIN Syntax”.

3.6.5 Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See Section 9.4, “User-Defined Variables”.)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Note

It is also possible to store the name of a database object such as a table or a column in a user variable and then to use this variable in an SQL statement; however, this requires the use of a prepared statement. See Section 13.5, “SQL Syntax for Prepared Statements”, for more information.

3.6.6 Using Foreign Keys

In MySQL, `InnoDB` tables support checking of foreign key constraints. See Section 14.2, “The InnoDB Storage Engine”, and Section 1.8.2.4, “Foreign Key Differences”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A foreign key constraint is not required merely to join two tables. For storage engines other than `InnoDB`, it is possible when defining a column to use a `REFERENCES tbl_name(col_name)` clause, which has no actual effect, and *serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table*. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of `CHECK` to make sure that *`col_name`* actually exists in *`tbl_name`* (or even that *`tbl_name`* itself exists).
- MySQL does not perform any sort of action on *`tbl_name`* such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you can write an `ON DELETE` or `ON UPDATE` clause as part of the `REFERENCES` clause, it is also ignored.)
- This syntax creates a *column*; it does **not** create any sort of index or key.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p INNER JOIN shirt s
   ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

When used in this fashion, the `REFERENCES` clause is not displayed in the output of `SHOW CREATE TABLE` or `DESCRIBE`:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of `REFERENCES` in this way as a comment or “reminder” in a column definition works with both `MyISAM` and `BerkeleyDB` tables.

3.6.7 Searching on Two Keys

An `OR` using a single key is well optimized, as is the handling of `AND`.

The one tricky case is that of searching on two different keys combined with `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

This case is optimized from MySQL 5.0.0. See Section 8.2.1.4, “Index Merge Optimization”.

You can also solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 13.2.8.3, “UNION Syntax”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    FROM test_table WHERE field2_index = '1';
```

3.6.8 Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
            day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
           (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9 Using AUTO_INCREMENT

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
) ENGINE=MyISAM;

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

No value was specified for the AUTO_INCREMENT column, so MySQL assigned sequence numbers automatically. You can also explicitly assign 0 to the column to generate sequence numbers. If the column is declared NOT NULL, it is also possible to assign NULL to the column to generate sequence numbers. When you insert any other value into a AUTO_INCREMENT column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the largest column value.

You can retrieve the most recent automatically generated AUTO_INCREMENT value with the LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use a large enough integer data type for the AUTO_INCREMENT column to hold the maximum sequence value you will need. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. For example, if you use TINYINT, the maximum permissible sequence number is 127. For TINYINT UNSIGNED, the maximum is 255.

Note

For a multiple-row insert, LAST_INSERT_ID() and mysql_insert_id() actually return the AUTO_INCREMENT key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

For MyISAM and BDB tables you can specify AUTO_INCREMENT on a secondary column in a multiple-column index. In this case, the generated value for the AUTO_INCREMENT column is calculated as MAX(*auto_increment_column*) + 1 WHERE prefix=*given-prefix*. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
    ('mammal','dog'),('mammal','cat'),
    ('bird','penguin'),('fish','lax'),('mammal','whale'),
    ('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

In this case (when the AUTO_INCREMENT column is part of a multiple-column index), AUTO_INCREMENT values are reused if you delete the row with the biggest AUTO_INCREMENT value in any group. This happens even for MyISAM tables, for which AUTO_INCREMENT values normally are not reused.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the AUTO_INCREMENT column is part of multiple indexes, MySQL will generate sequence values using the index that begins with the AUTO_INCREMENT column, if there is one. For example, if the animals table contained indexes PRIMARY KEY (grp, id) and INDEX (id), MySQL would ignore the PRIMARY KEY for generating sequence values. As a result, the table would contain a single sequence, not a sequence per grp value.

To start with an AUTO_INCREMENT value other than 1, you can set that value with CREATE TABLE or ALTER TABLE, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

More information about AUTO_INCREMENT is available here:

- How to assign the AUTO_INCREMENT attribute to a column: Section 13.1.10, “CREATE TABLE Syntax”, and Section 13.1.4, “ALTER TABLE Syntax”.
- How AUTO_INCREMENT behaves depending on the NO_AUTO_VALUE_ON_ZERO SQL mode: Section 5.1.7, “Server SQL Modes”.
- How to use the LAST_INSERT_ID() function to find the row that contains the most recent AUTO_INCREMENT value: Section 12.13, “Information Functions”.
- Setting the AUTO_INCREMENT value to be used: Section 5.1.4, “Server System Variables”.
- AUTO_INCREMENT and replication: Section 16.4.1.1, “Replication and AUTO_INCREMENT”.
- Server-system variables related to AUTO_INCREMENT (auto_increment_increment and auto_increment_offset) that can be used for replication: Section 5.1.4, “Server System Variables”.
- AUTO_INCREMENT and InnoDB tables: Section 14.2.3.3, “AUTO_INCREMENT Handling in InnoDB”.

3.7 Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the LogFormat line writes to the log file.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 4 MySQL Programs

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1 Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of MySQL Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. Chapter 17, *MySQL Cluster*, describes programs specific to MySQL Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing and Upgrading MySQL*, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 4.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See Section 4.3.1, “`mysqld` — The MySQL Server”.

- `mysqld_safe`

  A server startup script. `mysqld_safe` attempts to start `mysqld`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

- `mysql.server`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

- `mysqld_multi`

  A server startup script that can start or stop multiple servers installed on the system. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”. As of MySQL 5.0.3 (Unix-like systems) or 5.0.13 (Windows), an alternative to `mysqld_multi` is `mysqlmanager`, the MySQL Instance Manager. See Section 4.6.10, “`mysqlmanager` — The MySQL Instance Manager”.

There are several programs that perform setup operations during MySQL installation or upgrading:

- `comp_err`

  This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”.

- `make_binary_distribution`

  This program makes a binary release of a compiled MySQL. This could be sent by FTP to `/pub/mysql/upload/` on `ftp.mysql.com` for the convenience of other MySQL users.

- `make_win_bin_dist`

  This program is used on Windows. It packages a MySQL distribution for installation after the source distribution has been built. See Section 4.4.2, “`make_win_bin_dist` — Package MySQL Distribution as Zip Archive”.

- `mysql_fix_privilege_tables`

  This program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL. See Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

  Note: As of MySQL 5.0.19, this program has been superseded by `mysql_upgrade` and should no longer be used.

- `mysql_install_db`

  This program initializes the MySQL data directory, creates the `mysql` database, and initializes its grant tables with default privileges. It is usually executed only once, when first installing MySQL on a system. See Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”, and Section 2.18, “Postinstallation Setup and Testing”.

- `mysql_secure_installation`

  This program enables you to improve the security of your MySQL installation. See Section 4.4.7, “`mysql_secure_installation` — Improve MySQL Installation Security”.

- `mysql_tzinfo_to_sql`

  This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). See Section 4.4.8, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”.

- `mysql_upgrade`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This program is used after a MySQL upgrade operation. It checks tables for incompatibilities and repairs them if necessary, and updates the grant tables with any changes that have been made in newer versions of MySQL. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

- `make_win_src_distribution`

  This program is used on Unix or Unix-like systems to create a MySQL source distribution that can be compiled on Windows. See Section 2.10.8.5, “Creating a Windows Source Package from the Bazaar Repository”, and Section 4.4.3, “`make_win_src_distribution` — Create Source Distribution for Windows”.

MySQL client programs:

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”.

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”.

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL, text, or XML. See Section 4.5.4, “`mysqldump` — A Database Backup Program”.

- `mysqlimport`

  A client that imports text files into their respective tables using `LOAD DATA INFILE`. See Section 4.5.5, “`mysqlimport` — A Data Import Program”.

- `mysqlshow`

  A client that displays information about databases, tables, columns, and indexes. See Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

MySQL administrative and utility programs:

- `innochecksum`

  An offline `InnoDB` offline file checksum utility. See Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”.

- `myisam_ftdump`

  A utility that displays information about full-text indexes in `MyISAM` tables. See Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”.

- `myisamchk`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A utility to describe, check, optimize, and repair `MyISAM` tables. See Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

- `myisamlog`

  A utility that processes the contents of a `MyISAM` log file. See Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”.

- `myisampack`

  A utility that compresses `MyISAM` tables to produce smaller read-only tables. See Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

- `mysqlaccess`

  A script that checks the access privileges for a host name, user name, and database combination. See Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”.

- `mysqlbinlog`

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

- `mysqldumpslow`

  A utility to read and summarize the contents of a slow query log. See Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”.

- `mysqlhotcopy`

  A utility that quickly makes backups of `MyISAM` tables while the server is running. See Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”.

- `mysqlmanager`

  The MySQL Instance Manager, a program for monitoring and managing MySQL servers. See Section 4.6.10, “`mysqlmanager` — The MySQL Instance Manager”.

  Important

  MySQL Instance Manager has been deprecated and is removed in MySQL 5.5.

- `mysql_convert_table_format`

  A utility that converts tables in a database to use a given storage engine. See Section 4.6.11, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”.

- `mysql_explain_log`

  A utility that analyzes queries in the MySQL query log using `EXPLAIN` See Section 4.6.12, “`mysql_explain_log` — Use EXPLAIN on Statements in Query Log”.

- `mysql_find_rows`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A utility that reads files containing SQL statements (such as update logs) and extracts statements that match a given regular expression. See Section 4.6.13, “`mysql_find_rows` — Extract SQL Statements from Files”.

- `mysql_fix_extensions`

  A utility that converts the extensions for `MyISAM` table files to lowercase. This can be useful after transferring the files from a system with case-insensitive file names to a system with case-sensitive file names. See Section 4.6.14, “`mysql_fix_extensions` — Normalize Table File Name Extensions”.

- `mysql_setpermission`

  A utility for interactively setting permissions in the MySQL grant tables. See Section 4.6.15, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”.

- `mysql_tableinfo`

  A utility that generates database metadata. Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”.

- `mysql_waitpid`

  A utility that kills the process with a given process ID. See Section 4.6.17, “`mysql_waitpid` — Kill Process and Wait for Its Termination”.

- `mysql_zap`

  A utility that kills processes that match a pattern. See Section 4.6.18, “`mysql_zap` — Kill Processes That Match a Pattern”.

MySQL program-development utilities:

- `msql2mysql`

  A shell script that converts `mSQL` programs to MySQL. It doesn't handle every case, but it gives a good start when converting. See Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”.

- `mysql_config`

  A shell script that produces the option values needed when compiling MySQL programs. See Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”.

- `my_print_defaults`

  A utility that shows which options are present in option groups of option files. See Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”.

- `resolve_stack_dump`

  A utility program that resolves a numeric stack trace dump to symbols. See Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”.

Miscellaneous utilities:

- `perror`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A utility that displays the meaning of system or MySQL error codes. See Section 4.8.1, “`perror` — Explain Error Codes”.

- `replace`

  A utility program that performs string replacement in the input text. See Section 4.8.2, “`replace` — A String-Replacement Utility”.

- `resolveip`

  A utility program that resolves a host name to an IP address or vice versa. See Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”.

Oracle Corporation also provides the MySQL Workbench GUI tool, which is used to administer MySQL servers and databases, to create, execute, and evaluate queries, and to migrate schemas and data from other relational database management systems for use with MySQL. Additional GUI tools include MySQL Notifier and MySQL for Excel.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
`MYSQL_UNIX_PORT`	The default Unix socket file; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default port number; used for TCP/IP connections
`MYSQL_PWD`	The default password
`MYSQL_DEBUG`	Debug trace options when debugging
`TMPDIR`	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see Section 2.21, “Environment Variables”.

Use of `MYSQL_PWD` is insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

4.2 Using MySQL Programs

4.2.1 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “`shell>`” represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh` or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (“`-`”, “`--`”) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 4.2.3, “Specifying Program Options”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first nonoption argument as a database name, so the command `mysql --user=root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are `--port` (or `-P`) to specify a TCP/IP port number and `--socket` (or `-S`) to specify a Unix socket file on Unix (or named pipe name on Windows). For more information on options that specify connection options, see Section 4.2.2, “Connecting to the MySQL Server”.

You may find it necessary to invoke MySQL programs using the path name to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the path name of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you can run the program by invoking it as `mysql`, and it is not necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in Section 4.2.10, “Setting Environment Variables”.) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2 Connecting to the MySQL Server

This section describes how to establish a connection to the MySQL server. For additional information if you are unable to connect, see Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”.

For a client program to be able to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the user name and password of your MySQL account. Each connection parameter has a default value, but you can override them as necessary using program options specified either on the command line or in an option file.

The examples here use the `mysql` client program, but the principles apply to other clients such as `mysqldump`, `mysqladmin`, or `mysqlshow`.

This command invokes `mysql` without specifying any connection parameters explicitly:

```
shell> mysql
```

Because there are no parameter options, the default values apply:

- The default host name is `localhost`. On Unix, this has a special meaning, as described later.
- The default user name is `ODBC` on Windows or your Unix login name on Unix.
- No password is sent if neither `-p` nor `--password` is given.
- For `mysql`, the first nonoption argument is taken as the name of the default database. If there is no such option, `mysql` does not select a default database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To specify the host name and user name explicitly, as well as a password, supply appropriate options on the command line:

```
shell> mysql --host=localhost --user=myname --password=mypass mydb
shell> mysql -h localhost -u myname -pmypass mydb
```

For password options, the password value is optional:

- If you use a `-p` or `--password` option and specify the password value, there must be *no space* between `-p` or `--password=` and the password following it.
- If you use a `-p` or `--password` option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Other users on your system may be able to see a password specified on the command line by executing a command such as `ps auxw`. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

As just mentioned, including the password value on the command line can be a security risk. To avoid this problem, specify the `--password` or `-p` option without any following password value:

```
shell> mysql --host=localhost --user=myname --password mydb
shell> mysql -h localhost -u myname -p mydb
```

When the password option has no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `mydb` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL does not have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

On Unix, MySQL programs treat the host name `localhost` specially, in a way that is likely different from what you expect compared to other network-based programs. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file. This occurs even if a `--port` or `-P` option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use `--host` or `-h` to specify a host name value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=TCP` option. For example:

```
shell> mysql --host=127.0.0.1
shell> mysql --protocol=TCP
```

The `--protocol` option enables you to establish a particular type of connection even when the other options would normally default to some other protocol.

On Windows, you can force a MySQL client to use a named-pipe connection by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. If named-pipe connections are not enabled, an error occurs. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers always use TCP/IP. This command connects to the server running on `remote.example.com` using the default port number (3306):

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql --host=remote.example.com
```

To specify a port number explicitly, use the `--port` or `-P` option:

```
shell> mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to `localhost` on Unix will use a socket file by default. You will need to force a TCP/IP connection as already described or any option that specifies a port number will be ignored.

For this command, the program uses a socket file on Unix and the `--port` option is ignored:

```
shell> mysql --port=13306 --host=localhost
```

To cause the port number to be used, invoke the program in either of these ways:

```
shell> mysql --port=13306 --host=127.0.0.1
shell> mysql --port=13306 --protocol=TCP
```

The following list summarizes the options that can be used to control how client programs connect to the server:

- `--host=`*`host_name`*, `-h` *`host_name`*

  The host where the server is running. The default value is `localhost`.

- `--password[=`*`pass_val`*`]`, `-p[`*`pass_val`*`]`

  The password of the MySQL account. As described earlier, the password value is optional, but if given, there must be *no space* between `-p` or `--password=` and the password following it. The default is to send no password.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. The server must be started with the `--enable-named-pipe` option to enable named-pipe connections.

- `--port=`*`port_num`*, `-P` *`port_num`*

  The port number to use for the connection, for connections made using TCP/IP. The default port number is 3306.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  This option explicitly specifies a protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For example, connections on Unix to `localhost` are made using a Unix socket file by default:

  ```
  shell> mysql --host=localhost
  ```

  To force a TCP/IP connection to be used instead, specify a `--protocol` option:

  ```
  shell> mysql --host=localhost --protocol=TCP
  ```

  The following table shows the permissible `--protocol` option values and indicates the platforms on which each value may be used. The values are not case sensitive.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`--protocol` Value	Connection Protocol	Permissible Operating Systems
`TCP`	TCP/IP connection to local or remote server	All
`SOCKET`	Unix socket file connection to local server	Unix only
`PIPE`	Named-pipe connection to local or remote server	Windows only
`MEMORY`	Shared-memory connection to local server	Windows only

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

- `--socket=`*`file_name`*, `-S` *`file_name`*

  On Unix, the name of the Unix socket file to use, for connections made using a named pipe to a local server. The default Unix socket file name is `/tmp/mysql.sock`.

  On Windows, the name of the named pipe to use, for connections to a local server. The default Windows pipe name is `MySQL`. The pipe name is not case sensitive.

  The server must be started with the `--enable-named-pipe` option to enable named-pipe connections.

- `--ssl*`

  Options that begin with `--ssl` are used for establishing a secure connection to the server using SSL, if the server is configured with SSL support. For details, see Section 6.3.6.5, “Command Options for Secure Connections”.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account you want to use. The default user name is `ODBC` on Windows or your Unix login name on Unix.

It is possible to specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

```
[client]
host=host_name
user=user_name
password=your_pass
```

  Section 4.2.6, “Using Option Files”, discusses option files further.

- You can specify some connection parameters using environment variables. The host can be specified for `mysql` using `MYSQL_HOST`. The MySQL user name can be specified using `USER` (this is for Windows and NetWare only). The password can be specified using `MYSQL_PWD`, although this is insecure; see Section 6.1.2.1, “End-User Guidelines for Password Security”. For a list of variables, see Section 2.21, “Environment Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4.2.3 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see Section 4.2.10, “Setting Environment Variables”). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose, but Section 5.5.3, “Running Multiple MySQL Instances on Unix”, discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
shell> mysql -h example.com -h localhost
```

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in “no column names” mode:

```
shell> mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. This means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

An option can be specified by writing it in full or as any unambiguous prefix. For example, the `--compress` option can be given to `mysqldump` as `--compr`, but not as `--comp` because the latter is ambiguous:

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

Be aware that the use of option prefixes can cause problems in the event that new options are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

4.2.4 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.

- For a long option that takes a value, separate the option name and the value by an "=" sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=pass_val` or as `--password`. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

  ```
  shell> mysql -ptest
  shell> mysql -p test
  ```

  The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

- Within option names, dash ("-") and underscore ("_") may be used interchangeably. For example, `--skip-grant-tables` and `--skip_grant_tables` are equivalent. (However, the leading dashes cannot be given as underscores.)

- For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

  ```
  mysql> mysqladmin --count=1K --sleep=10 ping
  ```

Option values that contain spaces must be quoted when given on the command line. For example, the `--execute` (or `-e`) option can be used with `mysql` to pass SQL statements to the server. When this option is used, `mysql` executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example, you can use the following command to obtain a list of user accounts:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
Enter password: ******
+------+-----------+
| User | Host      |
+------+-----------+
|      | gigan     |
| root | gigan     |
|      | localhost |
| jon  | localhost |
| root | localhost |
+------+-----------+
shell>
```

Note

The long form (`--execute`) is followed by an equals sign (`=`).

If you wish to use quoted values within a statement, you will either need to escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

Multiple SQL statements may be passed in the option value on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.97-debug-log |
+-----------------+
+---------------------+
| NOW()               |
+---------------------+
| 2015-11-05 20:03:51 |
+---------------------+
```

The `--execute` or `-e` option may also be used to pass commands in an analogous fashion to the `ndb_mgm` management client for MySQL Cluster. See Section 17.2.5, “Safe Shutdown and Restart of MySQL Cluster”, for an example.

4.2.5 Program Option Modifiers

Some options are “boolean” and control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--loose-no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that

This documentation is for an older version. If you're
This documentation is for an older version. If you're

recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

The `--maximum` prefix is available for `mysqld` only and permits a limit to be placed on how large client programs can set session system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-max_heap_table_size=32M` prevents any client from making the heap table size limit larger than 32M.

The `--maximum` prefix is intended for use with system variables that have a session value. If applied to a system variable that has only a global value, an error occurs. For example, with `--maximum-query_cache_size=4M`, the server produces this error:

```
Maximum value of 'query_cache_size' cannot be set
```

4.2.6 Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program. For the MySQL server, MySQL provides a number of preconfigured option files.

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note

Option files used with MySQL Cluster programs are covered in Section 17.3, “MySQL Cluster Configuration”.

On Windows, MySQL programs read startup options from the following files, in the specified order (top files are read first, later files take precedence).

File Name	Purpose
`%WINDIR%\my.ini`, `%WINDIR%\my.cnf`	Global options
`C:\my.ini`, `C:\my.cnf`	Global options
`INSTALLDIR``\my.ini`, *`INSTALLDIR`*`\my.cnf`	Global options
`defaults-extra-file`	The file specified with `--defaults-extra-file=`*`file_name`*, if any

In table items, `%WINDIR%` represents the location of your Windows directory. This is commonly `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

`INSTALLDIR` represents the MySQL installation directory. This is typically `C:\`*`PROGRAMDIR`*`\MySQL\MySQL 5.0 Server` where *`PROGRAMDIR`* represents the programs directory (usually `Program Files` on English-language versions of Windows), when MySQL 5.0 has been installed using the installation and configuration wizards. See Section 2.10.3.1, “Starting the MySQL Server Instance Configuration Wizard”.

On Unix, Linux and OS X, MySQL programs read startup options from the following files, in the specified order (top files are read first, later files take precedence).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

File Name	Purpose
`/etc/my.cnf`	Global options
`SYSCONFDIR``/my.cnf`	Global options
`$MYSQL_HOME/my.cnf`	Server-specific options
`defaults-extra-file`	The file specified with `--defaults-extra-file=`*`file_name`*, if any
`~/.my.cnf`	User-specific options

In table items, `~` represents the current user's home directory (the value of `$HOME`).

`SYSCONFDIR` represents the directory specified with the `--sysconfdir` option to `configure` when MySQL was built. By default, this is the `etc` directory located under the compiled-in installation directory. This location is used as of MySQL 5.0.21. (From 5.0.21 to 5.0.53, it was read last, after `~/.my.cnf`.)

`MYSQL_HOME` is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides. (This was *`DATADIR`* prior to MySQL version 5.0.3.)

If `MYSQL_HOME` is not set and you start the server using the `mysqld_safe` program, `mysqld_safe` attempts to set `MYSQL_HOME` as follows:

- Let *`BASEDIR`* and *`DATADIR`* represent the path names of the MySQL base directory and data directory, respectively.
- If there is a `my.cnf` file in *`DATADIR`* but not in *`BASEDIR`*, `mysqld_safe` sets `MYSQL_HOME` to *`DATADIR`*.
- Otherwise, if `MYSQL_HOME` is not set and there is no `my.cnf` file in *`DATADIR`*, `mysqld_safe` sets `MYSQL_HOME` to *`BASEDIR`*.

In MySQL 5.0, use of *`DATADIR`* as the location for `my.cnf` is deprecated.

Typically, *`DATADIR`* is `/usr/local/mysql/data` for a binary installation or `/usr/local/var` for a source installation. This is the data directory location that was specified at configuration time, not the one specified with the `--datadir` option when `mysqld` starts. Use of `--datadir` at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option.

The syntax for specifying options in an option file is similar to command-line syntax (see Section 4.2.4, "Using Options on the Command Line"). However, in an option file, you omit the leading two dashes from the option name and you specify only one option per line. For example, `--quick` and `--host=localhost` on the command line should be specified as `quick` and `host=localhost` on separate lines in an option file. To specify an option of the form `--loose-`*`opt_name`* in an option file, write it as `loose-`*`opt_name`*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- `#comment`, `;comment`

  Comment lines start with "#" or ";". A "#" comment can start in the middle of a line as well.

- `[group]`

  group is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case sensitive.

- *opt_name*

  This is equivalent to `--opt_name` on the command line.

- *opt_name*`=`*value*

  This is equivalent to `--opt_name=value` on the command line. In an option file, you can have spaces around the "=" character, something that is not true on the command line. You can optionally enclose the value within single quotation marks or double quotation marks, which is useful if the value contains a "#" comment character.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences "`\b`", "`\t`", "`\n`", "`\r`", "`\\`", and "`\s`" in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. The escaping rules in option files are:

- If a backslash is followed by a valid escape sequence character, the sequence is converted to the character represented by the sequence. For example, "`\s`" is converted to a space.

- If a backslash is not followed by a valid escape sequence character, it remains unchanged. For example, "`\S`" is retained as is.

The preceding rules mean that a literal backslash can be given as "`\\`", or as "`\`" if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if "*x*" is not a valid escape sequence character, "`\`*x*" becomes "*x*" rather than "`\`*x*". See Section 9.1.1, "String Literals".

The escaping rules for option file values are especially pertinent for Windows path names, which use "`\`" as a path name separator. A separator in a Windows path name must be written as "`\\`" if it is followed by an escape sequence character. It can be written as "`\\`" or "`\`" if it is not. Alternatively, "`/`" may be used in Windows path names and will be treated as "`\`". Suppose that you want to specify a base directory of `C:\Program Files\MySQL\MySQL Server 5.0` in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 5.0"
basedir="C:\\Program Files\\MySQL\\MySQL Server 5.0"
basedir="C:/Program Files/MySQL/MySQL Server 5.0"
basedir=C:\\Program\sFiles\\MySQL\\MySQL\sServer\s5.0
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `[client]` option group is read by all client programs (but *not* by `mysqld`). This enables you to specify options that apply to all clients. For example, `[client]` is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses *`var_name=value`* syntax for the lines that set the `key_buffer_size` and `max_allowed_packet` variables.

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

If you want to create option groups that should be read by `mysqld` servers from a specific MySQL release series only, you can do this by using groups with names of `[mysqld-4.1]`, `[mysqld-5.0]`, and so forth. The following group indicates that the `sql_mode` setting should be used only by MySQL servers with 5.0.x version numbers:

```
[mysqld-5.0]
sql_mode=TRADITIONAL
```

Beginning with MySQL 5.0.4, it is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

There is no guarantee about the order in which the option files in the directory will be read.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

Any files to be found and included using the `!includedir` directive on Unix operating systems *must* have file names ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

4.2.7 Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. Because these options affect option-file handling, they must be given on the command line and not in an option file. To work properly, each of these options must be given before other options, with these exceptions:

- `--print-defaults` may be used immediately after `--defaults-file` or `--defaults-extra-file`.
- On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. See Section 2.10.4.7, “Starting MySQL as a Windows Service”.

When specifying file names, avoid the use of the “~” shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. (For information about the order in which option files are used, see Section 4.2.6, “Using Option Files”.) As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=`*`file_name`*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=`*`str`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups. This option was added in MySQL 5.0.10.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

4.2.8 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the `SET` statement. See Section 13.7.4, “SET Syntax”, and Section 5.1.5, “Using System Variables”.

Most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

A variable can be specified by writing it in full or as any unambiguous prefix. For example, the `max_allowed_packet` variable can be set for `mysql` as `--max_a`, but not as `--max` because the latter is ambiguous:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql --max=1000000
mysql: ambiguous option '--max=1000000' (max_allowed_packet, max_join_size)
```

Be aware that the use of variable prefixes can cause problems in the event that new variables are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Note

Before MySQL 4.0.2, the only syntax for setting program variables was `--set-variable=`*`option=value`* (or `set-variable=`*`option=value`* in option files). Underscores cannot be given as dashes, and the variable name must be specified in full. This syntax still is recognized, but is now deprecated and is removed in MySQL 5.5.

4.2.9 Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (`=`) sign, like this:

```
shell> mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equal sign is not required, and so the following is also valid:

```
shell> mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.0.96 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+----------------+
| jon@%          |
+----------------+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` on the command line is a host name, `--host --user` is interpreted as `--host=--user`, and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equal sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` option has the default value *`host_name`*`.err`, where *`host_name`* is the name of the host on which MySQL is running. Assume that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

After shutting down the server, restart it as follows:

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

The result is the same, since `--log-error` is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                    ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> tail /usr/local/mysql/var/tonfisk.err
080111 22:53:32  InnoDB: Started; log sequence number 0 46409
/usr/local/mysql/libexec/mysqld: Too many arguments (first extra is 'my-errors').
Use --verbose --help to get a list of available options
080111 22:53:32 [ERROR] Aborting

080111 22:53:32  InnoDB: Starting shutdown...
080111 22:53:34  InnoDB: Shutdown completed; log sequence number 0 46409
080111 22:53:34 [Note] /usr/local/mysql/libexec/mysqld: Shutdown complete

080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended
```

Because the `--log-error` option supplies a default value, you must use an equal sign to assign a different value to it, as shown here:

```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host --user` or `--host=--user`, with the result shown here:

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equal sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]

user jon
```

Trying to start `mysql` in this case causes a different error:

```
shell> mysql
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equals sign:

```
[mysql]

user=jon
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Now the login attempt succeeds:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.0.96 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equals sign is not required:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.96 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@tonfisk   |
+---------------+
1 row in set (0.00 sec)
```

4.2.10 Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. Section 2.21, “Environment Variables”, lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows or NetWare, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For `csh` and `tcsh`, use `setenv` to make the shell variable available to the environment:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are `.bashrc` or `.bash_profile` for `bash`, or `.tcshrc` for `tcsh`.

Suppose that your MySQL programs are installed in `/usr/local/mysql/bin` and that you want to make it easy to invoke these programs. To do this, set the value of the `PATH` environment variable to include that directory. For example, if your shell is `bash`, add the following line to your `.bashrc` file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

`bash` uses different startup files for login and nonlogin shells, so you might want to add the setting to `.bashrc` for login shells and to `.bash_profile` for nonlogin shells to make sure that `PATH` is set regardless.

If your shell is `tcsh`, add the following line to your `.tcshrc` file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3 MySQL Server and Server-Startup Programs

This section describes `mysqld`, the MySQL server, and several programs that are used to start the server.

4.3.1 `mysqld` — The MySQL Server

`mysqld`, also known as MySQL Server, is the main program that does most of the work in a MySQL installation. MySQL Server manages access to the MySQL data directory that contains databases and tables. The data directory is also the default location for other information such as log files and status files.

When MySQL server starts, it listens for network connections from client programs and manages access to databases on behalf of those clients.

The `mysqld` program has many options that can be specified at startup. For a complete list of options, run this command:

```
shell> mysqld --verbose --help
```

MySQL Server also has a set of system variables that affect its operation as it runs. System variables can be set at server startup, and many of them can be changed at runtime to effect dynamic server reconfiguration. MySQL Server also has a set of status variables that provide information about its operation. You can monitor these status variables to access runtime performance characteristics.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For a full description of MySQL Server command options, system variables, and status variables, see Section 5.1, “The MySQL Server”. For information about installing MySQL and setting up the initial configuration, see Chapter 2, *Installing and Upgrading MySQL*.

4.3.2 mysqld_safe — MySQL Server Startup Script

mysqld_safe is the recommended way to start a mysqld server on Unix and NetWare. mysqld_safe adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. NetWare-specific behaviors are listed later in this section.

Note

To preserve backward compatibility with older versions of MySQL, MySQL binary distributions still include safe_mysqld as a symbolic link to mysqld_safe. However, you should not rely on this because it is removed as of MySQL 5.1.

By default, mysqld_safe before MySQL 5.0.27 tries to start an executable named mysqld-max if it exists, and mysqld otherwise. Be aware of the implications of this behavior:

- On Linux, the MySQL-Max RPM relies on this mysqld_safe behavior. The RPM installs an executable named mysqld-max, which causes mysqld_safe to automatically use that executable rather than mysqld from that point on.
- If you install a MySQL-Max distribution that includes a server named mysqld-max, and then upgrade later to a non-Max version of MySQL, mysqld_safe will still attempt to run the old mysqld-max server. If you perform such an upgrade, you should manually remove the old mysqld-max server to ensure that mysqld_safe runs the new mysqld server.

To override the default behavior and specify explicitly the name of the server you want to run, specify a --mysqld or --mysqld-version option to mysqld_safe. You can also use --ledir to indicate the directory where mysqld_safe should look for the server.

Many of the options to mysqld_safe are the same as the options to mysqld. See Section 5.1.3, “Server Command Options”.

Options unknown to mysqld_safe are passed to mysqld if they are specified on the command line, but ignored if they are specified in the [mysqld_safe] group of an option file. See Section 4.2.6, “Using Option Files”.

mysqld_safe reads all options from the [mysqld], [server], and [mysqld_safe] sections in option files. For example, if you specify a [mysqld] section like this, mysqld_safe will find and use the --log-error option:

```
[mysqld]
log-error=error.log
```

For backward compatibility, mysqld_safe also reads [safe_mysqld] sections, but to be current you should rename such sections to [mysqld_safe].

mysqld_safe supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.7, “Command-Line Options that Affect Option-File Handling”.

Table 4.1 mysqld_safe Options

Format	Description	Introduced
--autoclose	On NetWare, mysqld_safe provides a screen presence	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--basedir	Path to MySQL installation directory	
--core-file-size	Size of core file that mysqld should be able to create	
--datadir	Path to data directory	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--help	Display help message and exit	5.0.3
--ledir	Path to directory where server is located	
--log-error	Write error log to named file	
--mysqld	Name of server program to start (in ledir directory)	
--mysqld-version	Suffix for server program name	
--nice	Use nice program to set server scheduling priority	
--no-defaults	Read no option files	
--open-files-limit	Number of files that mysqld should be able to open	
--pid-file	Path name of process ID file	
--port	Port number on which to listen for TCP/IP connections	
--skip-kill-mysqld	Do not try to kill stray mysqld processes	
--socket	Socket file on which to listen for Unix socket connections	
--timezone	Set TZ time zone environment variable to named value	
--user	Run mysqld as user having name user_name or numeric user ID user_id	

- `--help`

  Display a help message and exit. (Added in MySQL 5.0.3)

- `--autoclose`

  (NetWare only) On NetWare, `mysqld_safe` provides a screen presence. When you unload (shut down) the `mysqld_safe` NLM, the screen does not by default go away. Instead, it prompts for user input:

  ```
  *<NLM has terminated; Press any key to close the screen>*
  ```

  If you want NetWare to close the screen automatically instead, use the `--autoclose` option to `mysqld_safe`.

- `--basedir=dir_name`

  The path to the MySQL installation directory.

- `--core-file-size=size`

  The size of the core file that `mysqld` should be able to create. The option value is passed to `ulimit -c`.

- `--datadir=dir_name`

  The path to the data directory.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--defaults-extra-file=file_name`

  The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, the server will exit with an error.

- `--defaults-file=file_name`

  The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.

- `--ledir=dir_name`

  If `mysqld_safe` cannot find the server, use this option to indicate the path name to the directory where the server is located.

- `--log-error=file_name`

  Write the error log to the given file. See Section 5.4.1, “The Error Log”.

- `--mysqld=prog_name`

  The name of the server program (in the `ledir` directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If `mysqld_safe` cannot find the server, use the `--ledir` option to indicate the path name to the directory where the server is located.

- `--mysqld-version=suffix`

  This option is similar to the `--mysqld` option, but you specify only the suffix for the server program name. The base name is assumed to be `mysqld`. For example, if you use `--mysqld-version=debug`, `mysqld_safe` starts the `mysqld-debug` program in the `ledir` directory. If the argument to `--mysqld-version` is empty, `mysqld_safe` uses `mysqld` in the `ledir` directory.

- `--nice=priority`

  Use the `nice` program to set the server's scheduling priority to the given value.

- `--no-defaults`

  Do not read any option files. This must be the first option on the command line if it is used.

- `--open-files-limit=count`

  The number of files that `mysqld` should be able to open. The option value is passed to `ulimit -n`.

  Note

  You must start `mysqld_safe` as `root` for this to function properly.

- `--pid-file=file_name`

  The path name of the process ID file.

- `--port=port_num`

  The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--skip-kill-mysqld`

  Do not try to kill stray `mysqld` processes at startup. This option works only on Linux.

- `--socket=path`

  The Unix socket file that the server should use when listening for local connections.

- `--timezone=timezone`

  Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name|user_id}`

  Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute `mysqld_safe` with the `--defaults-file` or `--defaults-extra-file` option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The `mysqld_safe` script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See Section 2.7, “Installation Layouts”.) `mysqld_safe` expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which `mysqld_safe` is invoked). For binary distributions, `mysqld_safe` looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute `mysqld_safe` from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).
- If the server and databases cannot be found relative to the working directory, `mysqld_safe` attempts to locate them by absolute path names. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because `mysqld_safe` tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run `mysqld_safe` from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If `mysqld_safe` fails, even when invoked from the MySQL installation directory, specify the `--ledir` and `--datadir` options to indicate the directories in which the server and databases are located on your system.

Normally, you should not edit the `mysqld_safe` script. Instead, configure `mysqld_safe` by using command-line options or options in the `[mysqld_safe]` section of a `my.cnf` option file. In rare cases, it

This documentation is for an older version. If you're
This documentation is for an older version. If you're

might be necessary to edit mysqld_safe to get it to start the server properly. However, if you do this, your modified version of mysqld_safe might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

On NetWare, mysqld_safe is a NetWare Loadable Module (NLM) that is ported from the original Unix shell script. It starts the server as follows:

1. Runs a number of system and option checks.
2. Runs a check on MyISAM tables.
3. Provides a screen presence for the MySQL server.
4. Starts mysqld, monitors it, and restarts it if it terminates in error.
5. Sends error messages from mysqld to the *host_name*.err file in the data directory.
6. Sends mysqld_safe screen output to the *host_name*.safe file in the data directory.

4.3.3 mysql.server — MySQL Server Startup Script

MySQL distributions on Unix include a script named mysql.server, which starts the server using mysqld_safe. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the OS X Startup Item for MySQL.

To start or stop the server manually using the mysql.server script, invoke it with start or stop arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before mysql.server starts the server, it changes location to the MySQL installation directory, and then invokes mysqld_safe. To run the server as some specific user, add an appropriate user option to the [mysqld] group of the /etc/my.cnf option file, as shown later in this section. (It is possible that you must edit mysql.server if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to change location into the proper directory before it runs mysqld_safe. If you do this, your modified version of mysql.server may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

mysql.server stop stops the server by sending a signal to it. You can also stop the server manually by executing mysqladmin shutdown.

To start and stop MySQL automatically on your server, you must add start and stop commands to the appropriate places in your /etc/rc* files.

If you use the Linux server RPM package (MySQL-server-*VERSION*.rpm), the mysql.server script is installed in the /etc/init.d directory with the name mysql. You need not install it manually. See Section 2.12, “Installing MySQL on Linux Using RPM Packages”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as mysqld.

If you install MySQL from a source distribution or using a binary distribution format that does not install mysql.server automatically, you can install it manually. The script can be found in the support-files directory under the MySQL installation directory or in a MySQL source tree. Copy it to the /etc/init.d directory with the name mysql, and then make it executable:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Note

Older Red Hat systems use the `/etc/rc.d/init.d` directory rather than `/etc/init.d`. Adjust the preceding commands accordingly. Alternatively, first create `/etc/init.d` as a symbolic link that points to `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use `chkconfig`:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the `mysql` script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their base name matches the `*.sh` shell file name pattern. Any other files or directories present within the directory are silently ignored. In other words, on FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this method, append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

`mysql.server` reads options from the `[mysql.server]` and `[mysqld]` sections of option files. For backward compatibility, it also reads `[mysql_server]` sections, but to be current you should rename such sections to `[mysql.server]`.

You can add options for `mysql.server` in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The `mysql.server` script supports the following options. If specified, they *must* be placed in an option file, not on the command line. `mysql.server` supports only `start` and `stop` as command-line arguments.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 4.2 mysql.server Options

Format	Description	Introduced
--basedir	Path to MySQL installation directory	
--datadir	Path to MySQL data directory	
--pid-file	File in which server should write its process ID	
--service-startup-timeout	How long to wait for server startup	5.0.40
--use-manager	Use Instance Manager to start server	5.0.4
--use-mysqld_safe	Use mysqld_safe to start server	5.0.4
--user	Run server using this login user name	5.0.4

- `--basedir=dir_name`

  The path to the MySQL installation directory.

- `--datadir=dir_name`

  The path to the MySQL data directory.

- `--pid-file=file_name`

  The path name of the file in which the server should write its process ID.

- `--service-startup-timeout=seconds`

  How long in seconds to wait for confirmation of server startup. If the server does not start within this time, `mysql.server` exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout). This option was added in MySQL 5.0.40. Before that, a value of 900 is always used.

- `--use-mysqld_safe`

  Use `mysqld_safe` to start the server. This is the default. This option was added in MySQL 5.0.4.

- `--use-manager`

  Use Instance Manager to start the server. This option was added in MySQL 5.0.4.

- `--user=user_name`

  The login user name to use for running `mysqld`. This option was added in MySQL 5.0.4.

4.3.4 mysqld_multi — Manage Multiple MySQL Servers

`mysqld_multi` is designed to manage several `mysqld` processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status. The MySQL Instance Manager is an alternative means of managing multiple servers (see Section 4.6.10, "`mysqlmanager` — The MySQL Instance Manager").

`mysqld_multi` searches for groups named `[mysqldN]` in `my.cnf` (or in the file named by the `--config-file` option). *`N`* can be any positive integer. This number is referred to in the following discussion as the option group number, or *`GNR`*. Group numbers distinguish option groups from one another and are used as arguments to `mysqld_multi` to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the `[mysqld]` group used

This documentation is for an older version. If you're
This documentation is for an older version. If you're

for starting mysqld. (See, for example, Section 2.18.5, “Starting and Stopping MySQL Automatically”.) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see Section 5.5, “Running Multiple MySQL Instances on One Machine”.

To invoke mysqld_multi, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

start, stop, and report indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, mysqld_multi performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named [mysqld17] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [mysqld10] through [mysqld13]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [mysqld17]:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [mysqld8] and [mysqld10] through [mysqld13]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

As of MySQL 5.0.42, mysqld_multi searches for option files as follows:

- With --no-defaults, no option files are read.
- With --defaults-file=*file_name*, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the --defaults-extra-file=*file_name* option, if one is given. (If the option is given multiple times, the last value is used.)

Before MySQL 5.0.42, the preceding options are not recognized. Files in the standard locations are read, and any file named by the --config-file=*file_name* option, if one is given. A file named by --config-file is read only for [mysqld*N*] option groups, not the [mysqld_multi] group.

Option files read are searched for [mysqld_multi] and [mysqld*N*] option groups. The [mysqld_multi] group can be used for options to mysqld_multi itself. [mysqld*N*] groups can be used for options passed to specific mysqld instances.

As of MySQL 5.0.82, the [mysqld] or [mysqld_safe] groups can be used for common options read by all instances of mysqld or mysqld_safe. You can specify a --defaults-file=*file_name* option to use a different configuration file for that instance, in which case the [mysqld] or [mysqld_safe] groups

This documentation is for an older version. If you're
This documentation is for an older version. If you're

from that file will be used for that instance. Before MySQL 5.0.82, some versions of mysqld_multi pass the --no-defaults options to instances, so these techniques are inapplicable.

mysqld_multi supports the following options.

- --help

  Display a help message and exit.

- --config-file=*file_name*

  As of MySQL 5.0.42, this option is deprecated. If given, it is treated the same way as --defaults-extra-file, described earlier. --config-file is removed in MySQL 5.5.

  Before MySQL 5.0.42, this option specifies the name of an extra option file. It affects where mysqld_multi looks for [mysqld*N*] option groups. Without this option, all options are read from the usual my.cnf file. The option does not affect where mysqld_multi reads its own options, which are always taken from the [mysqld_multi] group in the usual my.cnf file.

- --example

  Display a sample option file.

- --log=*file_name*

  Specify the name of the log file. If the file exists, log output is appended to it.

- --mysqladmin=*prog_name*

  The mysqladmin binary to be used to stop servers.

- --mysqld=*prog_name*

  The mysqld binary to be used. Note that you can specify mysqld_safe as the value for this option also. If you use mysqld_safe to start the server, you can include the mysqld or ledir options in the corresponding [mysqld*N*] option group. These options indicate the name of the server that mysqld_safe should start and the path name of the directory where the server is located. (See the descriptions for these options in Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.) Example:

  ```
  [mysqld38]
  mysqld = mysqld-debug
  ledir  = /opt/local/mysql/libexec
  ```

- --no-log

  Print log information to stdout rather than to the log file. By default, output goes to the log file.

- --password=*password*

  The password of the MySQL account to use when invoking mysqladmin. Note that the password value is not optional for this option, unlike for other MySQL programs.

- --silent

  Silent mode; disable warnings.

- --tcp-ip

This documentation is for an older version. If you're

Connect to each MySQL server through the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only through the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects `stop` and `report` operations.

- `--user=user_name`

  The user name of the MySQL account to use when invoking `mysqladmin`.

- `--verbose`

  Be more verbose.

- `--version`

  Display version information and exit.

Some notes about `mysqld_multi`:

- **Most important**: Before using `mysqld_multi` be sure that you understand the meanings of the options that are passed to the `mysqld` servers and *why* you would want to have separate `mysqld` processes. Beware of the dangers of using multiple `mysqld` servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See Section 5.5, “Running Multiple MySQL Instances on One Machine”.

- 

  Important

  Make sure that the data directory for each server is fully accessible to the Unix account that the specific `mysqld` process is started as. *Do not* use the Unix *`root`* account for this, unless you *know* what you are doing. See Section 6.1.5, “How to Run MySQL as a Normal User”.

- Make sure that the MySQL account used for stopping the `mysqld` servers (with the `mysqladmin` program) has the same user name and password for each server. Also, make sure that the account has the `SHUTDOWN` privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -p
  Enter password:
  mysql> CREATE USER 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  mysql> GRANT SHUTDOWN ON *.* TO 'multi_admin'@'localhost';
  ```

  See Section 6.2, “The MySQL Access Privilege System”. You have to do this for each `mysqld` server. Change the connection parameters appropriately when connecting to each one. Note that the host name part of the account name must permit you to connect as `multi_admin` from the host where you want to run `mysqld_multi`.

- The Unix socket file and the TCP/IP port number must be different for every `mysqld`. (Alternatively, if the host has multiple network addresses, you can use `--bind-address` to cause different servers to listen to different interfaces.)

- The `--pid-file` option is very important if you are using `mysqld_safe` to start `mysqld` (for example, `--mysqld=mysqld_safe`) Every `mysqld` should have its own process ID file. The advantage of using `mysqld_safe` instead of `mysqld` is that `mysqld_safe` monitors its `mysqld` process and restarts it if

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the process terminates due to a signal sent using `kill -9` or for other reasons, such as a segmentation fault. Please note that the `mysqld_safe` script might require that you start it from a certain place. This means that you might have to change location to a certain directory before running `mysqld_multi`. If you have problems starting, please see the `mysqld_safe` script. Check especially the lines:

```
--------------------------------------------------------------
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release)
if test -d $MY_PWD/data/mysql -a -f ./share/mysql/english/errmsg.sys -a \
 -x ./bin/mysqld
--------------------------------------------------------------
```

The test performed by these lines should be successful, or you might encounter problems. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

- You might want to use the `--user` option for `mysqld`, but to do this you need to run the `mysqld_multi` script as the Unix superuser (`root`). Having the option in the option file doesn't matter; you just get a warning if you are not the superuser and the `mysqld` processes are started under your own Unix account.

The following example shows how you might set up an option file for use with `mysqld_multi`. The order in which the `mysqld` programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth `[mysqldN]` groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This is an example of a my.cnf file for mysqld_multi.
# Usually this file is located in home dir ~/.my.cnf or /etc/my.cnf

[mysqld_multi]
mysqld     = /usr/local/mysql/bin/mysqld_safe
mysqladmin = /usr/local/mysql/bin/mysqladmin
user       = multi_admin
password   = my_password

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/data2/hostname.pid2
datadir    = /usr/local/mysql/data2
language   = /usr/local/mysql/share/mysql/english
user       = unix_user1

[mysqld3]
mysqld     = /path/to/mysqld_safe
ledir      = /path/to/mysqld-binary/
mysqladmin = /path/to/mysqladmin
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/data3/hostname.pid3
datadir    = /usr/local/mysql/data3
language   = /usr/local/mysql/share/mysql/swedish
user       = unix_user2

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/data4/hostname.pid4
datadir    = /usr/local/mysql/data4
language   = /usr/local/mysql/share/mysql/estonia
user       = unix_user3
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/data6/hostname.pid6
datadir    = /usr/local/mysql/data6
language   = /usr/local/mysql/share/mysql/japanese
user       = unix_user4
```

See Section 4.2.6, “Using Option Files”.

4.4 MySQL Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

4.4.1 `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles the `errmsg.sys` file from the text file located at `sql/share/errmsg.txt` in MySQL source distributions.

`comp_err` also generates `mysqld_error.h`, `mysqld_ername.h`, and `sql_state.h` header files.

For more information about how error messages are defined, see the MySQL Internals Manual.

Invoke `comp_err` like this:

```
shell> comp_err [options]
```

`comp_err` supports the following options.

- `--help`, `-?`

  Display a help message and exit.

- `--charset=dir_name`, `-C dir_name`

  The character set directory. The default is `../sql/share/charsets`.

- `--debug=debug_options`, `-# debug_options`

  Write a debugging log. A typical `debug_options` string is `d:t:O,file_name`. The default is `d:t:O,/tmp/comp_err.trace`.

- `--debug-info`, `-T`

  Print some debugging information when the program exits.

- `--header_file=file_name`, `-H file_name`

  The name of the error header file. The default is `mysqld_error.h`.

- `--in_file=file_name`, `-F file_name`

  The name of the input file. The default is `../sql/share/errmsg.txt`.

- `--name_file=file_name`, `-N file_name`

  The name of the error name file. The default is `mysqld_ername.h`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--out_dir=dir_name`, `-D dir_name`

  The name of the output base directory. The default is `../sql/share/`.

- `--out_file=file_name`, `-O file_name`

  The name of the output file. The default is `errmsg.sys`.

- `--statefile=file_name`, `-S file_name`

  The name for the SQLSTATE header file. The default is `sql_state.h`.

- `--version`, `-V`

  Display version information and exit.

4.4.2 make_win_bin_dist — Package MySQL Distribution as Zip Archive

This script is used on Windows after building a MySQL distribution from source to create executable programs. It packages the binaries and support files into a Zip archive that can be unpacked at the location where you want to install MySQL.

`make_win_bin_dist` is a shell script, so you must have Cygwin installed to use it.

This program's use is subject to change. Currently, you invoke it as follows from the root directory of your source distribution:

```
shell> make_win_bin_dist [options] package_basename [copy_def ...]
```

The *`package_basename`* argument provides the base name for the resulting Zip archive. This name will be the name of the directory that results from unpacking the archive.

Because you might want to include files of directories from other builds, you can instruct this script to copy them in for you, using *`copy_def`* arguments, which have the form *`relative_dest_name`*=*`source_name`*.

Example:

```
bin/mysqld-max.exe=../my-max-build/sql/release/mysqld.exe
```

If you specify a directory, the entire directory will be copied.

`make_win_bin_dist` supports the following options.

- `--debug`

  Pack the debug binaries and produce an error if they were not built.

- `--embedded`

  Pack the embedded server and produce an error if it was not built. The default is to pack it if it was built.

- `--exe-suffix=suffix`

  Add a suffix to the base name of the `mysql` binary. For example, a suffix of `-abc` produces a binary named `mysqld-abc.exe`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--no-debug`

  Do not pack the debug binaries even if they were built.

- `--no-embedded`

  Do not pack the embedded server even if it was built.

- `--only-debug`

  Use this option when the target for this build was `Debug`, and you just want to replace the normal binaries with debug versions (that is, do not use separate `debug` directories).

4.4.3 `make_win_src_distribution` — Create Source Distribution for Windows

`make_win_src_distribution` creates a Windows source package to be used on Windows systems. It is used after you configure and build the source distribution on a Unix or Unix-like system so that you have a server binary to work with. (See the instructions at Section 2.10.8.5, “Creating a Windows Source Package from the Bazaar Repository”.)

Invoke `make_win_src_distribution` like this from the top-level directory of a MySQL source distribution:

```
shell> make_win_src_distribution [options]
```

`make_win_src_distribution` understands the following options:

- `--help`

  Display a help message and exit.

- `--debug`

  Print information about script operations; do not create a package.

- `--dirname`

  Directory name to copy files (intermediate).

- `--silent`

  Do not print verbose list of files processed.

- `--suffix`

  The suffix name for the package.

- `--tar`

  Create a `.tar.gz` package instead of a `.zip` package.

  By default, `make_win_src_distribution` creates a Zip-format archive with the name `mysql-VERSION-win-src.zip`, where *`VERSION`* represents the version of your MySQL source tree.

- `--tmp`

This documentation is for an older version. If you're

Specify the temporary location.

4.4.4 mysqlbug — Generate Bug Report

This program enables you to generate a bug report and send it to Oracle Corporation. It is a shell script and runs on Unix.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports. If you have no Web access, you can generate a bug report by using the mysqlbug script.

mysqlbug helps you generate a report by determining much of the following information automatically, but if something important is missing, please include it with your message. mysqlbug can be found in the scripts directory (source distribution) and in the bin directory under your MySQL installation directory (binary distribution).

Invoke mysqlbug without arguments:

```
shell> mysqlbug
```

The script will place you in an editor with a copy of the report to be sent. Edit the lines near the beginning that indicate the nature of the problem. Then write the file to save your changes, quit the editor, and mysqlbug will send the report by email.

4.4.5 mysql_fix_privilege_tables — Upgrade MySQL System Tables

Some releases of MySQL introduce changes to the structure of the system tables in the mysql database to add new privileges or support new features. When you update to a new version of MySQL, update your system tables as well to make sure that their structure is up to date. Otherwise, there might be capabilities that you cannot take advantage of.

mysql_fix_privilege_tables is an older script that previously was used to uprade the system tables in the mysql database after a MySQL upgrade.

Note

As of MySQL 5.0.19, mysql_fix_privilege_tables is superseded by mysql_upgrade, which should be used instead. See Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”.

Before running mysql_fix_privilege_tables, make a backup of your mysql database.

On Unix or Unix-like systems, update the system tables by running the mysql_fix_privilege_tables script:

```
shell> mysql_fix_privilege_tables
```

You must run this script while the server is running. It attempts to connect to the server running on the local host as root. If your root account requires a password, indicate the password on the command line like this:

This documentation is for an older version. If you're

```
shell> mysql_fix_privilege_tables --password=root_password
```

The mysql_fix_privilege_tables script performs any actions necessary to convert your system tables to the current format. You might see some Duplicate column name warnings as it runs; you can ignore them.

After running the script, stop the server and restart it so that any changes made to the system tables take effect.

On Windows systems, MySQL distributions include a mysql_fix_privilege_tables.sql SQL script that you can run using the mysql client. For example, if your MySQL installation is located at C:\Program Files\MySQL\MySQL Server 5.0, the commands look like this:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 5.0"
C:\> bin\mysql -u root -p mysql
mysql> SOURCE share/mysql_fix_privilege_tables.sql
```

Note

Prior to version 5.0.38, this script is found in the scripts directory.

The mysql command will prompt you for the root password; enter it when prompted.

If your installation is located in some other directory, adjust the path names appropriately.

As with the Unix procedure, you might see some Duplicate column name warnings as mysql processes the statements in the mysql_fix_privilege_tables.sql script; you can ignore them.

After running the script, stop the server and restart it.

4.4.6 mysql_install_db — Initialize MySQL Data Directory

mysql_install_db initializes the MySQL data directory and creates the system tables that it contains, if they do not exist. mysql_install_db is a shell script and is available only on Unix platforms.

To invoke mysql_install_db, use the following syntax:

```
shell> mysql_install_db [options]
```

Because the MySQL server, mysqld, must access the data directory when it runs later, you should either run mysql_install_db from the same system account that will be used for running mysqld, or run it as root and specify the --user option to indicate the user name that mysqld will run as. It might be necessary to specify other options such as --basedir or --datadir if mysql_install_db does not use the correct locations for the installation directory or data directory. For example:

```
shell> bin/mysql_install_db --user=mysql \
         --basedir=/opt/mysql/mysql \
         --datadir=/opt/mysql/mysql/data
```

mysql_install_db needs to invoke mysqld with the --bootstrap and --skip-grant-tables options. If MySQL was configured with the --disable-grant-options configuration option, --bootstrap and --skip-grant-tables will be disabled (see Section 2.17.3, “MySQL Source-Configuration Options”). To handle this, set the MYSQLD_BOOTSTRAP environment variable to the full path name of a server that has all options enabled. mysql_install_db will use that server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysql_install_db supports the following options, which can be specified on the command line or in the [mysql_install_db] group of an option file. (Options that are common to mysqld can also be specified in the [mysqld] group.) Other options are passed to mysqld. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”. mysql_install_db also supports the options for processing option files described at Section 4.2.7, “Command-Line Options that Affect Option-File Handling”.

- --help

  Display a help message and exit.

- --basedir=*dir_name*

  The path to the MySQL installation directory.

- --builddir=*dir_name*

  For use with --srcdir and out-of-source builds. Set this to the location of the directory where the built files reside.

- --cross-bootstrap

  For internal use. This option is used for building system tables on one host intended for another.

- --datadir=*dir_name*

  The path to the MySQL data directory.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --force

  Cause mysql_install_db to run even if DNS does not work. Grant table entries normally created using host names will use IP addresses instead.

- --ldata=*dir_name*

  A synonym for --datadir.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

- --rpm

  For internal use. This option is used during the MySQL installation process for install operations performed using RPM packages.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--skip-name-resolve`

  Use IP addresses rather than host names when creating grant table entries. This option can be useful if your DNS does not work.

- `--srcdir=dir_name`

  For internal use. This option specifies the directory under which `mysql_install_db` looks for support files such as the error message file and the file for populating the help tables. This option was added in MySQL 5.0.32.

- `--user=user_name`

  The system (login) user name to use for running `mysqld`. Files and directories created by `mysqld` will be owned by this user. You must be the system `root` user to use this option. By default, `mysqld` runs using your current login name and files and directories that it creates will be owned by you.

- `--verbose`

  Verbose mode. Print more information about what the program does.

- `--windows`

  For internal use. This option is used for creating Windows distributions.

4.4.7 `mysql_secure_installation` — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for `root` accounts.
- You can remove `root` accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the `test` database (which by default can be accessed by all users, even anonymous users), and privileges that permit anyone to access databases with names that start with `test_`.

`mysql_secure_installation` helps you implement security recommendations similar to those described at Section 2.18.4, “Securing the Initial MySQL Accounts”.

Invoke `mysql_secure_installation` without arguments:

```
shell> mysql_secure_installation
```

When executed, the script prompts you to determine which actions to perform.

`mysql_secure_installation` is not available on Windows.

4.4.8 `mysql_tzinfo_to_sql` — Load the Time Zone Tables

The `mysql_tzinfo_to_sql` program loads the time zone tables in the `mysql` database. It is used on systems that have a *zoneinfo* database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Solaris, and OS X. One likely location for these files is the `/usr/share/zoneinfo` directory (`/usr/share/lib/zoneinfo` on Solaris). If your system does not have a zoneinfo database, you can use the downloadable package described in Section 10.6, “MySQL Server Time Zone Support”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysql_tzinfo_to_sql can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the zoneinfo directory path name to mysql_tzinfo_to_sql and send the output into the mysql program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

mysql_tzinfo_to_sql reads your system's time zone files and generates SQL statements from them. mysql processes those statements to load the time zone tables.

The second syntax causes mysql_tzinfo_to_sql to load a single time zone file *tz_file* that corresponds to a time zone name *tz_name*:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke mysql_tzinfo_to_sql using the third syntax, which initializes the leap second information. *tz_file* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

After running mysql_tzinfo_to_sql, it is best to restart the server so that it does not continue to use any previously cached time zone data.

4.4.9 mysql_upgrade — Check Tables for MySQL Upgrade

mysql_upgrade examines all tables in all databases for incompatibilities with the current version of MySQL Server. mysql_upgrade also upgrades the system tables so that you can take advantage of new privileges or capabilities that might have been added.

If mysql_upgrade finds that a table has a possible incompatibility, it performs a table check and, if problems are found, attempts a table repair. If the table cannot be repaired, see Section 2.19.4, “Rebuilding or Repairing Tables or Indexes” for manual table repair strategies.

You should execute mysql_upgrade each time you upgrade MySQL. It supersedes the older mysql_fix_privilege_tables script, which should no longer be used.

If you install MySQL from RPM packages on Linux, you must install the server and client RPMs. mysql_upgrade is included in the server RPM but requires the client RPM because the latter includes mysqlcheck. (See Section 2.12, “Installing MySQL on Linux Using RPM Packages”.)

Caution

You should always back up your current MySQL installation *before* performing an upgrade. See Section 7.2, “Database Backup Methods”.

Some upgrade incompatibilities may require special handling before you upgrade your MySQL installation and run mysql_upgrade. See Section 2.19.1, “Upgrading MySQL”, for instructions on determining whether any such incompatibilities apply to your installation and how to handle them.

To use mysql_upgrade, make sure that the server is running. Then invoke it like this:

This documentation is for an older version. If you're

```
shell> mysql_upgrade [options]
```

After running mysql_upgrade, stop the server and restart it so that any changes made to the system tables take effect.

If you have multiple MySQL server instances running, invoke mysql_upgrade with connection parameters appropriate for connecting to the desired server. For example, with servers running on the local host on parts 3306 through 3308, upgrade each of them by connecting to the appropriate port:

```
shell> mysql_upgrade --protocol=tcp -P 3306 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3307 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3308 [other_options]
```

For local host connections on Unix, the --protocol=tcp option forces a connection using TCP/IP rather than the Unix socket file.

mysql_upgrade executes the following commands to check and repair tables and to upgrade the system tables:

```
mysqlcheck --no-defaults --check-upgrade --all-databases --auto-repair
mysql < fix_priv_tables
```

Notes about the preceding commands:

- Because mysql_upgrade invokes mysqlcheck with the --all-databases option, it processes all tables in all databases, which might take a long time to complete. Each table is locked and therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables.
- For details about what checks the --check-upgrade option entails, see the description of the FOR UPGRADE option of the CHECK TABLE statement (see Section 13.7.2.3, “CHECK TABLE Syntax”).
- *fix_priv_tables* represents a script generated internally by mysql_upgrade that contains SQL statements to upgrade the tables in the mysql database.

All checked and repaired tables are marked with the current MySQL version number. This ensures that next time you run mysql_upgrade with the same version of the server, it can tell whether there is any need to check or repair the table again.

mysql_upgrade also saves the MySQL version number in a file named mysql_upgrade_info in the data directory. This is used to quickly check whether all tables have been checked for this release so that table-checking can be skipped. To ignore this file and perform the check regardless, use the --force option.

In MySQL 5.0.19, mysql_upgrade was added as a shell script and worked only for Unix systems. As of MySQL 5.0.23, mysql_upgrade is an executable binary and is available on all systems.

mysql_upgrade does not upgrade the contents of the help tables. For upgrade instructions, see Section 5.1.8, “Server-Side Help”.

mysql_upgrade supports the following options, which can be specified on the command line or in the [mysql_upgrade] and [client] groups of an option file. Unrecognized options are passed to mysqlcheck. For information about option files, see Section 4.2.6, “Using Option Files”.

- --help

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Display a short help message and exit.

- `--basedir=dir_name`

  The path to the MySQL installation directory. This option is accepted for backward compatibility but ignored.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”. This option was added in MySQL 5.0.30.

- `--compress`

  Compress all information sent between the client and the server if both support compression. This option was added in MySQL 5.0.30.

- `--datadir=dir_name`

  The path to the data directory. This option is accepted for backward compatibility but ignored.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:O,/tmp/mysql_upgrade.trace`.

- `--debug-info`, `-T`

  Print some debugging information when the program exits.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”. This option was added in MySQL 5.0.30.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql_upgrade` normally reads the `[client]` and `[mysql_upgrade]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql_upgrade` also reads the `[client_other]` and `[mysql_upgrade_other]` groups.

- `--force`

  Ignore the `mysql_upgrade_info` file and force execution even if `mysql_upgrade` has already been executed for the current version of MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host. This option was added in MySQL 5.0.30.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysql_upgrade` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, "End-User Guidelines for Password Security". You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections. This option was added in MySQL 5.0.30.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, "Connecting to the MySQL Server".

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections. This option was added in MySQL 5.0.30.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, "Command Options for Secure Connections". These options were added in MySQL 5.0.30.

- `--tmpdir=dir_name`, `-t dir_name`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The path name of the directory to use for creating temporary files. This option was added in MySQL 5.0.62.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server. The default user name is `root`.

- `--verbose`

  Verbose mode. Print more information about what the program does.

4.5 MySQL Client Programs

This section describes client programs that connect to the MySQL server.

4.5.1 `mysql` — The MySQL Command-Line Tool

`mysql` is a simple SQL shell with input line editing capabilities. It supports interactive and noninteractive use. When used interactively, query results are presented in an ASCII-table format. When used noninteractively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` option. This forces `mysql` to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Using `mysql` is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with “`;`”, `\g`, or `\G` and press Enter.

As of MySQL 5.0.25, typing **Control+C** causes `mysql` to attempt to kill the current statement. If this cannot be done, or **Control+C** is typed again before the statement is killed, `mysql` exits. Previously, **Control+C** caused `mysql` to exit in all cases.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

On Unix, the `mysql` client logs statements executed interactively to a history file. See Section 4.5.1.3, “mysql Logging”.

4.5.1.1 mysql Options

`mysql` supports the following options, which can be specified on the command line or in the `[mysql]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

This documentation is for an older version. If you're

Table 4.3 mysql Options

Format	Description	Introduced
--auto-rehash	Enable automatic rehashing	
--batch	Do not use history file	
--character-sets-dir	Directory where character sets are installed	
--column-names	Write column names in results	
--comments	Whether to retain or strip comments in statements sent to the server	5.0.52
--compress	Compress all information sent between client and server	
--connect_timeout	Number of seconds before connection timeout	
--database	The database to use	
--debug	Write debugging log; supported only if MySQL was built with debugging support	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	5.0.10
--delimiter	Set the statement delimiter	
--execute	Execute the statement and quit	
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Connect to MySQL server on given host	
--html	Produce HTML output	
--ignore-spaces	Ignore spaces after function names	
--line-numbers	Write line numbers for errors	
--local-infile	Enable or disable for LOCAL capability for LOAD DATA INFILE	
--max_allowed_packet	Maximum packet length to send to or receive from server	
--max_join_size	The automatic limit for rows in a join when using --safe-updates	
--named-commands	Enable named mysql commands	
--net_buffer_length	Buffer size for TCP/IP and socket communication	
--no-auto-rehash	Disable automatic rehashing	
--no-beep	Do not beep when errors occur	
--no-defaults	Read no option files	
--no-named-commands	Disable named mysql commands	
--no-pager	Deprecated form of --skip-pager	
--no-tee	Do not copy output to a file	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--one-database	Ignore statements except those for the default database named on the command line	
--pager	Use the given command for paging query output	
--password	Password to use when connecting to server	
--pipe	On Windows, connect to server using named pipe	
--port	TCP/IP port number to use for connection	
--print-defaults	Print default options	
--prompt	Set the prompt to the specified format	
--protocol	Connection protocol to use	
--quick	Do not cache each query result	
--raw	Write column values without escape conversion	
--reconnect	If the connection to the server is lost, automatically try to reconnect	
--i-am-a-dummy, --safe-updates	Allow only UPDATE and DELETE statements that specify key values	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	
--select_limit	The automatic limit for SELECT statements when using --safe-updates	
--shared-memory-base-name	The name of shared memory to use for shared-memory connections	
--show-warnings	Show warnings after each statement if there are any	5.0.6
--sigint-ignore	Ignore SIGINT signals (typically the result of typing Control+C)	
--silent	Silent mode	
--skip-auto-rehash	Disable automatic rehashing	
--skip-column-names	Do not write column names in results	
--skip-line-numbers	Skip line numbers for errors	
--skip-named-commands	Disable named mysql commands	
--skip-pager	Disable paging	
--skip-reconnect	Disable reconnecting	
--socket	For connections to localhost, the Unix socket file to use	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert	Path of file that contains X509 certificate in PEM format	
--ssl-cipher	List of permitted ciphers to use for connection encryption	
--ssl-key	Path of file that contains X509 key in PEM format	
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--table	Display output in tabular format	
--tee	Append a copy of output to named file	
--unbuffered	Flush the buffer after each query	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--vertical	Print query output rows vertically (one line per column value)	
--wait	If the connection cannot be established, wait and retry instead of aborting	
--xml	Produce XML output	

- `--help`, `-?`

  Display a help message and exit.

- `--auto-rehash`

  Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--disable-auto-rehash` to disable rehashing. That causes `mysql` to start faster, but you must issue the `rehash` command if you want to use name completion.

  To complete a name, enter the first part and press Tab. If the name is unambiguous, `mysql` completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

- `--batch`, `-B`

  Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

  Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--column-names`

  Write column names in results.

- `--comments`, `-c`

  Whether to preserve comments in statements sent to the server. The default is --skip-comments (discard comments), enable with --comments (preserve comments). This option was added in MySQL 5.0.52.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--database=`*`db_name`*, `-D` *`db_name`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The database to use. This is useful primarily in an option file.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysql.trace`.

  This option is available only if MySQL was built using `--with-debug`. MySQL release binaries are *not* built using this option.

- `--debug-info`, `-T`

  Print some debugging information when the program exits.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set for the client and connection.

  A common issue that can occur when the operating system uses `utf8` or another multibyte character set is that output from the `mysql` client is formatted incorrectly, due to the fact that the MySQL client uses the `latin1` character set by default. You can usually fix such issues by using this option to force the client to use the system character set instead.

  See Section 10.5, “Character Set Configuration”, for more information.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql` normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups. This option was added in MySQL 5.0.10.

- `--delimiter=str`

  Set the statement delimiter. The default is the semicolon character (“`;`”).

- `--disable-named-commands`

  Disable named commands. Use the `\*` form only, or use named commands only at the beginning of a line ending with a semicolon (“`;`”). `mysql` starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See Section 4.5.1.2, “mysql Commands”.

- `--execute=statement`, `-e statement`

  Execute the statement and quit. The default output format is like that produced with `--batch`. See Section 4.2.4, “Using Options on the Command Line”, for some examples. With this option, `mysql` does not use the history file.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--html`, `-H`

  Produce HTML output.

- `--ignore-spaces`, `-i`

  Ignore spaces after function names. The effect of this is described in the discussion for the `IGNORE_SPACE` SQL mode (see Section 5.1.7, “Server SQL Modes”).

- `--line-numbers`

  Write line numbers for errors. Disable this with `--skip-line-numbers`.

- `--local-infile[={0|1}]`

  Enable or disable `LOCAL` capability for `LOAD DATA INFILE`. With no value, the option enables `LOCAL`. The option may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable `LOCAL`. Enabling `LOCAL` has no effect if the server does not also support it.

- `--named-commands`, `-G`

  Enable named `mysql` commands. Long-format commands are permitted, not just short-format commands. For example, `quit` and `\q` both are recognized. Use `--skip-named-commands` to disable named commands. See Section 4.5.1.2, “mysql Commands”.

- `--no-auto-rehash`, `-A`

  This has the same effect as `--skip-auto-rehash`. See the description for `--auto-rehash`.

- `--no-beep`, `-b`

  Do not beep when errors occur.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--no-named-commands`, `-g`

  Deprecated, use `--disable-named-commands` instead. `--no-named-commands` is removed in MySQL 5.5.

- `--no-pager`

  Deprecated form of `--skip-pager`. See the `--pager` option. `--no-pager` is removed in MySQL 5.5.

- `--no-tee`

  Deprecated form of `--skip-tee`. See the `--tee` option. `--no-tee` is removed in MySQL 5.5.

- `--one-database`, `-o`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Ignore statements except those that occur while the default database is the one named on the command line. This option is rudimentary and should be used with care. Statement filtering is based only on USE statements.

Initially, mysql executes statements in the input because specifying a database *db_name* on the command line is equivalent to inserting USE *db_name* at the beginning of the input. Then, for each USE statement encountered, mysql accepts or rejects following statements depending on whether the database named is the one on the command line. The content of the statements is immaterial.

Suppose that mysql is invoked to process this set of statements:

```
DELETE FROM db2.t2;
USE db2;
DROP TABLE db1.t1;
CREATE TABLE db1.t1 (i INT);
USE db1;
INSERT INTO t1 (i) VALUES(1);
CREATE TABLE db2.t1 (j INT);
```

If the command line is mysql --force --one-database db1, mysql handles the input as follows:

- The DELETE statement is executed because the default database is db1, even though the statement names a table in a different database.
- The DROP TABLE and CREATE TABLE statements are not executed because the default database is not db1, even though the statements name a table in db1.
- The INSERT and CREATE TABLE statements are executed because the default database is db1, even though the CREATE TABLE statement names a table in a different database.

- --pager[=*command*]

  Use the given command for paging query output. If the command is omitted, the default pager is the value of your PAGER environment variable. Valid pagers are less, more, cat [> filename], and so forth. This option works only on Unix and only in interactive mode. To disable paging, use --skip-pager. Section 4.5.1.2, “mysql Commands”, discusses output paging further.

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysql prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for the connection.

- --print-defaults

This documentation is for an older version. If you're

Print the program name and all options that it gets from option files.

- `--prompt=format_str`

  Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in Section 4.5.1.2, “mysql Commands”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--quick`, `-q`

  Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, `mysql` does not use the history file.

- `--raw`, `-r`

  For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the `--batch` or `--silent` option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, `NUL`, and backslash are written as `\n`, `\t`, `\0`, and `\\`. The `--raw` option disables this character escaping.

  The following example demonstrates tabular versus nontabular output and the use of raw mode to disable escaping:

```
% mysql
mysql> SELECT CHAR(92);
+----------+
| CHAR(92) |
+----------+
| \        |
+----------+

% mysql -s
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- `--reconnect`

  If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`.

- `--safe-updates`, `--i-am-a-dummy`, `-U`

  Permit only those `UPDATE` and `DELETE` statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using `--safe-updates` on the command line. See Section 4.5.1.6, “mysql Tips”, for more information about this option.

- `--secure-auth`

This documentation is for an older version. If you're

Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

- `--show-warnings`

  Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode. This option was added in MySQL 5.0.6.

- `--sigint-ignore`

  Ignore `SIGINT` signals (typically the result of typing **Control+C**).

- `--silent`, `-s`

  Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

  This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--skip-column-names`, `-N`

  Do not write column names in results.

- `--skip-line-numbers`, `-L`

  Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- `--table`, `-t`

  Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=file_name`

  Append a copy of output to the given file. This option works only in interactive mode. Section 4.5.1.2, “mysql Commands”, discusses tee files further.

- `--unbuffered`, `-n`

  Flush the buffer after each query.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version`, `-V`

  Display version information and exit.

- `--vertical`, `-E`

  Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait`, `-w`

  If the connection cannot be established, wait and retry instead of aborting.

- `--xml`, `-X`

  Produce XML output.

Note

Prior to MySQL 5.0.26, there was no differentiation in the output when using this option between columns containing the `NULL` value and columns containing the string literal `'NULL'`; both were represented as

```
<field name="column_name">NULL</field>
```

Beginning with MySQL 5.0.26, the output when `--xml` is used with `mysql` matches that of `mysqldump --xml`. See Section 4.5.4, “mysqldump — A Database Backup Program” for details.

Beginning with MySQL 5.0.40, the XML output also uses an XML namespace, as shown here:

```
shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
<row>
<field name="Variable_name">version</field>
<field name="Value">5.0.40-debug</field>
</row>

<row>
<field name="Variable_name">version_comment</field>
<field name="Value">Source distribution</field>
</row>

<row>
<field name="Variable_name">version_compile_machine</field>
<field name="Value">i686</field>
</row>

<row>
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
<field name="Variable_name">version_compile_os</field>
<field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

(See Bug #25946.)

You can also set the following variables by using `--var_name=value`. The `--set-variable` format is deprecated.

- `connect_timeout`

  The number of seconds before connection timeout. (Default value is `0`.)

- `max_allowed_packet`

  The maximum size of the buffer for client/server communication. The default is 16MB, the maximum is 1GB.

- `max_join_size`

  The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `net_buffer_length`

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

  The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

It is also possible to set variables by using `--var_name=value`. The `--set-variable` format is deprecated.

4.5.1.2 mysql Commands

`mysql` sends each SQL statement that you issue to the server to be executed. There is also a set of commands that `mysql` itself interprets. For a list of these commands, type `help` or `\h` at the `mysql>` prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear command.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given
               outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing
               binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.

For server side help, type 'help contents'
```

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The use of short-form commands within multiple-line `/* ... */` comments is not supported.

- `help [arg]`, `\h [arg]`, `\? [arg]`, `? [arg]`

  Display a help message listing the available `mysql` commands.

  If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see Section 4.5.1.4, “mysql Server-Side Help”.

- `charset charset_name`, `\C charset_name`

  Change the default character set and issue a `SET NAMES` statement. This enables the character set to remain synchronized on the client and server if `mysql` is run with auto-reconnect enabled (which is not recommended), because the specified character set is used for reconnects. This command was added in MySQL 5.0.19.

- `clear`, `\c`

  Clear the current input. Use this if you change your mind about executing the statement that you are entering.

- `connect [db_name host_name]`, `\r [db_name host_name]`

  Reconnect to the server. The optional database name and host name arguments may be given to specify the default database or the host where the server is running. If omitted, the current values are used.

- `delimiter str`, `\d str`

  Change the string that `mysql` interprets as the separator between SQL statements. The default is the semicolon character (“`;`”).

  The delimiter string can be specified as an unquoted or quoted argument on the `delimiter` command line. Quoting can be done with either single quote (`'`), double quote (`"`), or backtick (`` ` ``) characters. To include a quote within a quoted string, either quote the string with a different quote character or escape the quote with a backslash (“`\`”) character. Backslash should be avoided outside of quoted strings because it is the escape character for MySQL. For an unquoted argument, the delimiter is read up to the first space or end of line. For a quoted argument, the delimiter is read up to the matching quote on the line.

  `mysql` interprets instances of the delimiter string as a statement delimiter anywhere it occurs, except within quoted strings. Be careful about defining a delimiter that might occur within other words. For example, if you define the delimiter as `X`, you will be unable to use the word `INDEX` in statements. `mysql` interprets this as `INDE` followed by the delimiter `X`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When the delimiter recognized by mysql is set to something other than the default of ";", instances of that character are sent to the server without interpretation. However, the server itself still interprets ";" as a statement delimiter and processes statements accordingly. This behavior on the server side comes into play for multiple-statement execution (see Section 20.6.16, "C API Support for Multiple Statement Execution"), and for parsing the body of stored procedures and functions and triggers (see Section 18.1, "Defining Stored Programs").

- edit, \e

  Edit the current input statement. mysql checks the values of the EDITOR and VISUAL environment variables to determine which editor to use. The default editor is vi if neither variable is set.

  The edit command works only in Unix.

- ego, \G

  Send the current statement to the server to be executed and display the result using vertical format.

  Be careful about defining a delimiter that might occur within other words. For example, if you define the delimiter as X, you will be unable to use the word INDEX in statements.

- exit, \q

  Exit mysql.

- go, \g

  Send the current statement to the server to be executed.

- nopager, \n

  Disable output paging. See the description for pager.

  The nopager command works only in Unix.

- notee, \t

  Disable output copying to the tee file. See the description for tee.

- nowarning, \w

  Disable display of warnings after each statement. This command was added in MySQL 5.0.6.

- pager [*command*], \P [*command*]

  Enable output paging. By using the --pager option when you invoke mysql, it is possible to browse or search query results in interactive mode with Unix programs such as less, more, or any other similar program. If you specify no value for the option, mysql checks the value of the PAGER environment variable and sets the pager to that. Pager functionality works only in interactive mode.

  Output paging can be enabled interactively with the pager command and disabled with nopager. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or stdout if no pager was specified.

  Output paging works only in Unix because it uses the popen() function, which does not exist on Windows. For Windows, the tee option can be used instead to save query output, although it is not as convenient as pager for browsing output in some situations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- print, \p

  Print the current input statement without executing it.

- prompt [str], \R [str]

  Reconfigure the mysql prompt to the given string. The special character sequences that can be used in the prompt are described later in this section.

  If you specify the prompt command with no argument, mysql resets the prompt to the default of mysql>.

- quit, \q

  Exit mysql.

- rehash, \#

  Rebuild the completion hash that enables database, table, and column name completion while you are entering statements. (See the description for the --auto-rehash option.)

- source *file_name*, \. *file_name*

  Read the named file and executes the statements contained therein. On Windows, you can specify path name separators as / or \\.

- status, \s

  Provide status information about the connection and the server you are using. If you are running in --safe-updates mode, status also prints the values for the mysql variables that affect your queries.

- system *command*, \! *command*

  Execute the given command using your default command interpreter.

  The system command works only in Unix.

- tee [*file_name*], \T [*file_name*]

  By using the --tee option when you invoke mysql, you can log statements and their output. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. mysql flushes results to the file after each statement, just before it prints its next prompt. Tee functionality works only in interactive mode.

  You can enable this feature interactively with the tee command. Without a parameter, the previous file is used. The tee file can be disabled with the notee command. Executing tee again re-enables logging.

- use *db_name*, \u *db_name*

  Use *db_name* as the default database.

- warnings, \W

  Enable display of warnings after each statement (if there are any). This command was added in MySQL 5.0.6.

Here are a few tips about the pager command:

- You can use it to write to a file and the results go only to the file:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> pager cat > /tmp/log.txt
```

You can also pass any options for the program that you want to use as your pager:

```
mysql> pager less -n -i -S
```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

```
shell> man less
```

- The `-F` and `-X` options may be used with `less` to cause it to exit if output fits on one screen, which is convenient when no scrolling is necessary:

```
mysql> pager less -n -i -S -F -X
```

- You can specify very complex pager commands for handling query output:

```
mysql> pager cat | tee /dr1/tmp/res.txt \
         | tee /dr2/tmp/res2.txt | less -n -i -S
```

  In this example, the command would send query results to two files in two different directories on two different file systems mounted on `/dr1` and `/dr2`, yet still display the results onscreen using `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

The `prompt` command reconfigures the default `mysql>` prompt. The string for defining the prompt can contain the following special sequences.

Option	Description
`\c`	A counter that increments for each statement you issue
`\D`	The full current date
`\d`	The default database
`\h`	The server host
`\l`	The current delimiter (new in 5.0.25)
`\m`	Minutes of the current time
`\n`	A newline character
`\O`	The current month in three-letter format (Jan, Feb, …)
`\o`	The current month in numeric format

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option	Description
\P	am/pm
\p	The current TCP/IP port or socket file
\R	The current time, in 24-hour military time (0–23)
\r	The current time, standard 12-hour time (1–12)
\S	Semicolon
\s	Seconds of the current time
\t	A tab character
\U	Your full *user_name@host_name* account name
\u	Your user name
\v	The server version
\w	The current day of the week in three-letter format (Mon, Tue, …)
\Y	The current year, four digits
\y	The current year, two digits
_	A space
\	A space (a space follows the backslash)
\'	Single quote
\"	Double quote
\\	A literal "\" backslash character
x	*x*, for any "*x*" not listed above

You can set the prompt in several ways:

- *Use an environment variable.* You can set the MYSQL_PS1 environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- *Use a command-line option.* You can set the --prompt option on the command line to mysql. For example:

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the prompt option in the [mysql] group of any MySQL option file, such as /etc/my.cnf or the .my.cnf file in your home directory. For example:

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

In this example, note that the backslashes are doubled. If you set the prompt using the prompt option in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of permissible prompt options and the set of special escape sequences that are recognized in option files. (The rules for escape sequences in option files are listed in Section 4.2.6, "Using Option Files".) The overlap may cause you problems if you use single backslashes. For example, \s is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in HH:MM:SS> format:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

4.5.1.3 mysql Logging

On Unix, the `mysql` client logs statements executed interactively to a history file. By default, this file is named `.mysql_history` in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

How Logging Occurs

Statement logging occurs as follows:

- Statements are logged only when executed interactively. Statements are noninteractive, for example, when read from a file or a pipe. It is also possible to suppress statement logging by using the `--batch` or `--execute` option.
- `mysql` logs each nonempty statement line individually.
- If a statement spans multiple lines (not including the terminating delimiter), `mysql` concatenates the lines to form the complete statement, maps newlines to spaces, and logs the result, plus a delimiter.

Consequently, an input statement that spans multiple lines can be logged twice. Consider this input:

```
mysql> SELECT
    -> 'Today is'
    -> ,
    -> CURDATE()
    -> ;
```

In this case, `mysql` logs the “SELECT”, “'Today is'”, “,”, “CURDATE()”, and “;” lines as it reads them. It also logs the complete statement, after mapping `SELECT\n'Today is'\n,\nCURDATE()` to `SELECT 'Today is' , CURDATE()`, plus a delimiter. Thus, these lines appear in logged output:

```
SELECT
'Today is'
,
CURDATE()
;
SELECT 'Today is' , CURDATE();
```

Controlling the History File

The `.mysql_history` file should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you do not want to maintain a history file, first remove `.mysql_history` if it exists. Then use either of the following techniques to prevent it from being created again:

- Set the `MYSQL_HISTFILE` environment variable to `/dev/null`. To cause this setting to take effect each time you log in, put it in one of your shell's startup files.
- Create `.mysql_history` as a symbolic link to `/dev/null`; this need be done only once:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

4.5.1.4 mysql Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.8, “Server-Side Help”).

If there is no match for the search string, the search fails:

```
mysql> help me

Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use `help contents` to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
   Account Management
   Administration
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Language Structure
   Storage Engines
   Stored Routines
   Table Maintenance
   Transactions
   Triggers
```

If the search string matches multiple items, `mysql` shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
   SHOW
   SHOW BINARY LOGS
   SHOW ENGINE
   SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-binary-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

The search string can contain the wildcard characters “%” and “_”. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, `HELP rep%` returns a list of topics that begin with `rep`:

```
mysql> HELP rep%
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
   REPAIR TABLE
   REPEAT FUNCTION
   REPEAT LOOP
   REPLACE
   REPLACE FUNCTION
```

4.5.1.5 Executing SQL Statements from a Text File

The `mysql` client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell `mysql` to read its input from that file. To do so, create a text file *`text_file`* that contains the statements you wish to execute. Then invoke `mysql` as shown here:

```
shell> mysql db_name < text_file
```

If you place a `USE db_name` statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running `mysql`, you can execute an SQL script file using the `source` command or `\.` command:

```
mysql> source file_name
mysql> \. file_name
```


This documentation is for an older version. If you're

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs <info_to_display>.

You can also invoke mysql with the --verbose option, which causes each statement to be displayed before the result that it produces.

As of MySQL 5.0.54, mysql ignores Unicode byte order mark (BOM) characters at the beginning of input files. Previously, it read them and sent them to the server, resulting in a syntax error. Presence of a BOM does not cause mysql to change its default character set. To do that, invoke mysql with an option such as --default-character-set=utf8.

For more information about batch mode, see Section 3.5, “Using mysql in Batch Mode”.

4.5.1.6 mysql Tips

This section describes some techniques that can help you use mysql more effectively.

Input-Line Editing

mysql supports input-line editing, which enables you to modify the current input line in place or recall previous input lines. For example, the **left-arrow** and **right-arrow** keys move horizontally within the current input line, and the **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines. **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position. To enter the line, press **Enter**.

On Windows, the editing key sequences are the same as supported for command editing in console windows. On Unix, the key sequences depend on the input library used to build mysql (for example, the libedit or readline library).

Documentation for the libedit and readline libraries is available online. To change the set of key sequences permitted by a given input library, define key bindings in the library startup file. This is a file in your home directory: .editrc for libedit and .inputrc for readline.

For example, in libedit, **Control+W** deletes everything before the current cursor position and **Control+U** deletes the entire line. In readline, **Control+W** deletes the word before the cursor and **Control+U** deletes everything before the current cursor position. If mysql was built using libedit, a user who prefers the readline behavior for these two keys can put the following lines in the .editrc file (creating the file if necessary):

```
bind "^W" ed-delete-prev-word
bind "^U" vi-kill-line-prev
```

To see the current set of key bindings, temporarily put a line that says only bind at the end of .editrc. mysql will show the bindings when it starts.

Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
      sbj: UTF-8
      txt: >>>>> "Thimble" == Thimble Smith writes:

Thimble> Hi.  I think this is a good idea.  Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

Using the --safe-updates Option

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). It is helpful for cases when you might have issued a `DELETE FROM tbl_name` statement but forgotten the `WHERE` clause. Normally, such a statement deletes all rows from the table. With `--safe-updates`, you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the `--safe-updates` option, `mysql` issues the following statement when it connects to the MySQL server:

```
SET sql_safe_updates=1, sql_select_limit=1000, sql_max_join_size=1000000;
```

See Section 5.1.4, “Server System Variables”.

The `SET` statement has the following effects:

- You are not permitted to execute an `UPDATE` or `DELETE` statement unless you specify a key constraint in the `WHERE` clause or provide a `LIMIT` clause (or both). For example:

  ```
  UPDATE tbl_name SET not_key_column=val WHERE key_column=val;

  UPDATE tbl_name SET not_key_column=val LIMIT 1;
  ```

- The server limits all large `SELECT` results to 1,000 rows unless the statement includes a `LIMIT` clause.
- The server aborts multiple-table `SELECT` statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select_limit` and `--max_join_size` options:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

Disabling mysql Auto-Reconnect

If the `mysql` client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if `mysql`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

The `@a` user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have `mysql` terminate with an error if the connection has been lost, you can start the `mysql` client with the `--skip-reconnect` option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see Section 20.6.15, “Controlling Automatic Reconnection Behavior”.

4.5.2 `mysqladmin` — Client for Administering a MySQL Server

`mysqladmin` is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke `mysqladmin` like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

`mysqladmin` supports the following commands. Some of the commands take an argument following the command name.

- `create` *`db_name`*

  Create a new database named *`db_name`*.

- `debug`

  Tell the server to write debug information to the error log. Format and content of this information is subject to change.

- `drop` *`db_name`*

  Delete the database named *`db_name`* and all its tables.

- `extended-status`

This documentation is for an older version. If you're

Display the server status variables and their values.

- `flush-hosts`

  Flush all information in the host cache.

- `flush-logs`

  Flush all logs.

- `flush-privileges`

  Reload the grant tables (same as `reload`).

- `flush-status`

  Clear status variables.

- `flush-tables`

  Flush all tables.

- `flush-threads`

  Flush the thread cache.

- `kill` *`id,id,...`*

  Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

- `old-password` *`new_password`*

  This is like the `password` command but stores the password using the old (pre-4.1) password-hashing format. (See Section 6.1.2.4, “Password Hashing in MySQL”.)

- `password` *`new_password`*

  Set a new password. This changes the password to *`new_password`* for the account that you use with `mysqladmin` for connecting to the server. Thus, the next time you invoke `mysqladmin` (or any other client program) using the same account, you will need to specify the new password.

  If the *`new_password`* value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotation marks. On Windows, be sure to use double quotation marks rather than single quotation marks; single quotation marks are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

Caution

Do not use this command used if the server was started with the `--skip-grant-tables` option. No password change will be applied. This is true even if you precede the `password` command with `flush-privileges` on the same command line to re-enable the grant tables because the flush operation occurs after you connect. However, you can use `mysqladmin flush-privileges` to re-enable the grant table and then use a separate `mysqladmin password` command to change the password.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `ping`

  Check whether the server is available. The return status from `mysqladmin` is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as `Access denied`, because this means that the server is running but refused the connection, which is different from the server not running.

- `processlist`

  Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` option is given, the output is like that of `SHOW FULL PROCESSLIST`. (See Section 13.7.5.27, “SHOW PROCESSLIST Syntax”.)

- `reload`

  Reload the grant tables.

- `refresh`

  Flush all tables and close and open log files.

- `shutdown`

  Stop the server.

- `start-slave`

  Start replication on a slave server.

- `status`

  Display a short server status message.

- `stop-slave`

  Stop replication on a slave server.

- `variables`

  Display the server system variables and their values.

- `version`

  Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User  | Host      | db | Command | Time | State | Info             |
+----+-------+-----------+----+---------+------+-------+------------------+
| 51 | monty | localhost |    | Query   | 0    |       | show processlist |
+----+-------+-----------+----+---------+------+-------+------------------+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The `mysqladmin status` command result displays the following values:

- `Uptime`

This documentation is for an older version. If you're

The number of seconds the MySQL server has been running.

- `Threads`

  The number of active threads (clients).

- `Questions`

  The number of questions (queries) from clients since the server was started.

- `Slow queries`

  The number of queries that have taken more than `long_query_time` seconds. See Section 5.4.4, “The Slow Query Log”.

- `Opens`

  The number of tables the server has opened.

- `Flush tables`

  The number of `flush-*`, `refresh`, and `reload` commands the server has executed.

- `Open tables`

  The number of tables that currently are open.

- `Memory in use`

  The amount of memory allocated directly by `mysqld`. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

- `Maximum memory used`

  The maximum amount of memory allocated directly by `mysqld`. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

If you execute `mysqladmin shutdown` when connecting to a local server using a Unix socket file, `mysqladmin` waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

`mysqladmin` supports the following options, which can be specified on the command line or in the `[mysqladmin]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Table 4.4 `mysqladmin` Options

Format	Description	Introduced
--compress	Compress all information sent between client and server	
--connect_timeout	Number of seconds before connection timeout	
--count	Number of iterations to make for repeated command execution	
--debug	Write debugging log	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--defaults-group-suffix	Option group suffix value	5.0.10
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Connect to MySQL server on given host	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	
--pipe	On Windows, connect to server using named pipe	
--port	TCP/IP port number to use for connection	
--print-defaults	Print default options	
--protocol	Connection protocol to use	
--relative	Show the difference between the current and previous values when used with the --sleep option	
--shared-memory-base-name	The name of shared memory to use for shared-memory connections	
--shutdown_timeout	The maximum number of seconds to wait for server shutdown	
--silent	Silent mode	
--sleep	Execute commands repeatedly, sleeping for delay seconds in between	
--socket	For connections to localhost, the Unix socket file to use	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert	Path of file that contains X509 certificate in PEM format	
--ssl-cipher	List of permitted ciphers to use for connection encryption	
--ssl-key	Path of file that contains X509 key in PEM format	
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--vertical	Print query output rows vertically (one line per column value)	
--wait	If the connection cannot be established, wait and retry instead of aborting	

- `--help`, `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`dir_name`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--count=N`, `-c N`

  The number of iterations to make for repeated command execution if the `--sleep` option is given.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqladmin.trace`.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqladmin` normally reads the `[client]` and `[mysqladmin]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqladmin` also reads the `[client_other]` and `[mysqladmin_other]` groups. This option was added in MySQL 5.0.10.

- `--force`, `-f`

  Do not ask for confirmation for the `drop db_name` command. With multiple commands, continue even if an error occurs.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqladmin` prompts for one.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--relative`, `-r`

  Show the difference between the current and previous values when used with the `--sleep` option. This option works only with the `extended-status` command.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

- `--silent`, `-s`

  Exit silently if a connection to the server cannot be established.

- `--sleep=delay`, `-i delay`

  Execute commands repeatedly, sleeping for *`delay`* seconds in between. The `--count` option determines the number of iterations. If `--count` is not given, `mysqladmin` executes commands indefinitely until interrupted.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

- `--vertical`, `-E`

  Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=count]`, `-w[count]`

  If the connection cannot be established, wait and retry instead of aborting. If a *`count`* value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value` The `--set-variable` format is deprecated. syntax:

- `connect_timeout`

  The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

  The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

It is also possible to set variables by using `--var_name=value`. The `--set-variable` format is deprecated.

4.5.3 mysqlcheck — A Table Maintenance Program

The `mysqlcheck` client performs table maintenance: It checks, repairs, optimizes, or analyzes tables.

Each table is locked and therefore unavailable to other sessions while it is being processed, although for check operations, the table is locked with a `READ` lock only (see Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”, for more information about `READ` and `WRITE` locks). Table maintenance operations can be time-consuming, particularly for large tables. If you use the `--databases` or `--all-databases` option to process all tables in one or more databases, an invocation of `mysqlcheck` might take a long time. (This is also true for `mysql_upgrade` because that program invokes `mysqlcheck` to check all tables and repair them if necessary.)

`mysqlcheck` is similar in function to `myisamchk`, but works differently. The main operational difference is that `mysqlcheck` must be used when the `mysqld` server is running, whereas `myisamchk` should be used when it is not. The benefit of using `mysqlcheck` is that you do not have to stop the server to perform table maintenance.

`mysqlcheck` uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in Section 13.7.2, “Table Maintenance Statements”.

The `MyISAM` storage engine supports all four maintenance operations, so `mysqlcheck` can be used to perform any of them on `MyISAM` tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if `test.t` is a `MEMORY` table, an attempt to check it produces this result:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqlcheck test t
test.t
note     : The storage engine for the table doesn't support check
```

If mysqlcheck is unable to repair a table, see Section 2.19.4, “Rebuilding or Repairing Tables or Indexes” for manual table repair strategies. This will be the case, for example, for InnoDB tables, which can be checked with CHECK TABLE, but not repaired with REPAIR TABLE.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

There are three general ways to invoke mysqlcheck:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the --databases or --all-databases option, entire databases are checked.

mysqlcheck has a special feature compared to other client programs. The default behavior of checking tables (--check) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of mysqlcheck named mysqlrepair, or make a symbolic link to mysqlcheck named mysqlrepair. If you invoke mysqlrepair, it repairs tables.

The names shown in the following table can be used to change mysqlcheck default behavior.

Command	Meaning
mysqlrepair	The default option is --repair
mysqlanalyze	The default option is --analyze
mysqloptimize	The default option is --optimize

mysqlcheck supports the following options, which can be specified on the command line or in the [mysqlcheck] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Table 4.5 mysqlcheck Options

Format	Description	Introduced
--all-databases	Check all tables in all databases	
--all-in-1	Execute a single statement for each database that names all the tables from that database	
--analyze	Analyze the tables	
--auto-repair	If a checked table is corrupted, automatically fix it	
--character-sets-dir	Directory where character sets are installed	
--check	Check the tables for errors	
--check-only-changed	Check only tables that have changed since the last check	
--check-upgrade	Invoke CHECK TABLE with the FOR UPGRADE option	5.0.19
--compress	Compress all information sent between client and server	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--databases	Interpret all arguments as database names	
--debug	Write debugging log	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	5.0.10
--extended	Check and repair tables	
--fast	Check only tables that have not been closed properly	
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Connect to MySQL server on given host	
--medium-check	Do a check that is faster than an --extended operation	
--no-defaults	Read no option files	
--optimize	Optimize the tables	
--password	Password to use when connecting to server	
--pipe	On Windows, connect to server using named pipe	
--port	TCP/IP port number to use for connection	
--print-defaults	Print default options	
--protocol	Connection protocol to use	
--quick	The fastest method of checking	
--repair	Perform a repair that can fix almost anything except unique keys that are not unique	
--shared-memory-base-name	The name of shared memory to use for shared-memory connections	
--silent	Silent mode	
--socket	For connections to localhost, the Unix socket file to use	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert	Path of file that contains X509 certificate in PEM format	
--ssl-cipher	List of permitted ciphers to use for connection encryption	
--ssl-key	Path of file that contains X509 key in PEM format	
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23
--tables	Overrides the --databases or -B option	
--use-frm	For repair operations on MyISAM tables	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--all-databases`, `-A`

  Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--all-in-1`, `-1`

  Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze`, `-a`

  Analyze the tables.

- `--auto-repair`

  If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--check`, `-c`

  Check the tables for errors. This is the default operation.

- `--check-only-changed`, `-C`

  Check only tables that have changed since the last check or that have not been closed properly.

- `--check-upgrade`, `-g`

  Invoke `CHECK TABLE` with the `FOR UPGRADE` option to check tables for incompatibilities with the current version of the server. This option was added in MySQL 5.0.19.

- `--compress`

  Compress all information sent between the client and the server if both support compression.

- `--databases`, `-B`

  Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and any following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o`.

- `--default-character-set=`*`charset_name`*

This documentation is for an older version. If you're

Use *charset_name* as the default character set. See Section 10.5, “Character Set Configuration”.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlcheck normally reads the [client] and [mysqlcheck] groups. If the --defaults-group-suffix=_other option is given, mysqlcheck also reads the [client_other] and [mysqlcheck_other] groups. This option was added in MySQL 5.0.10.

- --extended, -e

  If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

  If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- --fast, -F

  Check only tables that have not been closed properly.

- --force, -f

  Continue even if an SQL error occurs.

- --host=*host_name*, -h *host_name*

  Connect to the MySQL server on the given host.

- --medium-check, -m

  Do a check that is faster than an --extended operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

- --optimize, -o

  Optimize the tables.

- --password[=*password*], -p[*password*]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqlcheck` prompts for one.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=`*`port_num`*, `-P` *`port_num`*

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--quick`, `-q`

  If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

  If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair`, `-r`

  Perform a repair that can fix almost anything except unique keys that are not unique.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

- `--silent`, `-s`

  Silent mode. Print only error messages.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- `--tables`

  Override the `--databases` or `-B` option. All name arguments following the option are regarded as table names.

- `--use-frm`

  For repair operations on `MyISAM` tables, get the table structure from the `.frm` file so that the table can be repaired even if the `.MYI` header is corrupted.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print information about the various stages of program operation.

- `--version`, `-V`

  Display version information and exit.

4.5.4 mysqldump — A Database Backup Program

The `mysqldump` client is a backup program originally written by Igor Romanenko. It can be used to dump a database or a collection of databases for backup or transfer to another SQL server (not necessarily a MySQL server). The dump typically contains SQL statements to create the table, populate it, or both. However, `mysqldump` can also be used to generate files in CSV, other delimited text, or XML format.

`mysqldump` requires at least the `SELECT` privilege for dumped tables, `SHOW VIEW` for dumped views, `SUPER` for dumped triggers, and `LOCK TABLES` if the `--single-transaction` option is not used. Certain options might require other privileges as noted in the option descriptions.

To reload a dump file, you must have the privileges required to execute the statements that it contains, such as the appropriate `CREATE` privileges for objects created by those statements.

If you are doing a backup on the server and your tables all are `MyISAM` tables, consider using the `mysqlhotcopy` instead because it can accomplish faster backups and faster restores. See Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”.

There are three general ways to invoke `mysqldump`:

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

If you do not name any tables following *`db_name`* or if you use the `--databases` or `--all-databases` option, entire databases are dumped.

`mysqldump` does not dump the `INFORMATION_SCHEMA` database. If you name that database explicitly on the command line, `mysqldump` silently ignores it.

To see a list of the options your version of `mysqldump` supports, execute `mysqldump --help`.

Some `mysqldump` options are shorthand for groups of other options:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Use of --opt is the same as specifying --add-drop-table, --add-locks, --create-options, --disable-keys, --extended-insert, --lock-tables, --quick, and --set-charset. All of the options that --opt stands for also are on by default because --opt is on by default.
- Use of --compact is the same as specifying --skip-add-drop-table, --skip-add-locks, --skip-comments, --skip-disable-keys, and --skip-set-charset options.

To reverse the effect of a group option, uses its --skip-*xxx* form (--skip-opt or --skip-compact). It is also possible to select only part of the effect of a group option by following it with options that enable or disable specific features. Here are some examples:

- To select the effect of --opt except for some features, use the --skip option for each feature. To disable extended inserts and memory buffering, use --opt --skip-extended-insert --skip-quick. (Actually, --skip-extended-insert --skip-quick is sufficient because --opt is on by default.)
- To reverse --opt for all features except index disabling and table locking, use --skip-opt --disable-keys --lock-tables.

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, --disable-keys --lock-tables --skip-opt would not have the intended effect; it is the same as --skip-opt by itself.

mysqldump can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the --quick option (or --opt, which enables --quick). The --opt option (and hence --quick) is enabled by default, so to enable memory buffering, use --skip-quick.

If you are using a recent version of mysqldump to generate a dump to be reloaded into a very old MySQL server, you should not use the --opt or --extended-insert option. Use --skip-opt instead.

Before MySQL 4.1.2, out-of-range numeric values such as -inf and inf, as well as NaN (not-a-number) values are dumped by mysqldump as NULL. You can see this using the following sample table:

```
mysql> CREATE TABLE t (f DOUBLE);
mysql> INSERT INTO t VALUES(1e+111111111111111111111);
mysql> INSERT INTO t VALUES(-1e111111111111111111111);
mysql> SELECT f FROM t;
+------+
| f    |
+------+
|  inf |
| -inf |
+------+
```

For this table, mysqldump produces the following data output:

```
--
-- Dumping data for table `t`
--

INSERT INTO t VALUES (NULL);
INSERT INTO t VALUES (NULL);
```

The significance of this behavior is that if you dump and restore the table, the new table has contents that differ from the original contents. This problem is fixed as of MySQL 4.1.2; you cannot insert inf in the table, so this mysqldump behavior is only relevant when you deal with old servers.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For additional information about mysqldump, see Section 7.4, “Using mysqldump for Backups”.

mysqldump supports the following options, which can be specified on the command line or in the [mysqldump] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Table 4.6 mysqldump Options

Format	Description	Introduced
--add-drop-database	Add DROP DATABASE statement before each CREATE DATABASE statement	
--add-drop-table	Add DROP TABLE statement before each CREATE TABLE statement	
--add-locks	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements	
--all-databases	Dump all tables in all databases	
--allow-keywords	Allow creation of column names that are keywords	
--character-sets-dir	Directory where character sets are installed	
--comments	Add comments to dump file	
--compact	Produce more compact output	
--compatible	Produce output that is more compatible with other database systems or with older MySQL servers	
--complete-insert	Use complete INSERT statements that include column names	
--compress	Compress all information sent between client and server	
--create-options	Include all MySQL-specific table options in CREATE TABLE statements	
--databases	Interpret all name arguments as database names	
--debug	Write debugging log	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	5.0.32
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	5.0.10
--delayed-insert	Write INSERT DELAYED statements rather than INSERT statements	
--delete-master-logs	On a master replication server, delete the binary logs after performing the dump operation	
--disable-keys	For each table, surround INSERT statements with statements to disable and enable keys	
--dump-date	Include dump date as "Dump completed on" comment if --comments is given	5.0.52
--extended-insert	Use multiple-row INSERT syntax	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--fields-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-escaped-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-optionally-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--first-slave	Deprecated; use --lock-all-tables instead	
--flush-logs	Flush MySQL server log files before starting dump	
--flush-privileges	Emit a FLUSH PRIVILEGES statement after dumping mysql database	
--force	Continue even if an SQL error occurs during a table dump	
--help	Display help message and exit	
--hex-blob	Dump binary columns using hexadecimal notation	
--host	Host to connect to (IP address or hostname)	
--ignore-table	Do not dump given table	
--insert-ignore	Write INSERT IGNORE rather than INSERT statements	
--lines-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--lock-all-tables	Lock all tables across all databases	
--lock-tables	Lock all tables before dumping them	
--log-error	Append warnings and errors to named file	5.0.42
--master-data	Write the binary log file name and position to the output	
--max_allowed_packet	Maximum packet length to send to or receive from server	
--net_buffer_length	Buffer size for TCP/IP and socket communication	
--no-autocommit	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements	
--no-create-db	Do not write CREATE DATABASE statements	
--no-create-info	Do not write CREATE TABLE statements that re-create each dumped table	
--no-data	Do not dump table contents	
--no-defaults	Read no option files	
--no-set-names	Same as --skip-set-charset	
--opt	Shorthand for --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset.	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--order-by-primary	Dump each table's rows sorted by its primary key, or by its first unique index	
--password	Password to use when connecting to server	
--pipe	On Windows, connect to server using named pipe	
--port	TCP/IP port number to use for connection	
--print-defaults	Print default options	
--protocol	Connection protocol to use	
--quick	Retrieve rows for a table from the server a row at a time	
--quote-names	Quote identifiers within backtick characters	
--result-file	Direct output to a given file	
--routines	Dump stored routines (procedures and functions) from dumped databases	5.0.13
--set-charset	Add SET NAMES default_character_set to output	
--shared-memory-base-name	The name of shared memory to use for shared-memory connections	
--single-transaction	Issue a BEGIN SQL statement before dumping data from server	
--skip-add-drop-table	Do not add a DROP TABLE statement before each CREATE TABLE statement	
--skip-add-locks	Do not add locks	
--skip-comments	Do not add comments to dump file	
--skip-compact	Do not produce more compact output	
--skip-disable-keys	Do not disable keys	
--skip-extended-insert	Turn off extended-insert	
--skip-opt	Turn off options set by --opt	
--skip-quick	Do not retrieve rows for a table from the server a row at a time	
--skip-quote-names	Do not quote identifiers	
--skip-set-charset	Do not write SET NAMES statement	
--skip-triggers	Do not dump triggers	5.0.11
--skip-tz-utc	Turn off tz-utc	5.0.15
--socket	For connections to localhost, the Unix socket file to use	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert	Path of file that contains X509 certificate in PEM format	
--ssl-cipher	List of permitted ciphers to use for connection encryption	
--ssl-key	Path of file that contains X509 key in PEM format	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23
--tab	Produce tab-separated data files	
--tables	Override --databases or -B option	
--triggers	Dump triggers for each dumped table	
--tz-utc	Add SET TIME_ZONE='+00:00' to dump file	5.0.15
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--where	Dump only rows selected by given WHERE condition	
--xml	Produce XML output	

- `--help`, `-?`

  Display a help message and exit.

- `--add-drop-database`

  Write a `DROP DATABASE` statement before each `CREATE DATABASE` statement. This option is typically used in conjunction with the `--all-databases` or `--databases` option because no `CREATE DATABASE` statements are written unless one of those options is specified.

- `--add-drop-table`

  Write a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-locks`

  Surround each table dump with `LOCK TABLES` and `UNLOCK TABLES` statements. This results in faster inserts when the dump file is reloaded. See Section 8.2.2.1, “Speed of INSERT Statements”.

- `--all-databases`, `-A`

  Dump all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--allow-keywords`

  Permit creation of column names that are keywords. This works by prefixing each column name with the table name.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--comments`, `-i`

  Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use `--skip-comments`.

- `--compact`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Produce more compact output. This option enables the `--skip-add-drop-table`, `--skip-add-locks`, `--skip-comments`, `--skip-disable-keys`, and `--skip-set-charset` options.

Note

Prior to MySQL 5.0.48, this option did not create valid SQL if the database dump contained views. The recreation of views requires the creation and removal of temporary tables and this option suppressed the removal of those temporary tables. As a workaround, use `--compact` with the `--add-drop-table` option and then manually adjust the dump file.

- `--compatible=`*`name`*

  Produce output that is more compatible with other database systems or with older MySQL servers. The value of *`name`* can be `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, or `no_field_options`. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See Section 5.1.7, “Server SQL Modes”.

  This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, `--compatible=oracle` does not map data types to Oracle types or use Oracle comment syntax.

  This option requires a server version of 4.1.0 or higher. With older servers, it does nothing.

- `--complete-insert`, `-c`

  Use complete `INSERT` statements that include column names.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--create-options`

  Include all MySQL-specific table options in the `CREATE TABLE` statements.

- `--databases`, `-B`

  Dump several databases. Normally, `mysqldump` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default value is `d:t:o,/tmp/mysqldump.trace`.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.0.32.

- `--default-character-set=`*`charset_name`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Use *charset_name* as the default character set. See Section 10.5, “Character Set Configuration”. If no character set is specified, mysqldump uses utf8.

This option has no effect for output data files produced by using the --tab option. See the description for that option.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqldump normally reads the [client] and [mysqldump] groups. If the --defaults-group-suffix=_other option is given, mysqldump also reads the [client_other] and [mysqldump_other] groups. This option was added in MySQL 5.0.10.

- --delayed-insert

  Write INSERT DELAYED statements rather than INSERT statements.

- --delete-master-logs

  On a master replication server, delete the binary logs by sending a PURGE BINARY LOGS statement to the server after performing the dump operation. This option automatically enables --master-data.

- --disable-keys, -K

  For each table, surround the INSERT statements with /*!40000 ALTER TABLE *tbl_name* DISABLE KEYS */; and /*!40000 ALTER TABLE *tbl_name* ENABLE KEYS */; statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for nonunique indexes of MyISAM tables. It has no effect for other tables.

- --dump-date

  If the --comments option is given, mysqldump produces a comment at the end of the dump of the following form:

```
-- Dump completed on DATE
```

  However, the date causes dump files taken at different times to appear to be different, even if the data are otherwise identical. --dump-date and --skip-dump-date control whether the date is added to the comment. The default is --dump-date (include the date in the comment). --skip-dump-date suppresses date printing. This option was added in MySQL 5.0.52.

- --extended-insert, -e

  Write INSERT statements using multiple-row syntax that includes several VALUES lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options are used with the `--tab` option and have the same meaning as the corresponding `FIELDS` clauses for `LOAD DATA INFILE`. See Section 13.2.6, “LOAD DATA INFILE Syntax”.

- `--first-slave`

  Deprecated. Use `--lock-all-tables` instead. `--first-slave` is removed in MySQL 5.5.

- `--flush-logs`, `-F`

  Flush the MySQL server log files before starting the dump. This option requires the `RELOAD` privilege. If you use this option in combination with the `--all-databases` option, the logs are flushed *for each database dumped*. The exception is when using `--lock-all-tables` or `--master-data`: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use `--flush-logs` together with either `--lock-all-tables` or `--master-data`.

- `--flush-privileges`

  Add a `FLUSH PRIVILEGES` statement to the dump output after dumping the `mysql` database. This option should be used any time the dump contains the `mysql` database and any other database that depends on the data in the `mysql` database for proper restoration. This option was added in MySQL 5.0.26.

- `--force`, `-f`

  Continue even if an SQL error occurs during a table dump.

  One use for this option is to cause `mysqldump` to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force`, `mysqldump` exits with an error message. With `--force`, `mysqldump` prints the error message, but it also writes an SQL comment containing the view definition to the dump output and continues executing.

- `--host=`*`host_name`*, `-h` *`host_name`*

  Dump data from the MySQL server on the given host. The default host is `localhost`.

- `--hex-blob`

  Dump binary columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected data types are `BINARY`, `VARBINARY`, and the `BLOB` types. As of MySQL 5.0.13, `BIT` columns are affected as well.

- `--ignore-table=`*`db_name.tbl_name`*

  Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times. This option also can be used to ignore views.

- `--insert-ignore`

  Write `INSERT IGNORE` statements rather than `INSERT` statements.

- `--lines-terminated-by=...`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This option is used with the `--tab` option and has the same meaning as the corresponding `LINES` clause for `LOAD DATA INFILE`. See Section 13.2.6, "LOAD DATA INFILE Syntax".

- `--lock-all-tables`, `-x`

  Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off `--single-transaction` and `--lock-tables`.

- `--lock-tables`, `-l`

  For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with `READ LOCAL` to permit concurrent inserts in the case of `MyISAM` tables. For transactional tables such as `InnoDB` and `BDB`, `--single-transaction` is a much better option than `--lock-tables` because it does not need to lock the tables at all.

  Because `--lock-tables` locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

- `--log-error=`*`file_name`*

  Log warnings and errors by appending them to the named file. The default is to do no logging. This option was added in MySQL 5.0.42.

- `--master-data[=`*`value`*`]`

  Use this option to dump a master replication server to produce a dump file that can be used to set up another server as a slave of the master. It causes the dump output to include a `CHANGE MASTER TO` statement that indicates the binary log coordinates (file name and position) of the dumped server. These are the master server coordinates from which the slave should start replicating after you load the dump file into the slave.

  If the option value is 2, the `CHANGE MASTER TO` statement is written as an SQL comment, and thus is informative only; it has no effect when the dump file is reloaded. If the option value is 1, the statement is not written as a comment and takes effect when the dump file is reloaded. If no option value is specified, the default value is 1.

  This option requires the `RELOAD` privilege and the binary log must be enabled.

  The `--master-data` option automatically turns off `--lock-tables`. It also turns on `--lock-all-tables`, unless `--single-transaction` also is specified, in which case, a global read lock is acquired only for a short time at the beginning of the dump (see the description for `--single-transaction`). In all cases, any action on logs happens at the exact moment of the dump.

  It is also possible to set up a slave by dumping an existing slave of the master. To do this, use the following procedure on the existing slave:

  1. Stop the slave's SQL thread and get its current status:

     ```
     mysql> STOP SLAVE SQL_THREAD;
     mysql> SHOW SLAVE STATUS;
     ```

  2. From the output of the `SHOW SLAVE STATUS` statement, the binary log coordinates of the master server from which the new slave should start replicating are the values of the `Relay_Master_Log_File` and `Exec_Master_Log_Pos` fields. Denote those values as *`file_name`* and *`file_pos`*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

3. Dump the slave server:

```
shell> mysqldump --master-data=2 --all-databases > dumpfile
```

Using `--master-data=2` works only if binary logging has been enabled on the slave. Otherwise, `mysqldump` fails with the error `Binlogging on server not active`. In this case you must handle any locking issues in another manner, using one or more of `--add-locks`, `--lock-tables`, `--lock-all-tables`, or `--single-transaction`, as required by your application and environment.

4. Restart the slave:

```
mysql> START SLAVE;
```

5. On the new slave, load the dump file:

```
shell> mysql < dumpfile
```

6. On the new slave, set the replication coordinates to those of the master server obtained earlier:

```
mysql> CHANGE MASTER TO
    -> MASTER_LOG_FILE = 'file_name', MASTER_LOG_POS = file_pos;
```

The `CHANGE MASTER TO` statement might also need other parameters, such as `MASTER_HOST` to point the slave to the correct master server host. Add any such parameters as necessary.

- `--no-autocommit`

  Enclose the `INSERT` statements for each dumped table within `SET autocommit = 0` and `COMMIT` statements.

- `--no-create-db`, `-n`

  Suppress the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` or `--all-databases` option is given.

- `--no-create-info`, `-t`

  Do not write `CREATE TABLE` statements that create each dumped table.

- `--no-data`, `-d`

  Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the `CREATE TABLE` statement for the table (for example, to create an empty copy of the table by loading the dump file).

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--no-set-names`, `-N`

  This has the same effect as `--skip-set-charset`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--opt`

  This option is shorthand. It is the same as specifying `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. It should give you a fast dump operation and produce a dump file that can be reloaded into a MySQL server quickly.

  The `--opt` option is enabled by default. Use `--skip-opt` to disable it. See the discussion at the beginning of this section for information about selectively enabling or disabling a subset of the options affected by `--opt`.

- `--order-by-primary`

  Dump each table's rows sorted by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a `MyISAM` table to be loaded into an `InnoDB` table, but will make the dump operation take considerably longer.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqldump` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--quick`, `-q`

  This option is useful for dumping large tables. It forces `mysqldump` to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- `--quote-names`, `-Q`

  Quote identifiers (such as database, table, and column names) within “`” characters. If the `ANSI_QUOTES` SQL mode is enabled, identifiers are quoted within “"” characters. This option is enabled by default. It can be disabled with `--skip-quote-names`, but this option should be given after any option such as `--compatible` that may enable `--quote-names`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--result-file=file_name`, `-r file_name`

  Direct output to the named file. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

  This option should be used on Windows to prevent newline "`\n`" characters from being converted to "`\r\n`" carriage return/newline sequences.

- `--routines`, `-R`

  Include stored routines (procedures and functions) for the dumped databases in the output. Use of this option requires the `SELECT` privilege for the `mysql.proc` table.

  The output generated by using `--routines` contains `CREATE PROCEDURE` and `CREATE FUNCTION` statements to create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps, so when the routines are reloaded, they are created with timestamps equal to the reload time.

  If you require routines to be created with their original timestamp attributes, do not use `--routines`. Instead, dump and reload the contents of the `mysql.proc` table directly, using a MySQL account that has appropriate privileges for the `mysql` database.

  This option was added in MySQL 5.0.13. Before that, stored routines are not dumped. Routine `DEFINER` values are not dumped until MySQL 5.0.20. This means that before 5.0.20, when routines are reloaded, they will be created with the definer set to the reloading user. If you require routines to be re-created with their original definer, dump and load the contents of the `mysql.proc` table directly as described earlier.

- `--set-charset`

  Write `SET NAMES default_character_set` to the output. This option is enabled by default. To suppress the `SET NAMES` statement, use `--skip-set-charset`.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

- `--single-transaction`

  This option sets the transaction isolation mode to `REPEATABLE READ` and sends a `START TRANSACTION` SQL statement to the server before dumping data. It is useful only with transactional tables such as `InnoDB` and `BDB`, because then it dumps the consistent state of the database at the time when `START TRANSACTION` was issued without blocking any applications.

  When using this option, you should keep in mind that only `InnoDB` tables are dumped in a consistent state. For example, any `MyISAM` or `MEMORY` tables dumped while using this option may still change state.

  While a `--single-transaction` dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: `ALTER TABLE`, `CREATE TABLE`, `DROP TABLE`, `RENAME TABLE`, `TRUNCATE TABLE`. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the `SELECT` that is performed by `mysqldump` to retrieve the table contents to obtain incorrect contents or fail.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The --single-transaction option and the --lock-tables option are mutually exclusive because LOCK TABLES causes any pending transactions to be committed implicitly.

This option is not supported for MySQL Cluster tables; the results cannot be guaranteed to be consistent due to the fact that the NDBCLUSTER storage engine supports only the READ_COMMITTED transaction isolation level. You should always use NDB backup and restore instead.

To dump large tables, combine the --single-transaction option with the --quick option.

- --skip-comments

  See the description for the --comments option.

- --skip-opt

  See the description for the --opt option.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- --tab=*dir_name*, -T *dir_name*

  Produce tab-separated text-format data files. For each dumped table, mysqldump creates a *tbl_name*.sql file that contains the CREATE TABLE statement that creates the table, and the server writes a *tbl_name*.txt file that contains its data. The option value is the directory in which to write the files.

  Note

  This option should be used only when mysqldump is run on the same machine as the mysqld server. Because the server creates files *.txt file in the directory that you specify, the directory must be writable by the server and the MySQL account that you use must have the FILE privilege. Because mysqldump creates *.sql in the same directory, it must be writable by your system login account.

  By default, the .txt data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the --fields-*xxx* and --lines-terminated-by options.

  Column values are dumped using the binary character set and the --default-character-set option is ignored. In effect, there is no character set conversion. If a table contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

- --tables

  Override the --databases or -B option. mysqldump regards all name arguments following the option as table names.

- --triggers

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Include triggers for each dumped table in the output. This option is enabled by default; disable it with `--skip-triggers`. This option was added in MySQL 5.0.11. Before that, triggers are not dumped.

- `--tz-utc`

  This option enables `TIMESTAMP` columns to be dumped and reloaded between servers in different time zones. `mysqldump` sets its connection time zone to UTC and adds `SET TIME_ZONE='+00:00'` to the dump file. Without this option, `TIMESTAMP` columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. `--tz-utc` also protects against changes due to daylight saving time. `--tz-utc` is enabled by default. To disable it, use `--skip-tz-utc`. This option was added in MySQL 5.0.15.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

- `--where='where_condition'`, `-w 'where_condition'`

  Dump only rows selected by the given `WHERE` condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

  Examples:

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

- `--xml`, `-X`

  Write dump output as well-formed XML.

  `NULL`, `'NULL'`, and Empty Values: For a column named *`column_name`*, the `NULL` value, an empty string, and the string value `'NULL'` are distinguished from one another in the output generated by this option as follows.

Value:	XML Representation:
`NULL` (*unknown value*)	`<field name="column_name" xsi:nil="true" />`
`''` (*empty string*)	`<field name="column_name"></field>`
`'NULL'` (*string value*)	`<field name="column_name">NULL</field>`

  Beginning with MySQL 5.0.26, the output from the `mysql` client when run using the `--xml` option also follows the preceding rules. (See Section 4.5.1.1, “mysql Options”.)

  Beginning with MySQL 5.0.40, XML output from `mysqldump` includes the XML namespace, as shown here:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabol</field>
<field name="Population">1780000</field>
</row>

...

<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

You can also set the following variables by using `--var_name=value` syntax:

- `max_allowed_packet`

  The maximum size of the buffer for client/server communication. The default is 24MB, the maximum is 1GB.

- `net_buffer_length`

  The initial size of the buffer for client/server communication. When creating multiple-row `INSERT` statements (as with the `--extended-insert` or `--opt` option), `mysqldump` creates rows up to `net_buffer_length` bytes long. If you increase this variable, ensure that the MySQL server `net_buffer_length` system variable has a value at least this large.

It is also possible to set variables by using `--var_name=value`. The `--set-variable` format is deprecated.

A common use of `mysqldump` is for making a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

You can load the dump file back into the server like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql db_name < backup-file.sql
```

Or like this:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the --all-databases option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, mysqldump provides a way of making an online backup:

```
shell> mysqldump --all-databases --master-data --single-transaction > all_databases.sql
```

This backup acquires a global read lock on all tables (using FLUSH TABLES WITH READ LOCK) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the MySQL server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see Section 5.4.3, “The Binary Log”) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2
            > all_databases.sql
```

The --master-data and --single-transaction options can be used simultaneously, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the InnoDB storage engine.

For more information on making backups, see Section 7.2, “Database Backup Methods”, and Section 7.3, “Example Backup and Recovery Strategy”.

If you encounter problems backing up views, please read the section that covers restrictions on views which describes a workaround for backing up views when this fails due to insufficient privileges. See Section C.4, “Restrictions on Views”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4.5.5 mysqlimport — A Data Import Program

The mysqlimport client provides a command-line interface to the LOAD DATA INFILE SQL statement. Most options to mysqlimport correspond directly to clauses of LOAD DATA INFILE syntax. See Section 13.2.6, “LOAD DATA INFILE Syntax”.

Invoke mysqlimport like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, mysqlimport strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents. For example, files named patient.txt, patient.text, and patient all would be imported into a table named patient.

mysqlimport supports the following options, which can be specified on the command line or in the [mysqlimport] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Table 4.7 mysqlimport Options

Format	Description	Introduced
--columns	This option takes a comma-separated list of column names as its value	
--compress	Compress all information sent between client and server	
--debug	Write debugging log	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	5.0.10
--delete	Empty the table before importing the text file	
--fields-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-escaped-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-optionally-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Connect to MySQL server on given host	
--ignore	See the description for the --replace option	
--ignore-lines	Ignore the first N lines of the data file	
--lines-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--local	Read input files locally from the client host	
--lock-tables	Lock all tables for writing before processing any text files	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--low-priority	Use LOW_PRIORITY when loading the table.	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	
--pipe	On Windows, connect to server using named pipe	
--port	TCP/IP port number to use for connection	
--print-defaults	Print default options	
--protocol	Connection protocol to use	
--replace	The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values	
--shared-memory-base-name	The name of shared memory to use for shared-memory connections	
--silent	Produce output only when errors occur	
--socket	For connections to localhost, the Unix socket file to use	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert	Path of file that contains X509 certificate in PEM format	
--ssl-cipher	List of permitted ciphers to use for connection encryption	
--ssl-key	Path of file that contains X509 key in PEM format	
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--columns=`*`column_list`*, `-c` *`column_list`*

  This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match data file columns with table columns.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--debug[=`*`debug_options`*`]`, `-#` `[`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlimport` normally reads the `[client]` and `[mysqlimport]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlimport` also reads the `[client_other]` and `[mysqlimport_other]` groups. This option was added in MySQL 5.0.10.

- `--delete`, `-D`

  Empty the table before importing the text file.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See Section 13.2.6, “LOAD DATA INFILE Syntax”.

- `--force`, `-f`

  Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force`, `mysqlimport` exits if a table does not exist.

- `--host=host_name`, `-h host_name`

  Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore`, `-i`

  See the description for the `--replace` option.

- `--ignore-lines=N`

  Ignore the first *`N`* lines of the data file.

- `--lines-terminated-by=...`

  This option has the same meaning as the corresponding clause for `LOAD DATA INFILE`. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"`. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See Section 13.2.6, “LOAD DATA INFILE Syntax”.

- `--local`, `-L`

  Read input files locally from the client host.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--lock-tables`, `-l`

  Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--low-priority`

  Use `LOW_PRIORITY` when loading the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--password[=`*`password`*`]`, `-p[`*`password`*`]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqlimport` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=`*`port_num`*, `-P` *`port_num`*

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--replace`, `-r`

  The `--replace` and `--ignore` options control handling of input rows that duplicate existing rows on unique key values. If you specify `--replace`, new rows replace existing rows that have the same unique key value. If you specify `--ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--silent`, `-s`

  Silent mode. Produce output only when errors occur.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

  Some options, such as `--opt`, automatically enable `--lock-tables`. If you want to override this, use `--skip-lock-tables` at the end of the option list.

Here is a sample session that demonstrates use of `mysqlimport`:

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
shell> od -c imptest.txt
0000000   1   0   0  \t   M   a   x       S   y   d   o   w  \n   1   0
0000020   1  \t   C   o   u   n   t       D   r   a   c   u   l   a  \n
0000040
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+--------------+
| id   | n            |
+------+--------------+
|  100 | Max Sydow    |
|  101 | Count Dracula |
+------+--------------+
```

4.5.6 mysqlshow — Display Database, Table, and Column Information

The `mysqlshow` client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysqlshow provides a command-line interface to several SQL SHOW statements. See Section 13.7.5, “SHOW Syntax”. The same information can be obtained by using those statements directly. For example, you can issue them from the mysql client program.

Invoke mysqlshow like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (“*”, “?”, “%”, or “_”), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. “*” and “?” characters are converted into SQL “%” and “_” wildcard characters. This might cause some confusion when you try to display the columns for a table with a “_” in the name, because in this case, mysqlshow shows you only the table names that match the pattern. This is easily fixed by adding an extra “%” last on the command line as a separate argument.

mysqlshow supports the following options, which can be specified on the command line or in the [mysqlshow] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Table 4.8 mysqlshow Options

Format	Description	Introduced
--compress	Compress all information sent between client and server	
--count	Show the number of rows per table	5.0.6
--debug	Write debugging log	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	5.0.10
--help	Display help message and exit	
--host	Connect to MySQL server on given host	
--keys	Show table indexes	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	
--pipe	On Windows, connect to server using named pipe	
--port	TCP/IP port number to use for connection	
--print-defaults	Print default options	
--protocol	Connection protocol to use	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--shared-memory-base-name	The name of shared memory to use for shared-memory connections	
--show-table-type	Show a column indicating the table type	5.0.4
--socket	For connections to localhost, the Unix socket file to use	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert	Path of file that contains X509 certificate in PEM format	
--ssl-cipher	List of permitted ciphers to use for connection encryption	
--ssl-key	Path of file that contains X509 key in PEM format	
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23
--status	Display extra information about each table	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--count`

  Show the number of rows per table. This can be slow for non-`MyISAM` tables. This option was added in MySQL 5.0.6.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o`.

- `--default-character-set=`*`charset_name`*

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=`*`file_name`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, `mysqlshow` normally reads the `[client]` and `[mysqlshow]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlshow` also reads the `[client_other]` and `[mysqlshow_other]` groups. This option was added in MySQL 5.0.10.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--keys`, `-k`

  Show table indexes.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, `mysqlshow` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--show-table-type`, `-t`

  Show a column indicating the table type, as in `SHOW FULL TABLES`. The type is `BASE TABLE` or `VIEW`. This option was added in MySQL 5.0.4.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- `--status`, `-i`

  Display extra information about each table.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version`, `-V`

  Display version information and exit.

4.6 MySQL Administrative and Utility Programs

This section describes administrative programs and programs that perform miscellaneous utility operations.

4.6.1 `innochecksum` — Offline InnoDB File Checksum Utility

`innochecksum` prints checksums for `InnoDB` files. This tool reads an `InnoDB` tablespace file, calculates the checksum for each page, compares the calculated checksum to the stored checksum, and reports mismatches, which indicate damaged pages. It was originally developed to speed up verifying the integrity of tablespace files after power outages but can also be used after file copies. Because checksum mismatches will cause `InnoDB` to deliberately shut down a running server, it can be preferable to use this tool rather than waiting for a server in production usage to encounter the damaged pages.

`innochecksum` cannot be used on tablespace files that the server already has open. For such files, you should use `CHECK TABLE` to check tables within the tablespace.

If checksum mismatches are found, you would normally restore the tablespace from backup or start the server and attempt to use `mysqldump` to make a backup of the tables within the tablespace.

Invoke `innochecksum` like this:

```
shell> innochecksum [options] file_name
```

`innochecksum` supports the following options. For options that refer to page numbers, the numbers are zero-based.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `-c`

  Print a count of the number of pages in the file.

- `-d`

  Debug mode; prints checksums for each page.

- `-e num`

  End at this page number.

- `-p num`

  Check only this page number.

- `-s num`

  Start at this page number.

- `-v`

  Verbose mode; print a progress indicator every five seconds.

4.6.2 myisam_ftdump — Display Full-Text Index information

`myisam_ftdump` displays information about `FULLTEXT` indexes in `MyISAM` tables. It reads the `MyISAM` index file directly, so it must be run on the server host where the table is located. Before using `myisam_ftdump`, be sure to issue a `FLUSH TABLES` statement first if the server is running.

`myisam_ftdump` scans and dumps the entire index, which is not particularly fast. On the other hand, the distribution of words changes infrequently, so it need not be run often.

Invoke `myisam_ftdump` like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The *`tbl_name`* argument should be the name of a `MyISAM` table. You can also specify a table by naming its index file (the file with the `.MYI` suffix). If you do not invoke `myisam_ftdump` in the directory where the table files are located, the table or index file name must be preceded by the path name to the table's database directory. Index numbers begin with 0.

Example: Suppose that the `test` database contains a table named `mytexttable` that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
) ENGINE=MyISAM;
```

The index on `id` is index 0 and the `FULLTEXT` index on `txt` is index 1. If your working directory is the `test` database directory, invoke `myisam_ftdump` as follows:

```
shell> myisam_ftdump mytexttable 1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the path name to the `test` database directory is `/usr/local/mysql/data/test`, you can also specify the table name argument using that path name. This is useful if you do not invoke `myisam_ftdump` in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

You can use `myisam_ftdump` to generate a list of index entries in order of frequency of occurrence like this:

```
shell> myisam_ftdump -c mytexttable 1 | sort -r
```

`myisam_ftdump` supports the following options:

- `--help`, `-h -?`

  Display a help message and exit.

- `--count`, `-c`

  Calculate per-word statistics (counts and global weights).

- `--dump`, `-d`

  Dump the index, including data offsets and word weights.

- `--length`, `-l`

  Report the length distribution.

- `--stats`, `-s`

  Report global index statistics. This is the default operation if no other operation is specified.

- `--verbose`, `-v`

  Verbose mode. Print more output about what the program does.

4.6.3 `myisamchk` — MyISAM Table-Maintenance Utility

The `myisamchk` utility gets information about your database tables or checks, repairs, or optimizes them. `myisamchk` works with `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.7.2.3, “CHECK TABLE Syntax”, and Section 13.7.2.6, “REPAIR TABLE Syntax”.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Invoke `myisamchk` like this:

```
shell> myisamchk [options] tbl_name ...
```

The *`options`* specify what you want `myisamchk` to do. They are described in the following sections. You can also get a list of options by invoking `myisamchk --help`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

With no options, myisamchk simply checks your table as the default operation. To get more information or to tell myisamchk to take corrective action, specify options as described in the following discussion.

tbl_name is the database table you want to check or repair. If you run myisamchk somewhere other than in the database directory, you must specify the path to the database directory, because myisamchk has no idea where the database is located. In fact, myisamchk does not actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the myisamchk command line if you wish. You can also specify a table by naming its index file (the file with the .MYI suffix). This enables you to specify all tables in a directory by using the pattern *.MYI. For example, if you are in a database directory, you can check all the MyISAM tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all MyISAM tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \
          --key_buffer_size=64M --sort_buffer_size=64M \
          --read_buffer_size=1M --write_buffer_size=1M \
          /path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with myisamchk, see Section 4.6.3.6, “myisamchk Memory Usage”.

For additional information about using myisamchk, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Important

You must ensure that no other program is using the tables while you are running myisamchk. The most effective means of doing so is to shut down the MySQL server while running myisamchk, or to lock all tables that myisamchk is being used on.

Otherwise, when you run myisamchk, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

> This means that you are trying to check a table that has been updated by another program (such as the mysqld server) that hasn't yet closed the file or that has died without closing the file properly, which can sometimes lead to the corruption of one or more MyISAM tables.
>
> If mysqld is running, you must force it to flush any table modifications that are still buffered in memory by using FLUSH TABLES. You should then ensure that no one is using the tables while you are running myisamchk
>
> However, the easiest way to avoid this problem is to use CHECK TABLE instead of myisamchk to check tables. See Section 13.7.2.3, "CHECK TABLE Syntax".

myisamchk supports the following options, which can be specified on the command line or in the [myisamchk] group of an option file. For information about option files used by MySQL programs, see Section 4.2.6, "Using Option Files".

Table 4.9 myisamchk Options

Format	Description	Introduced	Removed
--analyze	Analyze the distribution of key values		
--backup	Make a backup of the .MYD file as file_name-time.BAK		
--block-search	Find the record that a block at the given offset belongs to		
--check	Check the table for errors		
--check-only-changed	Check only tables that have changed since the last check		
--correct-checksum	Correct the checksum information for the table		
--data-file-length	Maximum length of the data file (when re-creating data file when it is full)		
--debug	Write debugging log		
--decode_bits	Decode_bits		
--defaults-extra-file	Read named option file in addition to usual option files		
--defaults-file	Read only named option file		
--defaults-group-suffix	Option group suffix value	5.0.10	
--description	Print some descriptive information about the table		
--extend-check	Do very thorough table check or repair that tries to recover every possible row from the data file		
--fast	Check only tables that haven't been closed properly		
--force	Do a repair operation automatically if myisamchk finds any errors in the table		
--force	Overwrite old temporary files. For use with the -r or -o option		
--ft_max_word_len	Maximum word length for FULLTEXT indexes		
--ft_min_word_len	Minimum word length for FULLTEXT indexes		
--ft_stopword_file	Use stopwords from this file instead of built-in list		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced	Removed
--HELP	Display help message and exit		
--help	Display help message and exit		
--information	Print informational statistics about the table that is checked		
--key_buffer_size	Size of buffer used for index blocks for MyISAM tables		
--keys-used	A bit-value that indicates which indexes to update		
--max-record-length	Skip rows larger than the given length if myisamchk cannot allocate memory to hold them		
--medium-check	Do a check that is faster than an --extend-check operation		
--myisam_block_size	Block size to be used for MyISAM index pages		
--no-defaults	Read no option files		
--parallel-recover	Uses the same technique as -r and -n, but creates all the keys in parallel, using different threads (beta)		
--print-defaults	Print default options		
--quick	Achieve a faster repair by not modifying the data file.		
--read_buffer_size	Each thread that does a sequential scan allocates a buffer of this size for each table it scans		
--read-only	Do not mark the table as checked		
--recover	Do a repair that can fix almost any problem except unique keys that aren't unique		
--safe-recover	Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found		
--set-auto-increment	Force AUTO_INCREMENT numbering for new records to start at the given value		
--set-character-set	Change the character set used by the table indexes		5.0.3
--set-collation	Specify the collation to use for sorting table indexes	5.0.3	
--silent	Silent mode		
--sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE		
--sort-index	Sort the index tree blocks in high-low order		
--sort_key_blocks	sort_key_blocks		
--sort-records	Sort records according to a particular index		
--sort-recover	Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large		
--stats_method	Specifies how MyISAM index statistics collection code should treat NULLs		
--tmpdir	Path of the directory to be used for storing temporary files		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced	Removed
--unpack	Unpack a table that was packed with myisampack		
--update-state	Store information in the .MYI file to indicate when the table was checked and whether the table crashed		
--verbose	Verbose mode		
--version	Display version information and exit		
--write_buffer_size	Write buffer size		

4.6.3.1 myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by `myisamchk`. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help`, `-?`

  Display a help message and exit. Options are grouped by type of operation.

- `--HELP`, `-H`

  Display a help message and exit. Options are presented in a single list.

- `--debug=`*`debug_options`*, `-#` *`debug_options`*

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o,/tmp/myisamchk.trace`.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=`*`file_name`*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=`*`str`*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `myisamchk` normally reads the `[myisamchk]` group. If the `--defaults-group-suffix=_other` option is given, `myisamchk` also reads the `[myisamchk_other]` group. This option was added in MySQL 5.0.10.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--silent`, `-s`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `myisamchk` very silent.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.

- `--version`, `-V`

  Display version information and exit.

- `--wait`, `-w`

  Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running `mysqld` with external locking disabled, the table can be locked only by another `myisamchk` command.

You can also set the following variables by using `--var_name=value` syntax:

Variable	Default Value
`decode_bits`	9
`ft_max_word_len`	version-dependent
`ft_min_word_len`	4
`ft_stopword_file`	built-in list
`key_buffer_size`	523264
`myisam_block_size`	1024
`read_buffer_size`	262136
`sort_buffer_size`	2097144
`sort_key_blocks`	16
`stats_method`	nulls_unequal
`write_buffer_size`	262136

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

The possible `myisamchk` variables and their default values can be examined with `myisamchk --help`:

`sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover`.

`key_buffer_size` is used when you are checking the table with `--extend-check` or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover`.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for `CHAR`, `VARCHAR`, or `TEXT` columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force `myisamchk` to repair by sorting, you can use the `--sort-recover` option.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

`myisam_block_size` is the size used for index blocks.

`stats_method` influences how `NULL` values are treated for index statistics collection when the `--analyze` option is given. It acts like the `myisam_stats_method` system variable. For more information, see the description of `myisam_stats_method` in Section 5.1.4, “Server System Variables”, and Section 8.3.7, “MyISAM Index Statistics Collection”. The `stats_method` method was added in MySQL 5.0.14. For older versions, the statistics collection method is equivalent to `nulls_equal`.

`ft_min_word_len` and `ft_max_word_len` indicate the minimum and maximum word length for `FULLTEXT` indexes. `ft_stopword_file` names the stopword file. These need to be set under the following circumstances.

If you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, you can place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE`. These statements are performed by the server, which knows the proper full-text parameter values to use.

4.6.3.2 myisamchk Check Options

`myisamchk` supports the following options for table checking operations:

- `--check`, `-c`

  Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- `--check-only-changed`, `-C`

  Check only tables that have changed since the last check.

- `--extend-check`, `-e`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, `myisamchk` or `myisamchk --medium-check` should be able to determine whether there are any errors in the table.

If you are using `--extend-check` and have plenty of memory, setting the `key_buffer_size` variable to a large value helps the repair operation run faster.

See also the description of this option under table repair options.

For a description of the output format, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- `--fast`, `-F`

  Check only tables that haven't been closed properly.

- `--force`, `-f`

  Do a repair operation automatically if `myisamchk` finds any errors in the table. The repair type is the same as that specified with the `--recover` or `-r` option.

- `--information`, `-i`

  Print informational statistics about the table that is checked.

- `--medium-check`, `-m`

  Do a check that is faster than an `--extend-check` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only`, `-T`

  Do not mark the table as checked. This is useful if you use `myisamchk` to check a table that is in use by some other application that does not use locking, such as `mysqld` when run with external locking disabled.

- `--update-state`, `-U`

  Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed` option, but you shouldn't use this option if the `mysqld` server is using the table and you are running it with external locking disabled.

4.6.3.3 myisamchk Repair Options

`myisamchk` supports the following options for table repair operations (operations performed when an option such as `--recover` or `--safe-recover` is given):

- `--backup`, `-B`

  Make a backup of the `.MYD` file as *`file_name-time`*`.BAK`

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--correct-checksum`

  Correct the checksum information for the table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--data-file-length=len`, `-D len`

  The maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check`, `-e`

  Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Do not use this option unless you are desperate.

  See also the description of this option under table checking options.

  For a description of the output format, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- `--force`, `-f`

  Overwrite old intermediate files (files with names like *`tbl_name`*`.TMD`) instead of aborting.

- `--keys-used=val`, `-k val`

  For `myisamchk`, the option value is a bit-value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r`.

- `--no-symlinks`, `-l`

  Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=len`

  Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them.

- `--parallel-recover`, `-p`

  Use the same technique as `-r` and `-n`, but create all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick`, `-q`

  Achieve a faster repair by modifying only the index file, not the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover`, `-r`

  Do a repair that can fix almost any problem except unique keys that are not unique (which is an extremely unlikely error with `MyISAM` tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` only if `myisamchk` reports that the table cannot be recovered using `--recover`. (In the unlikely case that `--recover` fails, the data file remains intact.)

  If you have lots of memory, you should increase the value of `sort_buffer_size`.

- `--safe-recover`, `-o`

  Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover`, but can handle

This documentation is for an older version. If you're
This documentation is for an older version. If you're

a couple of very unlikely cases that `--recover` cannot. This recovery method also uses much less disk space than `--recover`. Normally, you should repair first using `--recover`, and then with `--safe-recover` only if `--recover` fails.

If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-character-set=name`

  Change the character set used by the table indexes. This option was replaced by `--set-collation` in MySQL 5.0.3.

- `--set-collation=name`

  Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name. This option was added in MySQL 5.0.3.

- `--sort-recover`, `-n`

  Force `myisamchk` to use sorting to resolve the keys even if the temporary files would be very large.

- `--tmpdir=dir_name`, `-t dir_name`

  The path of the directory to be used for storing temporary files. If this is not set, `myisamchk` uses the value of the `TMPDIR` environment variable. `--tmpdir` can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (“`:`”) on Unix and the semicolon (“`;`”) on Windows, NetWare, and OS/2.

- `--unpack`, `-u`

  Unpack a table that was packed with `myisampack`.

4.6.3.4 Other myisamchk Options

`myisamchk` supports the following options for actions other than table checks and repairs:

- `--analyze`, `-a`

  Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a `myisamchk --description --verbose tbl_name` command or the `SHOW INDEX FROM tbl_name` statement.

- `--block-search=offset`, `-b offset`

  Find the record that a block at the given offset belongs to.

- `--description`, `-d`

  Print some descriptive information about the table. Specifying the `--verbose` option once or twice produces additional information. See Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- `--set-auto-increment[=value]`, `-A[value]`

  Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If `value` is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- `--sort-index`, `-S`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N`, `-R N`

  Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

  If keys are not packed (`PACK_KEYS=0`), they have the same length, so when `myisamchk` sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), `myisamchk` must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

4.6.3.5 Obtaining Table Information with myisamchk

To obtain a description of a `MyISAM` table or statistics about it, use the commands shown here. The output from these commands is explained later in this section.

- `myisamchk -d tbl_name`

  Runs `myisamchk` in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, `myisamchk` may report an error for a table that is updated while it runs. However, because `myisamchk` does not change the table in describe mode, there is no risk of destroying data.

- `myisamchk -dv tbl_name`

  Adding `-v` runs `myisamchk` in verbose mode so that it produces more information about the table. Adding `-v` a second time produces even more information.

- `myisamchk -eis tbl_name`

  Shows only the most important information from a table. This operation is slow because it must read the entire table.

- `myisamchk -eiv tbl_name`

  This is like `-eis`, but tells you what is being done.

The *`tbl_name`* argument can be either the name of a `MyISAM` table or the name of its index file, as described in Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”. Multiple *`tbl_name`* arguments can be given.

Suppose that a table named `person` has the following structure. (The `MAX_ROWS` table option is included so that in the example output from `myisamchk` shown later, some values are smaller and fit the output format more easily.)

```
CREATE TABLE person
(
  id         INT NOT NULL AUTO_INCREMENT,
  last_name  VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  birth      DATE,
  death      DATE,
  PRIMARY KEY (id),
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  INDEX (last_name, first_name),
  INDEX (birth)
) MAX_ROWS = 1000000;
```

Suppose also that the table has these data and index file sizes:

```
-rw-rw----  1 mysql  mysql  9347072 Aug 19 11:47 person.MYD
-rw-rw----  1 mysql  mysql  6066176 Aug 19 11:47 person.MYI
```

Example of `myisamchk -dvv` output:

```
MyISAM file:         person
Record format:       Packed
Character set:       latin1_swedish_ci (8)
File-version:        1
Creation time:       2009-08-19 16:47:41
Recover time:        2009-08-19 16:47:56
Status:              checked,analyzed,optimized keys
Auto increment key:             1  Last value:               306688
Data records:              306688  Deleted blocks:                0
Datafile parts:            306688  Deleted data:                  0
Datafile pointer (bytes):       4  Keyfile pointer (bytes):       3
Datafile length:          9347072  Keyfile length:          6066176
Max datafile length:    4294967294  Max keyfile length:   17179868159
Recordlength:                 54

table description:
Key Start Len Index   Type                 Rec/key         Root  Blocksize
1   2     4   unique  long                       1        99328       1024
2   6     20  multip. varchar prefix           512      3563520       1024
    27    20          varchar                  512
3   48    3   multip. uint24 NULL           306688      6065152       1024

Field Start Length Nullpos Nullbit Type
1     1     1
2     2     4                     no zeros
3     6     21                    varchar
4     27    21                    varchar
5     48    3      1       1      no zeros
6     51    3      1       2      no zeros
```

Explanations for the types of information `myisamchk` produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous, as are “field” and “column.”

The initial part of the table description contains these values:

- `MyISAM file`

  Name of the `MyISAM` (index) file.

- `Record format`

  The format used to store table rows. The preceding examples use `Fixed length`. Other possible values are `Compressed` and `Packed`. (`Packed` corresponds to what `SHOW TABLE STATUS` reports as `Dynamic`.)

- `Chararacter set`

  The table default character set.

- `File-version`

  Version of `MyISAM` format. Always 1.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Creation time`

  When the data file was created.

- `Recover time`

  When the index/data file was last reconstructed.

- `Status`

  Table status flags. Possible values are `crashed`, `open`, `changed`, `analyzed`, `optimized keys`, and `sorted index pages`.

- `Auto increment key`, `Last value`

  The key number associated the table's `AUTO_INCREMENT` column, and the most recently generated value for this column. These fields do not appear if there is no such column.

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Datafile parts`

  For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as `Data records`.

- `Deleted data`

  How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Datafile pointer`

  The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- `Keyfile pointer`

  The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- `Max datafile length`

  How long the table data file can become, in bytes.

- `Max keyfile length`

  How long the table index file can become, in bytes.

- `Recordlength`

  How much space each row takes, in bytes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `table description` part of the output includes a list of all keys in the table. For each key, `myisamchk` displays some low-level information:

- `Key`

  This key's number. This value is shown only for the first column of the key. If this value is missing, the line corresponds to the second or later column of a multiple-column key. For the table shown in the example, there are two `table description` lines for the second index. This indicates that it is a multiple-part index with two parts.

- `Start`

  Where in the row this portion of the index starts.

- `Len`

  How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column. The total length of a multiple-part key is the sum of the `Len` values for all key parts.

- `Index`

  Whether a key value can exist multiple times in the index. Possible values are `unique` or `multip.` (multiple).

- `Type`

  What data type this portion of the index has. This is a `MyISAM` data type with the possible values `packed`, `stripped`, or `empty`.

- `Root`

  Address of the root index block.

- `Blocksize`

  The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- `Rec/key`

  This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `myisamchk -a`. If this is not updated at all, a default value of 30 is given.

The last part of the output provides information about each column:

- `Field`

  The column number.

- `Start`

  The byte position of the column within table rows.

- `Length`

  The length of the column in bytes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Nullpos`, `Nullbit`

  For columns that can be `NULL`, `MyISAM` stores `NULL` values as a flag in a byte. Depending on how many nullable columns there are, there can be one or more bytes used for this purpose. The `Nullpos` and `Nullbit` values, if nonempty, indicate which byte and bit contains that flag indicating whether the column is `NULL`.

  The position and number of bytes used to store `NULL` flags is shown in the line for field 1. This is why there are six `Field` lines for the `person` table even though it has only five columns.

- `Type`

  The data type. The value may contain any of the following descriptors:

  - `constant`

    All rows have the same value.

  - `no endspace`

    Do not store endspace.

  - `no endspace, not_always`

    Do not store endspace and do not do endspace compression for all values.

  - `no endspace, no empty`

    Do not store endspace. Do not store empty values.

  - `table-lookup`

    The column was converted to an `ENUM`.

  - `zerofill(N)`

    The most significant *`N`* bytes in the value are always 0 and are not stored.

  - `no zeros`

    Do not store zeros.

  - `always zero`

    Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

The `Huff tree` and `Bits` fields are displayed if the table has been compressed with `myisampack`. See Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”, for an example of this information.

Example of `myisamchk -eiv` output:

This documentation is for an older version. If you're

```
Checking MyISAM file: person
Data records:  306688   Deleted blocks:       0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  98%  Packed:    0%  Max levels:  3
- check data record references index: 2
Key:  2:  Keyblocks used:  99%  Packed:   97%  Max levels:  3
- check data record references index: 3
Key:  3:  Keyblocks used:  98%  Packed:  -14%  Max levels:  3
Total:    Keyblocks used:  98%  Packed:   89%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records:            306688  M.recordlength:       25  Packed:            83%
Recordspace used:       97% Empty space:           2% Blocks/Record:   1.00
Record blocks:      306688  Delete blocks:         0
Record data:       7934464  Deleted data:          0
Lost space:         256512  Linkdata:        1156096

User time 43.08, System time 1.68
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
Blocks in 0 out 7, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 0
Maximum memory usage: 1046926 bytes (1023k)
```

`myisamchk -eiv` output includes the following information:

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Key`

  The key number.

- `Keyblocks used`

  What percentage of the keyblocks are used. When a table has just been reorganized with `myisamchk`, the values are very high (very near theoretical maximum).

- `Packed`

  MySQL tries to pack key values that have a common suffix. This can only be used for indexes on `CHAR` and `VARCHAR` columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the preceding example, the second key is 40 bytes long and a 97% reduction in space is achieved.

- `Max levels`

  How deep the B-tree for this key is. Large tables with long key values get high values.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Records`

  How many rows are in the table.

- `M.recordlength`

  The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- `Packed`

  MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

  What percentage of the data file is used.

- `Empty space`

  What percentage of the data file is unused.

- `Blocks/Record`

  Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See Section 7.6.4, “MyISAM Table Optimization”.

- `Recordblocks`

  How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- `Deleteblocks`

  How many blocks (links) are deleted.

- `Recorddata`

  How many bytes in the data file are used.

- `Deleted data`

  How many bytes in the data file are deleted (unused).

- `Lost space`

  If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- `Linkdata`

  When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

4.6.3.6 myisamchk Memory Usage

Memory allocation is important when you run `myisamchk`. `myisamchk` uses no more memory than its memory-related variables are set to. If you are going to use `myisamchk` on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get `myisamchk` to operate faster. For example, if you have more than

This documentation is for an older version. If you're
This documentation is for an older version. If you're

512MB RAM available, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --sort_buffer_size=256M \
           --key_buffer_size=512M \
           --read_buffer_size=64M \
           --write_buffer_size=64M ...
```

Using `--sort_buffer_size=16M` is probably enough for most cases.

Be aware that `myisamchk` uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory file system, out of memory errors can easily occur. If this happens, run `myisamchk` with the `--tmpdir=dir_name` option to specify a directory located on a file system that has more space.

When performing repair operations, `myisamchk` also needs a lot of disk space:

- Twice the size of the data file (the original file and a copy). This space is not needed if you do a repair with `--quick`; in this case, only the index file is re-created. *This space must be available on the same file system as the original data file*, as the copy is created in the same directory as the original.
- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space must be available on the same file system as the original data file.
- When using `--recover` or `--sort-recover` (but not when using `--safe-recover`), you need space on disk for sorting. This space is allocated in the temporary directory (specified by `TMPDIR` or `--tmpdir=dir_name`). The following formula yields the amount of space required:

  ```
  (largest_key + row_pointer_length) * number_of_rows * 2
  ```

  You can check the length of the keys and the *`row_pointer_length`* with `myisamchk -dv tbl_name` (see Section 4.6.3.5, “Obtaining Table Information with myisamchk”). The *`row_pointer_length`* and *`number_of_rows`* values are the `Datafile pointer` and `Data records` values in the table description. To determine the *`largest_key`* value, check the `Key` lines in the table description. The `Len` column indicates the number of bytes for each key part. For a multiple-column index, the key size is the sum of the `Len` values for all key parts.

If you have a problem with disk space during repair, you can try `--safe-recover` instead of `--recover`.

4.6.4 `myisamlog` — Display MyISAM Log File Contents

`myisamlog` processes the contents of a `MyISAM` log file. To create such a file, start the server with a `--log-isam=log_file` option.

Invoke `myisamlog` like this:

```
shell> myisamlog [options] [file_name [tbl_name] ...]
```

The default operation is update (`-u`). If a recovery is done (`-r`), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is `myisam.log` if no *`log_file`* argument is given. If tables are named on the command line, only those tables are updated.

`myisamlog` supports the following options:

- `-?`, `-I`

  Display a help message and exit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `-c N`

  Execute only *N* commands.

- `-f N`

  Specify the maximum number of open files.

- `-i`

  Display extra information before exiting.

- `-o offset`

  Specify the starting offset.

- `-p N`

  Remove *N* components from path.

- `-r`

  Perform a recovery operation.

- `-R record_pos_file record_pos`

  Specify record position file and record position.

- `-u`

  Perform an update operation.

- `-v`

  Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- `-w write_file`

  Specify the write file.

- `-V`

  Display version information.

4.6.5 myisampack — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses `MyISAM` tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40% to 70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If the mysqld server was invoked with external locking disabled, it is not a good idea to invoke myisampack if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read only. This is generally intended (such as when accessing packed tables on a CD).

Invoke myisampack like this:

```
shell> myisampack [options] file_name ...
```

Each file name argument should be the name of an index (.MYI) file. If you are not in the database directory, you should specify the path name to the file. It is permissible to omit the .MYI extension.

After you compress a table with myisampack, use myisamchk -rq to rebuild its indexes. Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

myisampack supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.7, “Command-Line Options that Affect Option-File Handling”.

- --help, -?

  Display a help message and exit.

- --backup, -b

  Make a backup of each table's data file using the name *tbl_name*.OLD.

- --character-sets-dir=*dir_name*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o.

- --force, -f

  Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of myisampack exists. (myisampack creates an intermediate file named *tbl_name*.TMD in the database directory while it compresses the table. If you kill myisampack, the .TMD file might not be deleted.) Normally, myisampack exits with an error if it finds that *tbl_name*.TMD exists. With --force, myisampack packs the table anyway.

- --join=*big_tbl_name*, -j *big_tbl_name*

  Join all tables named on the command line into a single packed table *big_tbl_name*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

  big_tbl_name must not exist prior to the join operation. All source tables named on the command line to be merged into *big_tbl_name* must exist. The source tables are read for the join operation but not modified. The join operation does not create a .frm file for *big_tbl_name*, so after the join operation finishes, copy the .frm file from one of the source tables and name it *big_tbl_name*.frm.

- --silent, -s

  Silent mode. Write output only when errors occur.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--test`, `-t`

  Do not actually pack the table, just test packing it.

- `--tmpdir=dir_name`, `-T dir_name`

  Use the named directory as the location where `myisampack` creates temporary files.

- `--verbose`, `-v`

  Verbose mode. Write information about the progress of the packing operation and its result.

- `--version`, `-V`

  Display version information and exit.

- `--wait`, `-w`

  Wait and retry if the table is in use. If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 monty    my         994128 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          53248 Apr 17 19:00 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile parts:            1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long        1024       1024          1
2   32    30  multip. text                10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
13    187   35
14    222   4
15    226   16
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
33    388   4
34    392   4
35    396   4
36    400   4
37    404   1
38    405   4
39    409   4
40    413   4
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:     20  empty-space:   16  empty-zero:     12  empty-fill:  11
pre-space:   0  end-space:     12  table-lookups:   5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> myisamchk -rq station
- check record delete-chain
- recovering (with sort) MyISAM-table 'station'
Data records: 1192
- Fixing index 1
- Fixing index 2

shell> mysqladmin -uroot flush-tables

shell> ls -l station.*
-rw-rw-r--   1 monty    my         127874 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          55296 Apr 17 19:04 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
Data records:               1192  Deleted blocks:              0
Datafile parts:             1192  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:    16777215  Max keyfile length:     131071
Recordlength:                834
Record format: Compressed

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long       10240       1024          1
2   32    30  multip. text                54272       1024          1

Field Start Length Type                         Huff tree  Bits
1     1     1      constant                             1     0
2     2     4      zerofill(1)                          2     9
3     6     4      no zeros, zerofill(1)                2     9
4     10    1                                           3     9
5     11    20     table-lookup                         4     0
6     31    1                                           3     9
7     32    30     no endspace, not_always              5     9
8     62    35     no endspace, not_always, no empty    6     9
9     97    35     no empty                             7     9
10    132   35     no endspace, not_always, no empty    6     9
11    167   4      zerofill(1)                          2     9
12    171   16     no endspace, not_always, no empty    5     9
13    187   35     no endspace, not_always, no empty    6     9
14    222   4      zerofill(1)                          2     9
15    226   16     no endspace, not_always, no empty    5     9
16    242   20     no endspace, not_always              8     9
17    262   20     no endspace, no empty                8     9
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table-lookup                         9     0
24    349   8      table-lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table-lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table-lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
54    806   1                                           3     9
55    807   20     no empty                             3     9
56    827   4      no zeros, zerofill(2)                2     9
57    831   4      no zeros, zerofill(1)                2     9
```

`myisampack` displays the following kinds of information:

- `normal`

  The number of columns for which no extra packing is used.

- `empty-space`

  The number of columns containing values that are only spaces. These occupy one bit.

- `empty-zero`

  The number of columns containing values that are only binary zeros. These occupy one bit.

- `empty-fill`

  The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.

- `pre-space`

  The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- `end-space`

  The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- `table-lookup`

  The column had only a small number of different values, which were converted to an `ENUM` before Huffman compression.

- `zero`

  The number of columns for which all values are zero.

- `Original trees`

  The initial number of Huffman trees.

- `After join`

  The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, the `Field` lines displayed by `myisamchk -dvv` include additional information about each column:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Type

  The data type. The value may contain any of the following descriptors:

  - constant

    All rows have the same value.

  - no endspace

    Do not store endspace.

  - no endspace, not_always

    Do not store endspace and do not do endspace compression for all values.

  - no endspace, no empty

    Do not store endspace. Do not store empty values.

  - table-lookup

    The column was converted to an ENUM.

  - zerofill(*N*)

    The most significant *N* bytes in the value are always 0 and are not stored.

  - no zeros

    Do not store zeros.

  - always zero

    Zero values are stored using one bit.

- Huff tree

  The number of the Huffman tree associated with the column.

- Bits

  The number of bits used in the Huffman tree.

After you run myisampack, use myisamchk to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute mysqladmin flush-tables to force mysqld to start using the new table.

To unpack a packed table, use the --unpack option to myisamchk.

4.6.6 mysqlaccess — Client for Checking Access Privileges

mysqlaccess is a diagnostic tool that Yves Carlier has provided for the MySQL distribution. It checks the access privileges for a host name, user name, and database combination. Note that mysqlaccess checks

This documentation is for an older version. If you're
This documentation is for an older version. If you're

access using only the user, db, and host tables. It does not check table, column, or routine privileges specified in the tables_priv, columns_priv, or procs_priv tables.

Invoke mysqlaccess like this:

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

mysqlaccess supports the following options.

Table 4.10 mysqlaccess Options

Format	Description
--brief	Generate reports in single-line tabular format
--commit	Copy the new access privileges from the temporary tables to the original grant tables
--copy	Reload the temporary grant tables from original ones
--db	Specify the database name
--debug	Specify the debug level
--help	Display help message and exit
--host	Connect to MySQL server on given host
--howto	Display some examples that show how to use mysqlaccess
--old_server	Assume that the server is an old MySQL server (prior to MySQL 3.21)
--password	Password to use when connecting to server
--plan	Display suggestions and ideas for future releases
--preview	Show the privilege differences after making changes to the temporary grant tables
--relnotes	Display release notes
--rhost	Connect to MySQL server on given host
--rollback	Undo the most recent changes to the temporary grant tables.
--spassword	Password to use when connecting to server as the superuser
--superuser	Specify the user name for connecting as the superuser
--table	Generate reports in table format
--user	MySQL user name to use when connecting to server
--version	Display version information and exit

- --help, -?

  Display a help message and exit.

- --brief, -b

  Generate reports in single-line tabular format.

- --commit

  Copy the new access privileges from the temporary tables to the original grant tables. The grant tables must be flushed for the new privileges to take effect. (For example, execute a mysqladmin reload command.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--copy`

  Reload the temporary grant tables from original ones.

- `--db=db_name`, `-d db_name`

  Specify the database name.

- `--debug=N`

  Specify the debug level. *`N`* can be an integer from 0 to 3.

- `--host=host_name`, `-h host_name`

  The host name to use in the access privileges.

- `--howto`

  Display some examples that show how to use `mysqlaccess`.

- `--old_server`

  Assume that the server is an old MySQL server (before MySQL 3.21) that does not yet know how to handle full `WHERE` clauses.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqlaccess` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--plan`

  Display suggestions and ideas for future releases.

- `--preview`

  Show the privilege differences after making changes to the temporary grant tables.

- `--relnotes`

  Display the release notes.

- `--rhost=host_name`, `-H host_name`

  Connect to the MySQL server on the given host.

- `--rollback`

  Undo the most recent changes to the temporary grant tables.

- `--spassword[=password]`, `-P[password]`

  The password to use when connecting to the server as the superuser. If you omit the *`password`* value following the `--spassword` or `-p` option on the command line, `mysqlaccess` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--superuser=user_name`, `-U user_name`

  Specify the user name for connecting as the superuser.

- `--table`, `-t`

  Generate reports in table format.

- `--user=user_name`, `-u user_name`

  The user name to use in the access privileges.

- `--version`, `-v`

  Display version information and exit.

If your MySQL distribution is installed in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

4.6.7 `mysqlbinlog` — Utility for Processing Binary Log Files

The server's binary log consists of files containing “events” that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the `mysqlbinlog` utility. You can also use `mysqlbinlog` to display the contents of relay log files written by a slave server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in Section 5.4.3, “The Binary Log”, and Section 16.2.2, “Replication Relay and Status Logs”.

Invoke `mysqlbinlog` like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.0000003
```

The output includes events contained in `binlog.000003`. Event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309  9:28:36 server id 123  end_log_pos 245
  Query thread_id=3350  exec_time=11  error_code=0
```

In the first line, the number following `at` indicates the file offset, or starting position, of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to slave servers. `server id` is the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

server_id value of the server where the event originated. end_log_pos indicates where the next event starts (that is, it is the end position of the current event + 1). thread_id indicates which thread executed the event. exec_time is the time spent executing the event, on a master server. On a slave, it is the difference of the end execution time on the slave minus the beginning execution time on the master. The difference serves as an indicator of how much replication lags behind the master. error_code indicates the result from executing the event. Zero means that no error occurred.

Note

When using event groups, the file offsets of events may be grouped together and the comments of events may be grouped together. Do not mistake these grouped events for blank file offsets.

The output from mysqlbinlog can be re-executed (for example, by using it as input to mysql) to redo the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section and in Section 7.5, "Point-in-Time (Incremental) Recovery Using the Binary Log".

Normally, you use mysqlbinlog to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the --read-from-remote-server option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are --host, --password, --port, --protocol, --socket, and --user; they are ignored except when you also use the --read-from-remote-server option.

When running mysqlbinlog against a large binary log, be careful that the filesystem has enough space for the resulting files. To configure the directory that mysqlbinlog uses for temporary files, use the TMPDIR environment variable.

mysqlbinlog supports the following options, which can be specified on the command line or in the [mysqlbinlog] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, "Using Option Files".

Table 4.11 mysqlbinlog Options

Format	Description	Introduced
--character-sets-dir	Directory where character sets are installed	
--database	List entries for just this database	
--debug	Write debugging log	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	5.0.10
--disable-log-bin	Disable binary logging	
--force-read	If mysqlbinlog reads a binary log event that it does not recognize, it prints a warning	
--help	Display help message and exit	
--hexdump	Display a hex dump of the log in comments	5.0.16
--host	Connect to MySQL server on given host	
--local-load	Prepare local temporary files for LOAD DATA INFILE in the specified directory	
--no-defaults	Read no option files	
--offset	Skip the first N entries in the log	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Introduced
--password	Password to use when connecting to server	
--port	TCP/IP port number to use for connection	
--position	Deprecated. Use --start-position	
--print-defaults	Print default options	
--protocol	Connection protocol to use	
--read-from-remote-server	Read binary log from MySQL server rather than local log file	
--result-file	Direct output to named file	
--set-charset	Add a SET NAMES charset_name statement to the output	5.0.23
--short-form	Display only the statements contained in the log	
--socket	For connections to localhost, the Unix socket file to use	
--start-datetime	Read binary log from first event with timestamp equal to or later than datetime argument	
--start-position	Read binary log from first event with position equal to or greater than argument	
--stop-datetime	Stop reading binary log at first event with timestamp equal to or greater than datetime argument	
--stop-position	Stop reading binary log at first event with position equal to or greater than argument	
--to-last-log	Do not stop at the end of requested binary log from a MySQL server, but rather continue printing to end of last binary log	
--user	MySQL user name to use when connecting to server	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--database=`*`db_name`*, `-d` *`db_name`*

  This option causes `mysqlbinlog` to output entries from the binary log (local log only) that occur while *`db_name`* is been selected as the default database by `USE`.

  The `--database` option for `mysqlbinlog` is similar to the `--binlog-do-db` option for `mysqld`, but can be used to specify only one database. If `--database` is given multiple times, only the last instance is used.

  The `--database` option works as follows:

  - While *`db_name`* is the default database, statements are output whether they modify tables in *`db_name`* or a different database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Unless *db_name* is selected as the default database, statements are not output, even if they modify tables in *db_name*.
- There is an exception for CREATE DATABASE, ALTER DATABASE, and DROP DATABASE. The database being *created, altered, or dropped* is considered to be the default database when determining whether to output the statement.

Suppose that the binary log contains these statements:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j)  VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i)      VALUES(102);
INSERT INTO db2.t2 (j)  VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j)  VALUES(202);
INSERT INTO t2 (j)      VALUES(203);
```

mysqlbinlog --database=test does not output the first two INSERT statements because there is no default database. It outputs the three INSERT statements following USE test, but not the three INSERT statements following USE db2.

mysqlbinlog --database=db2 does not output the first two INSERT statements because there is no default database. It does not output the three INSERT statements following USE test, but does output the three INSERT statements following USE db2.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o,/tmp/mysqlbinlog.trace.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is the full path name to the file.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlbinlog normally reads the [client] and [mysqlbinlog] groups. If the --defaults-group-suffix=_other option is given, mysqlbinlog also reads the [client_other] and [mysqlbinlog_other] groups. This option was added in MySQL 5.0.10.

- --disable-log-bin, -D

  Disable binary logging. This is useful for avoiding an endless loop if you use the --to-last-log option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This option requires that you have the SUPER privilege. It causes mysqlbinlog to include a SET sql_log_bin = 0 statement in its output to disable binary logging of the remaining output. The SET statement is ineffective unless you have the SUPER privilege.

- --force-read, -f

  With this option, if mysqlbinlog reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, mysqlbinlog stops if it reads such an event.

- --hexdump, -H

  Display a hex dump of the log in comments. The hex output can be helpful for replication debugging. Hex dump format is discussed later in this section. This option was added in MySQL 5.0.16.

- --host=*host_name*, -h *host_name*

  Get the binary log from the MySQL server on the given host.

- --local-load=*dir_name*, -l *dir_name*

  Prepare local temporary files for LOAD DATA INFILE in the specified directory.

  Important

  These temporary files are not automatically removed by mysqlbinlog or any other MySQL program.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

- --offset=*N*, -o *N*

  Skip the first *N* entries in the log.

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysqlbinlog prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for connecting to a remote server.

- --position=*N*

  Deprecated. Use --start-position instead. --position is removed in MySQL 5.5.

- --print-defaults

  Print the program name and all options that it gets from option files.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--read-from-remote-server`, `-R`

  Read the binary log from a MySQL server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`.

  This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.

- `--result-file=`*`name`*, `-r` *`name`*

  Direct output to the given file.

- `--set-charset=`*`charset_name`*

  Add a `SET NAMES` *`charset_name`* statement to the output to specify the character set to be used for processing log files. This option was added in MySQL 5.0.23.

- `--short-form`, `-s`

  Display only the statements contained in the log, without any extra information. This is for testing only, and should not be used in production systems.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--start-datetime=`*`datetime`*

  Start reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. The *`datetime`* value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

  ```
  shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
  ```

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--start-position=`*`N`*, `-j` *`N`*

  Start reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the first log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-datetime=`*`datetime`*

  Stop reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` option for information about the *`datetime`* value.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-position=N`

  Stop reading the binary log at the first event having a position equal to or greater than *N*. This option applies to the last log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--to-last-log`, `-t`

  Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to a remote server.

- `--version`, `-V`

  Display version information and exit.

  In MySQL 5.0, the version number shown for `mysqlbinlog` is always 3.2.

You can also set the following variable by using `--var_name=value` syntax:

- `open_files_limit`

  Specify the number of open file descriptors to reserve.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated*.

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the events contained in the binary log. This technique is used to recover from a crash when you have an old backup (see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”). For example:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the `mysql` program:

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
shell> mysql -u root -p < tmpfile
```

When `mysqlbinlog` is invoked with the `--start-position` option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start

This documentation is for an older version. If you're
This documentation is for an older version. If you're

of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the `--stop-datetime` option (to be able to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* `mysql` process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

`mysqlbinlog` can produce output that reproduces a `LOAD DATA INFILE` operation without the original data file. `mysqlbinlog` copies the data to a temporary file and writes a `LOAD DATA LOCAL INFILE` statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the `--local-load` option.

Because `mysqlbinlog` converts `LOAD DATA INFILE` statements to `LOAD DATA LOCAL INFILE` statements (that is, it adds `LOCAL`), both the client and the server that you use to process the statements must be configured with the `LOCAL` capability enabled. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

Warning

The temporary files created for `LOAD DATA LOCAL` statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like *`original_file_name-#-#`*.

The `--hexdump` option produces a hex dump of the log contents:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

The hex output consists of comment lines beginning with `#`, so the output might look like this for the preceding command:

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp   Type   Master ID        Size      Master Pos    Flags
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
# 00000004 9d fc 5c 43   0f   01 00 00 00   5e 00 00 00   62 00 00 00   00 00
# 00000017 04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og..............|
# 00000037 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
# 00000047 00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.......C.8......|
# 00000057 04 04 04 04 12 00 00 4b  00 04 1a               |.......K...|
#       Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
#       at startup
ROLLBACK;
```

Hex dump output currently contains the following elements. This format is subject to change.

- `Position`: The byte position within the log file.
- `Timestamp`: The event timestamp. In the example shown, `'9d fc 5c 43'` is the representation of `'051024 17:24:13'` in hexadecimal.
- `Type`: The event type code. In the example shown, `'0f'` indicates a `FORMAT_DESCRIPTION_EVENT`. The following table lists the possible type codes.

Type	Name	Meaning
00	UNKNOWN_EVENT	This event should never be present in the log.
01	START_EVENT_V3	This indicates the start of a log file written by MySQL 4 or earlier.
02	QUERY_EVENT	The most common type of events. These contain statements executed on the master.
03	STOP_EVENT	Indicates that master has stopped.
04	ROTATE_EVENT	Written when the master switches to a new log file.
05	INTVAR_EVENT	Used for AUTO_INCREMENT values or when the LAST_INSERT_ID() function is used in the statement.
06	LOAD_EVENT	Used for LOAD DATA INFILE in MySQL 3.23.
07	SLAVE_EVENT	Reserved for future use.
08	CREATE_FILE_EVENT	Used for LOAD DATA INFILE statements. This indicates the start of execution of such a statement. A temporary file is created on the slave. Used in MySQL 4 only.
09	APPEND_BLOCK_EVENT	Contains data for use in a LOAD DATA INFILE statement. The data is stored in the temporary file on the slave.
0a	EXEC_LOAD_EVENT	Used for LOAD DATA INFILE statements. The contents of the temporary file is stored in the table on the slave. Used in MySQL 4 only.
0b	DELETE_FILE_EVENT	Rollback of a LOAD DATA INFILE statement. The temporary file should be deleted on the slave.
0c	NEW_LOAD_EVENT	Used for LOAD DATA INFILE in MySQL 4 and earlier.
0d	RAND_EVENT	Used to send information about random values if the RAND() function is used in the statement.
0e	USER_VAR_EVENT	Used to replicate user variables.
0f	FORMAT_DESCRIPTION_EVENT	This indicates the start of a log file written by MySQL 5 or later.
10	XID_EVENT	Event indicating commit of an XA transaction.
11	BEGIN_LOAD_QUERY_EVENT	Used for LOAD DATA INFILE statements in MySQL 5 and later.
12	EXECUTE_LOAD_QUERY_EVENT	Used for LOAD DATA INFILE statements in MySQL 5 and later.
13	TABLE_MAP_EVENT	Reserved for future use.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Type	Name	Meaning
14	WRITE_ROWS_EVENT	Reserved for future use.
15	UPDATE_ROWS_EVENT	Reserved for future use.
16	DELETE_ROWS_EVENT	Reserved for future use.

- Master ID: The server ID of the master that created the event.
- Size: The size in bytes of the event.
- Master Pos: The position of the next event in the original master log file.
- Flags: 16 flags. The following flags are used. The others are reserved for future use.

Flag	Name	Meaning
01	LOG_EVENT_BINLOG_IN_USE_F	Log file correctly closed. (Used only in FORMAT_DESCRIPTION_EVENT.) If this flag is set (if the flags are, for example, '01 00') in a FORMAT_DESCRIPTION_EVENT, the log file has not been properly closed. Most probably this is because of a master crash (for example, due to power failure).
02		Reserved for future use.
04	LOG_EVENT_THREAD_SPECIFIC_F	Set if the event is dependent on the connection it was executed in (for example, '04 00'), for example, if the event uses temporary tables.
08	LOG_EVENT_SUPPRESS_USE_F	Set in some circumstances when the event is not dependent on the default database.

4.6.8 mysqldumpslow — Summarize Slow Query Log Files

The MySQL slow query log contains information about queries that take a long time to execute (see Section 5.4.4, “The Slow Query Log”). mysqldumpslow parses MySQL slow query log files and prints a summary of their contents.

Normally, mysqldumpslow groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to N and 'S' when displaying summary output. The -a and -n options can be used to modify value abstracting behavior.

Invoke mysqldumpslow like this:

```
shell> mysqldumpslow [options] [log_file ...]
```

mysqldumpslow supports the following options.

Table 4.12 mysqldumpslow Options

Format	Description
-a	Do not abstract all numbers to N and strings to S
-n	Abstract numbers with at least the specified digits
--debug	Write debugging information
-g	Only consider statements that match the pattern
--help	Display help message and exit
-h	Host name of the server in the log file name

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description
-i	Name of the server instance
-l	Do not subtract lock time from total time
-r	Reverse the sort order
-s	How to sort output
-t	Display only first num queries
--verbose	Verbose mode

- `--help`

  Display a help message and exit.

- `-a`

  Do not abstract all numbers to `N` and strings to `'S'`.

- `--debug`, `-d`

  Run in debug mode.

- `-g` *`pattern`*

  Consider only queries that match the (`grep`-style) pattern.

- `-h` *`host_name`*

  Host name of MySQL server for `*-slow.log` file name. The value can contain a wildcard. The default is `*` (match all).

- `-i` *`name`*

  Name of server instance (if using `mysql.server` startup script).

- `-l`

  Do not subtract lock time from total time.

- `-n` *`N`*

  Abstract numbers with at least *`N`* digits within names.

- `-r`

  Reverse the sort order.

- `-s` *`sort_type`*

  How to sort the output. The value of *`sort_type`* should be chosen from the following list:

  - `t`, `at`: Sort by query time or average query time
  - `l`, `al`: Sort by lock time or average lock time
  - `r`, `ar`: Sort by rows sent or average rows sent
  - `c`: Sort by count

This documentation is for an older version. If you're
This documentation is for an older version. If you're

By default, mysqldumpslow sorts by average query time (equivalent to -s at).

- -t N

  Display only the first N queries in the output.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

Example of usage:

```
shell> mysqldumpslow

Reading mysql slow query log from /usr/local/mysql/data/mysqld51-apple-slow.log
Count: 1  Time=4.32s (4s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1

Count: 3  Time=2.53s (7s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1 limit N

Count: 3  Time=2.13s (6s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t1 select * from t1
```

4.6.9 mysqlhotcopy — A Database Backup Program

mysqlhotcopy is a Perl script that was originally written and contributed by Tim Bunce. It uses FLUSH TABLES, LOCK TABLES, and cp or scp to make a database backup. It is a fast way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. mysqlhotcopy works only for backing up MyISAM and ARCHIVE tables. It runs on Unix and NetWare.

To use mysqlhotcopy, you must have read access to the files for the tables that you are backing up, the SELECT privilege for those tables, the RELOAD privilege (to be able to execute FLUSH TABLES), and the LOCK TABLES privilege (to be able to lock the tables).

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (“~”):

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopy supports the following options, which can be specified on the command line or in the [mysqlhotcopy] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Table 4.13 mysqlhotcopy Options

Format	Description
--addtodest	Do not rename target directory (if it exists); merely add files to it

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description
--allowold	Do not abort if a target exists; rename it by adding an _old suffix
--checkpoint	Insert checkpoint entries
--chroot	Base directory of the chroot jail in which mysqld operates
--debug	Write debugging log
--dryrun	Report actions without performing them
--flushlog	Flush logs after all tables are locked
--help	Display help message and exit
--host	Connect to MySQL server on given host
--keepold	Do not delete previous (renamed) target when done
--method	The method for copying files
--noindices	Do not include full index files in the backup
--password	Password to use when connecting to server
--port	TCP/IP port number to use for connection
--quiet	Be silent except for errors
--regexp	Copy all databases with names that match the given regular expression
--resetmaster	Reset the binary log after locking all the tables
--resetslave	Reset the master.info file after locking all the tables
--socket	For connections to localhost, the Unix socket file to use
--tmpdir	The temporary directory
--user	MySQL user name to use when connecting to server

- `--help`, `-?`

  Display a help message and exit.

- `--addtodest`

  Do not rename target directory (if it exists); merely add files to it.

- `--allowold`

  Do not abort if a target exists; rename it by adding an `_old` suffix.

- `--checkpoint=`*`db_name.tbl_name`*

  Insert checkpoint entries into the specified database *`db_name`* and table *`tbl_name`*.

- `--chroot=`*`dir_name`*

  Base directory of the `chroot` jail in which `mysqld` operates. The *`dir_name`* value should match that of the `--chroot` option given to `mysqld`.

- `--debug`

  Enable debug output.

- `--dryrun`, `-n`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Report actions without performing them.

- `--flushlog`

  Flush logs after all tables are locked.

- `--host=host_name`, `-h host_name`

  The host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to `localhost` using a Unix socket file.

- `--keepold`

  Do not delete previous (renamed) target when done.

- `--method=command`

  The method for copying files (`cp` or `scp`). The default is `cp`.

- `--noindices`

  Do not include full index files for `MyISAM` tables in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with `myisamchk -rq`.

- `--password=password`, `-ppassword`

  The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use when connecting to the local server.

- `--quiet`, `-q`

  Be silent except for errors.

- `--record_log_pos=db_name.tbl_name`

  Record master and slave status in the specified database *`db_name`* and table *`tbl_name`*.

- `--regexp=expr`

  Copy all databases with names that match the given regular expression.

- `--resetmaster`

  Reset the binary log after locking all the tables.

- `--resetslave`

  Reset the `master.info` file after locking all the tables.

- `--socket=path`, `-S path`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The Unix socket file to use for connections to `localhost`.

- `--suffix=str`

  The suffix to use for names of copied databases.

- `--tmpdir=dir_name`

  The temporary directory. The default is `/tmp`.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

Use `perldoc` for additional `mysqlhotcopy` documentation, including information about the structure of the tables needed for the `--checkpoint` and `--record_log_pos` options:

```
shell> perldoc mysqlhotcopy
```

4.6.10 `mysqlmanager` — The MySQL Instance Manager

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

`mysqlmanager` is the MySQL Instance Manager (IM). This program monitors and manages MySQL Database Server instances. MySQL Instance Manager is available for Unix-like operating systems, and also on Windows as of MySQL 5.0.13. It runs as a daemon that listens on a TCP/IP port. On Unix, it also listens on a Unix socket file.

MySQL Instance Manager is included in MySQL distributions from version 5.0.3, and can be used in place of the `mysqld_safe` script to start and stop one or more instances of MySQL Server. Because Instance Manager can manage multiple server instances, it can also be used in place of the `mysqld_multi` script. Instance Manager offers these capabilities:

- Instance Manager can start and stop instances, and report on the status of instances.
- Server instances can be treated as guarded or unguarded:
  - When Instance Manager starts, it starts each guarded instance. If the instance crashes, Instance Manager detects this and restarts it. When Instance Manager stops, it stops the instance.
  - A nonguarded instance is not started when Instance Manager starts or monitored by it. If the instance crashes after being started, Instance Manager does not restart it. When Instance Manager exits, it does not stop the instance if it is running.

  Instances are guarded by default. An instance can be designated as nonguarded by including the `nonguarded` option in the configuration file.
- Instance Manager provides an interactive interface for configuring instances, so that the need to edit the configuration file manually is reduced or eliminated.
- Instance Manager provides remote instance management. That is, it runs on the host where you want to control MySQL Server instances, but you can connect to it from a remote host to perform instance-management operations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following sections describe MySQL Instance Manager operation in more detail.

4.6.10.1 MySQL Instance Manager Command Options

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

The MySQL Instance Manager supports a number of command options. For a brief listing, invoke `mysqlmanager` with the `--help` option. Options may be given on the command line or in the Instance Manager configuration file. On Windows, the standard configuration file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard file is `/etc/my.cnf`. To specify a different configuration file, start Instance Manager with the `--defaults-file` option.

`mysqlmanager` supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.7, "Command-Line Options that Affect Option-File Handling".

- `--help`, `-?`

  Display a help message and exit.

- `--angel-pid-file=`*`file_name`*

  The file in which the angel process records its process ID when `mysqlmanager` runs in daemon mode (that is, when the `--run-as-service` option is given). The default file name is `mysqlmanager.angel.pid`.

  If the `--angel-pid-file` option is not given, the default angel PID file has the same name as the PID file except that any PID file extension is replaced with an extension of `.angel.pid`. (For example, `mysqlmanager.pid` becomes `mysqlmanager.angel.pid`.)

  This option was added in MySQL 5.0.23.

- `--bind-address=`*`IP`*

  The IP address to bind to.

- `--default-mysqld-path=`*`file_name`*

  The path name of the MySQL Server binary. This path name is used for all server instance sections in the configuration file for which no `mysqld-path` option is present. The default value of this option is the compiled-in path name, which depends on how the MySQL distribution was configured. Example: `--default-mysqld-path=/usr/sbin/mysqld`

- `--defaults-file=`*`file_name`*

  Read Instance Manager and MySQL Server settings from the given file. All configuration changes made by the Instance Manager will be written to this file. This must be the first option on the command line if it is used, and the file must exist.

  If this option is not given, Instance Manager uses its standard configuration file. On Windows, the standard file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard file is `/etc/my.cnf`.

- `--install`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

On Windows, install Instance Manager as a Windows service. The service name is `MySQL Manager`. This option was added in MySQL 5.0.11.

- `--log=file_name`

  The path to the Instance Manager log file. This option has no effect unless the `--run-as-service` option is also given. If the file name specified for the option is a relative name, the log file is created under the directory from which Instance Manager is started. To ensure that the file is created in a specific directory, specify it as a full path name.

  If `--run-as-service` is given without `--log`, the log file is `mysqlmanager.log` in the data directory.

  If `--run-as-service` is not given, log messages go to the standard output. To capture log output, you can redirect Instance Manager output to a file:

  ```
  mysqlmanager > im.log
  ```

- `--monitoring-interval=seconds`

  The interval in seconds for monitoring server instances. The default value is 20 seconds. Instance Manager tries to connect to each monitored (guarded) instance using the nonexisting `MySQL_Instance_Manager` user account to check whether it is available/not hanging. If the result of the connection attempt indicates that the instance is unavailable, Instance Manager performs several attempts to restart the instance.

  Normally, the `MySQL_Instance_Manager` account does not exist, so the connection attempts by Instance Manager cause the monitored instance to produce messages in its general query log similar to the following:

  ```
  Access denied for user 'MySQL_Instance_M'@'localhost' (using password: YES)
  ```

  The `nonguarded` option in the appropriate server instance section disables monitoring for a particular instance. If the instance dies after being started, Instance Manager will not restart it. Instance Manager tries to connect to a nonguarded instance only when you request the instance's status (for example, with the `SHOW INSTANCES` status.

  See Section 4.6.10.5, “MySQL Server Instance Status Monitoring”, for more information.

- `--passwd`, `-P`

  Prepare an entry for the password file, print it to the standard output, and exit. You can redirect the output from Instance Manager to a file to save the entry in the file. See also Section 4.6.10.4, “Instance Manager User and Password Management”. This

- `--password-file=file_name`

  The name of the file where the Instance Manager looks for users and passwords. On Windows, the default is `mysqlmanager.passwd` in the directory where Instance Manager is installed. On Unix, the default file is `/etc/mysqlmanager.passwd`. See also Section 4.6.10.4, “Instance Manager User and Password Management”.

- `--pid-file=file_name`

  The process ID file to use. On Windows, the default file is `mysqlmanager.pid` in the directory where Instance Manager is installed. On Unix, the default is `mysqlmanager.pid` in the data directory.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--port=port_num`

  The port number to use when listening for TCP/IP connections from clients. The default port number (assigned by IANA) is 2273.

- `--print-defaults`

  Print the current defaults and exit. This must be the first option on the command line if it is used.

- `--remove`

  On Windows, removes Instance Manager as a Windows service. This assumes that Instance Manager has been run with `--install` previously. This option was added in MySQL 5.0.11.

- `--run-as-service`

  On Unix, daemonize and start an angel process. The angel process monitors Instance Manager and restarts it if it crashes. (The angel process itself is simple and unlikely to crash.)

- `--socket=path`

  On Unix, the socket file to use for incoming connections. The default file is named `/tmp/mysqlmanager.sock`. This option has no meaning on Windows.

- `--standalone`

  This option is used on Windows to run Instance Manager in standalone mode. You should specify it when you start Instance Manager from the command line. This option was added in MySQL 5.0.13.

- `--user=user_name`

  On Unix, the user name of the system account to use for starting and running `mysqlmanager`. This option generates a warning and has no effect unless you start `mysqlmanager` as `root` (so that it can change its effective user ID), or as the named user. It is recommended that you configure `mysqlmanager` to run using the same account used to run the `mysqld` server. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

- `--version`, `-V`

  Display version information and exit.

- `--wait-timeout=N`

  The number of seconds to wait for activity on an incoming connection before closing it. The default is 28800 seconds (8 hours).

  This option was added in MySQL 5.0.19. Before that, the timeout is 30 seconds and cannot be changed.

4.6.10.2 MySQL Instance Manager Configuration Files

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

Instance Manager uses its standard configuration file unless it is started with a `--defaults-file` option that specifies a different file. On Windows, the standard file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard file is `/etc/my.cnf`. (Prior to MySQL 5.0.10, the MySQL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Instance Manager read the same configuration files as the MySQL Server, including `/etc/my.cnf`, `~/.my.cnf`, and so forth.)

Instance Manager reads options for itself from the `[manager]` section of the configuration file, and options for server instances from `[mysqld]` or `[mysqldN]` sections. The `[manager]` section contains any of the options listed in Section 4.6.10.1, "MySQL Instance Manager Command Options", except for those specified as having to be given as the first option on the command line. Here is a sample `[manager]` section:

```
# MySQL Instance Manager options section
[manager]
default-mysqld-path = /usr/local/mysql/libexec/mysqld
socket=/tmp/manager.sock
pid-file=/tmp/manager.pid
password-file = /home/cps/.mysqlmanager.passwd
monitoring-interval = 2
port = 1999
bind-address = 192.168.1.5
```

Each `[mysqld]` or `[mysqldN]` instance section specifies options given by Instance Manager to a server instance at startup. These are mainly common MySQL Server options (see Section 5.1.3, "Server Command Options"). In addition, a `[mysqldN]` section can contain the options in the following list, which are specific to Instance Manager. These options are interpreted by Instance Manager itself; it does not pass them to the server when it attempts to start that server.

Warning

The Instance Manager-specific options must not be used in a `[mysqld]` section. If a server is started without using Instance Manager, it will not recognize these options and will fail to start properly.

- `mysqld-path = file_name`

  The path name of the `mysqld` server binary to use for the server instance.

- `nonguarded`

  This option disables Instance Manager monitoring functionality for the server instance. By default, an instance is guarded: At Instance Manager start time, it starts the instance. It also monitors the instance status and attempts to restart it if it fails. At Instance Manager exit time, it stops the instance. None of these things happen for nonguarded instances.

- `shutdown-delay = seconds`

  The number of seconds Instance Manager should wait for the server instance to shut down. The default value is 35 seconds. After the delay expires, Instance Manager assumes that the instance is hanging and attempts to terminate it. If you use `InnoDB` with large tables, you should increase this value.

Here are some sample instance sections:

```
[mysqld1]
mysqld-path=/usr/local/mysql/libexec/mysqld
socket=/tmp/mysql.sock
port=3307
server_id=1
skip-stack-trace
core-file
skip-bdb
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
log-bin
log-error
log=mylog
log-slow-queries

[mysqld2]
nonguarded
port=3308
server_id=2
mysqld-path= /home/cps/mysql/trees/mysql-5.0/sql/mysqld
socket     = /tmp/mysql.sock5
pid-file   = /tmp/hostname.pid5
datadir= /home/cps/mysql_data/data_dir1
language=/home/cps/mysql/trees/mysql-5.0/sql/share/english
log-bin
log=/tmp/fordel.log
```

4.6.10.3 Starting the MySQL Server with MySQL Instance Manager

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

This section discusses how Instance Manager starts server instances when it starts. However, before you start Instance Manager, you should set up a password file for it. Otherwise, you will not be able to connect to Instance Manager to control it after it starts. For details about creating Instance Manager accounts, see Section 4.6.10.4, “Instance Manager User and Password Management”.

On Unix, the `mysqld` MySQL database server normally is started with the `mysql.server` script, which usually resides in the `/etc/init.d/` directory. In MySQL 5.0.3, this script invokes `mysqlmanager` (the MySQL Instance Manager binary) to start MySQL. (In prior versions of MySQL the `mysqld_safe` script is used for this purpose.) Starting from MySQL 5.0.4, the behavior of the startup script was changed again to incorporate both setup schemes. In version 5.0.4, the startup script uses the old scheme (invoking `mysqld_safe`) by default, but one can set the `use_mysqld_safe` variable in the script to `0` (zero) to use the MySQL Instance Manager to start a server.

Starting with MySQL 5.0.19, you can use Instance Manager if you modify the `my.cnf` configuration file by adding `use-manager` to the `[mysql.server]` section:

```
[mysql.server]
use-manager
```

When Instance Manager starts, it reads its configuration file if it exists to find server instance sections and prepare a list of instances. Instance sections have names of the form `[mysqld]` or `[mysqldN]`, where *`N`* is an unsigned integer (for example, `[mysqld1]`, `[mysqld2]`, and so forth).

After preparing the list of instances, Instance Manager starts the guarded instances in the list. If there are no instances, Instance Manager creates an instance named `mysqld` and attempts to start it with default (compiled-in) configuration values. This means that the Instance Manager cannot find the `mysqld` program if it is not installed in the default location. (Section 2.7, “Installation Layouts”, describes default locations for components of MySQL distributions.) If you have installed the MySQL server in a nonstandard location, you should create the Instance Manager configuration file.

Instance Manager also stops all guarded server instances when it shuts down.

The permissible options for `[mysqldN]` server instance sections are described in Section 4.6.10.2, “MySQL Instance Manager Configuration Files”. In these sections, you can use a special `mysqld-path=`*`path-to-mysqld-binary`* option that is recognized only by Instance Manager. Use this option

This documentation is for an older version. If you're
This documentation is for an older version. If you're

to let Instance Manager know where the mysqld binary resides. If there are multiple instances, it may also be necessary to set other options such as datadir and port, to ensure that each instance has a different data directory and TCP/IP port number. Section 5.5, “Running Multiple MySQL Instances on One Machine”, discusses the configuration values that must differ for each instance when you run multiple instance on the same machine.

Warning

The [mysqld] instance section, if it exists, must not contain any Instance Manager-specific options.

The typical Unix startup/shutdown cycle for a MySQL server with the MySQL Instance Manager enabled is as follows:

1. The /etc/init.d/mysql script starts MySQL Instance Manager.
2. Instance Manager starts the guarded server instances and monitors them.
3. If a server instance fails, Instance Manager restarts it.
4. If Instance Manager is shut down (for example, with the /etc/init.d/mysql stop command), it shuts down all server instances.

4.6.10.4 Instance Manager User and Password Management

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

The Instance Manager stores its user information in a password file. On Windows, the default is mysqlmanager.passwd in the directory where Instance Manager is installed. On Unix, the default file is /etc/mysqlmanager.passwd. To specify a different location for the password file, use the --password-file option.

If the password file does not exist or contains no password entries, you cannot connect to the Instance Manager.

Note

Any Instance Manager process that is running to monitor server instances does not notice changes to the password file. You must stop it and restart it after making password entry changes.

Entries in the password file have the following format, where the two fields are the account user name and encrypted password, separated by a colon:

```
petr:*35110DC9B4D8140F5DE667E28C72DD2597B5C848
```

Instance Manager password encryption is the same as that used by MySQL Server. It is a one-way operation; no means are provided for decrypting encrypted passwords.

Instance Manager accounts differ somewhat from MySQL Server accounts:

- MySQL Server accounts are associated with a host name, user name, and password (see Section 6.3.1, “User Names and Passwords”).
- Instance Manager accounts are associated with a user name and password only.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This means that a client can connect to Instance Manager with a given user name from any host. To limit connections so that clients can connect only from the local host, start Instance Manager with the `--bind-address=127.0.0.1` option so that it listens only to the local network interface. Remote clients will not be able to connect. Local clients can connect like this:

```
shell> mysql -h 127.0.0.1 -P 2273
```

To generate a new entry, invoke Instance Manager with the `--passwd` option and append the output to the `/etc/mysqlmanager.passwd` file. Here is an example:

```
shell> mysqlmanager --passwd >> /etc/mysqlmanager.passwd
Creating record for new user.
Enter user name: mike
Enter password: mikepass
Re-type password: mikepass
```

At the prompts, enter the user name and password for the new Instance Manager user. You must enter the password twice. It does not echo to the screen, so double entry guards against entering a different password than you intend (if the two passwords do not match, no entry is generated).

The preceding command causes the following line to be added to `/etc/mysqlmanager.passwd`:

```
mike:*BBF1F551DD9DD96A01E66EC7DDC073911BAD17BA
```

Use of the `--passwd` option fails if `mysqlmanager` is invoked directly from an IBM 5250 terminal. To work around this, use a command like the following from the command line to generate the password entry:

```
shell> mysql -B --skip-column-name \
        -e 'SELECT CONCAT("user_name",":",PASSWORD("pass_val"));'
```

The output from the command can be used an entry in the `/etc/mysqlmanager.passwd` file.

4.6.10.5 MySQL Server Instance Status Monitoring

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

To monitor the status of each guarded server instance, the MySQL Instance Manager attempts to connect to the instance at regular intervals using the `MySQL_Instance_Manager@localhost` user account with a password of `check_connection`.

You are *not* required to create this account for MySQL Server; in fact, it is expected that it will not exist. Instance Manager can tell that a server is operational if the server accepts the connection attempt but refuses access for the account by returning a login error. However, these failed connection attempts are logged by the server to its general query log (see Section 5.4.2, “The General Query Log”).

Instance Manager also attempts a connection to nonguarded server instances when you use the `SHOW INSTANCES` or `SHOW INSTANCE STATUS` command. This is the only status monitoring done for nonguarded instances.

Instance Manager knows if a server instance fails at startup because it receives a status from the attempt. For an instance that starts but later crashes, Instance Manager receives a signal because it is the parent process of the instance.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4.6.10.6 Connecting to MySQL Instance Manager

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

After you set up a password file for the MySQL Instance Manager and Instance Manager is running, you can connect to it. The MySQL client/server protocol is used to communicate with the Instance Manager. For example, you can connect to it using the standard `mysql` client program:

```
shell> mysql --port=2273 --host=im.example.org --user=mysql --password
```

Instance Manager supports the version of the MySQL client/server protocol used by the client tools and libraries distributed with MySQL 4.1 or later, so other programs that use the MySQL C API also can connect to it.

4.6.10.7 MySQL Instance Manager Commands

Important

MySQL Instance Manager is been deprecated in MySQL 5.1 and is removed in MySQL 5.5.

After you connect to MySQL Instance Manager, you can issue commands. The following general principles apply to Instance Manager command execution:

- Commands that take an instance name fail if the name is not a valid instance name.
- Commands that take an instance name fail if the instance does not exist.
- Instance Manager maintains information about instance configuration in an internal (in-memory) cache. Initially, this information comes from the configuration file if it exists, but some commands change the configuration of an instance. Commands that modify the configuration file fail if the file does not exist or is not accessible to Instance Manager.
- On Windows, the standard file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard configuration file is `/etc/my.cnf`. To specify a different configuration file, start Instance Manager with the `--defaults-file` option.
- If a `[mysqld]` instance section exists in the configuration file, it must not contain any Instance Manager-specific options (see Section 4.6.10.2, “MySQL Instance Manager Configuration Files”). Therefore, you must not add any of these options if you change the configuration for an instance named `mysqld`.

The following list describes the commands that Instance Manager accepts, with examples.

- `START INSTANCE` *`instance_name`*

  This command attempts to start an offline instance. The command is asynchronous; it does not wait for the instance to start.

  ```
  mysql> START INSTANCE mysqld4;
  Query OK, 0 rows affected (0,00 sec)
  ```

- `STOP INSTANCE` *`instance_name`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This command attempts to stop an instance. The command is synchronous; it waits for the instance to stop.

```
mysql> STOP INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- `SHOW INSTANCES`

  Shows the names and status of all loaded instances.

```
mysql> SHOW INSTANCES;
+---------------+---------+
| instance_name | status  |
+---------------+---------+
| mysqld3       | offline |
| mysqld4       | online  |
| mysqld2       | offline |
+---------------+---------+
```

- `SHOW INSTANCE STATUS` *`instance_name`*

  Shows status and version information for an instance.

```
mysql> SHOW INSTANCE STATUS mysqld3;
+---------------+--------+---------+
| instance_name | status | version |
+---------------+--------+---------+
| mysqld3       | online | unknown |
+---------------+--------+---------+
```

- `SHOW INSTANCE OPTIONS` *`instance_name`*

  Shows the options used by an instance.

```
mysql> SHOW INSTANCE OPTIONS mysqld3;
+---------------+-----------------------------------------------------+
| option_name   | value                                               |
+---------------+-----------------------------------------------------+
| instance_name | mysqld3                                             |
| mysqld-path   | /home/cps/mysql/trees/mysql-4.1/sql/mysqld          |
| port          | 3309                                                |
| socket        | /tmp/mysql.sock3                                    |
| pid-file      | hostname.pid3                                       |
| datadir       | /home/cps/mysql_data/data_dir1/                     |
| language      | /home/cps/mysql/trees/mysql-4.1/sql/share/english |
+---------------+-----------------------------------------------------+
```

- `SHOW` *`instance_name`* `LOG FILES`

  The command lists all log files used by the instance. The result set contains the path to the log file and the log file size. If no log file path is specified in the instance section of the configuration file (for example, `log=/var/mysql.log`), the Instance Manager tries to guess its placement. If Instance Manager is unable to guess the log file placement you should specify the log file location explicitly by using a log option in the appropriate instance section of the configuration file.

```
mysql> SHOW mysqld LOG FILES;
+-------------+-----------------------------------+----------+
| Logfile     | Path                              | Filesize |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+-------------+----------------------------------+----------+
| ERROR LOG   | /home/cps/var/mysql/owlet.err      | 9186     |
| GENERAL LOG | /home/cps/var/mysql/owlet.log      | 471503   |
| SLOW LOG    | /home/cps/var/mysql/owlet-slow.log | 4463     |
+-------------+----------------------------------+----------+
```

Log options are described in Section 5.1.3, “Server Command Options”.

- `SHOW instance_name LOG {ERROR | SLOW | GENERAL} size[,offset_from_end]`

  This command retrieves a portion of the specified log file. Because most users are interested in the latest log messages, the *`size`* parameter defines the number of bytes to retrieve from the end of the log. To retrieve data from the middle of the log file, specify the optional *`offset_from_end`* parameter. The following example retrieves 21 bytes of data, starting 23 bytes before the end of the log file and ending 2 bytes before the end:

  ```
  mysql> SHOW mysqld LOG GENERAL 21, 2;
  +---------------------+
  | Log                 |
  +---------------------+
  | using password: YES |
  +---------------------+
  ```

- `SET instance_name.option_name[=option_value]`

  This command edits the specified instance's configuration section to change or add instance options. The option is added to the section is it is not already present. Otherwise, the new setting replaces the existing one.

  ```
  mysql> SET mysqld2.port=3322;
  Query OK, 0 rows affected (0.00 sec)
  ```

  Changes made to the configuration file do not take effect until the MySQL server is restarted. In addition, these changes are not stored in the instance manager's local cache of instance settings until a `FLUSH INSTANCES` command is executed.

- `UNSET instance_name.option_name`

  This command removes an option from an instance's configuration section.

  ```
  mysql> UNSET mysqld2.port;
  Query OK, 0 rows affected (0.00 sec)
  ```

  Changes made to the configuration file do not take effect until the MySQL server is restarted. In addition, these changes are not stored in the instance manager's local cache of instance settings until a `FLUSH INSTANCES` command is executed.

- `FLUSH INSTANCES`

  This command forces Instance Manager reread the configuration file and to refresh internal structures. This command should be performed after editing the configuration file. The command does not restart instances.

  ```
  mysql> FLUSH INSTANCES;
  Query OK, 0 rows affected (0.04 sec)
  ```

  `FLUSH INSTANCES` is deprecated and will be removed in a future MySQL release.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4.6.11 mysql_convert_table_format — Convert Tables to Use a Given Storage Engine

`mysql_convert_table_format` converts the tables in a database to use a particular storage engine (`MyISAM` by default). `mysql_convert_table_format` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see Section 2.22, "Perl Installation Notes").

Invoke `mysql_convert_table_format` like this:

```
shell> mysql_convert_table_format [options]db_name
```

The *db_name* argument indicates the database containing the tables to be converted.

`mysql_convert_table_format` supports the options described in the following list.

- `--help`

  Display a help message and exit.

- `--force`

  Continue even if errors occur.

- `--host=host_name`

  Connect to the MySQL server on the given host.

- `--password=password`

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, "End-User Guidelines for Password Security". You can use an option file to avoid giving the password on the command line.

- `--port=port_num`

  The TCP/IP port number to use for the connection.

- `--socket=path`

  For connections to `localhost`, the Unix socket file to use.

- `--type=engine_name`

  Specify the storage engine that the tables should be converted to use. The default is `MyISAM` if this option is not given.

- `--user=user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`

  Verbose mode. Print more information about what the program does.

- `--version`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Display version information and exit.

4.6.12 mysql_explain_log — Use EXPLAIN on Statements in Query Log

mysql_explain_log reads its standard input for query log contents. It uses EXPLAIN to analyze SELECT statements found in the input. UPDATE statements are rewritten to SELECT statements and also analyzed with EXPLAIN. mysql_explain_log then displays a summary of its results.

The results may assist you in determining which queries result in table scans and where it would be beneficial to add indexes to your tables.

Invoke mysql_explain_log like this, where *log_file* contains all or part of a MySQL query log:

```
shell> mysql_explain_log [options] < log_file
```

mysql_explain_log understands the following options:

- --help, -?

  Display a help message and exit.

- --date=*YYMMDD*, -d *YYMMDD*

  Select entries from the log only for the given date.

- --host=*host_name*, -h *host_name*

  Connect to the MySQL server on the given host.

- --password=*password*, -p *password*

  The password to use when connecting to the server.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- --printerror=1, -e 1

  Enable error output.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --user=*user_name*, -u *user_name*

  The MySQL user name to use when connecting to the server.

4.6.13 mysql_find_rows — Extract SQL Statements from Files

mysql_find_rows reads files containing SQL statements and extracts statements that match a given regular expression or that contain USE *db_name* or SET statements. The utility was written for use with update log files (as used prior to MySQL 5.0) and as such expects statements to be terminated with semicolon (;) characters. It may be useful with other files that contain SQL statements as long as statements are terminated with semicolons.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Invoke mysql_find_rows like this:

```
shell> mysql_find_rows [options] [file_name ...]
```

Each *file_name* argument should be the name of file containing SQL statements. If no file names are given, mysql_find_rows reads the standard input.

Examples:

```
mysql_find_rows --regexp=problem_table --rows=20 < update.log
mysql_find_rows --regexp=problem_table  update-log.1 update-log.2
```

mysql_find_rows supports the following options:

- --help, --Information

  Display a help message and exit.

- --regexp=*pattern*

  Display queries that match the pattern.

- --rows=*N*

  Quit after displaying *N* queries.

- --skip-use-db

  Do not include USE *db_name* statements in the output.

- --start_row=*N*

  Start output from this row.

4.6.14 mysql_fix_extensions — Normalize Table File Name Extensions

mysql_fix_extensions converts the extensions for MyISAM (or ISAM) table files to their canonical forms. It looks for files with extensions matching any lettercase variant of .frm, .myd, .myi, .isd, and .ism and renames them to have extensions of .frm, .MYD, .MYI, .ISD, and .ISM, respectively. This can be useful after transferring the files from a system with case-insensitive file names (such as Windows) to a system with case-sensitive file names.

Invoke mysql_fix_extensions like this, where *data_dir* is the path name to the MySQL data directory.

```
shell> mysql_fix_extensions data_dir
```

4.6.15 mysql_setpermission — Interactively Set Permissions in Grant Tables

mysql_setpermission is a Perl script that was originally written and contributed by Luuk de Boer. It interactively sets permissions in the MySQL grant tables. mysql_setpermission is written in Perl and requires that the DBI and DBD::mysql Perl modules be installed (see Section 2.22, “Perl Installation Notes”).

Invoke mysql_setpermission like this:

This documentation is for an older version. If you're

```
shell> mysql_setpermission [options]
```

options should be either --help to display the help message, or options that indicate how to connect to the MySQL server. The account used when you connect determines which permissions you have when attempting to modify existing permissions in the grant tables.

mysql_setpermissions also reads options from the [client] and [perl] groups in the .my.cnf file in your home directory, if the file exists.

mysql_setpermission supports the following options:

- --help

  Display a help message and exit.

- --host=*host_name*

  Connect to the MySQL server on the given host.

- --password=*password*

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, "End-User Guidelines for Password Security". You can use an option file to avoid giving the password on the command line.

- --port=*port_num*

  The TCP/IP port number to use for the connection.

- --socket=*path*

  For connections to localhost, the Unix socket file to use.

- --user=*user_name*

  The MySQL user name to use when connecting to the server.

4.6.16 mysql_tableinfo — Generate Database Metadata

mysql_tableinfo creates tables and populates them with database metadata. It uses SHOW DATABASES, SHOW TABLES, SHOW TABLE STATUS, SHOW COLUMNS, and SHOW INDEX to obtain the metadata.

In MySQL 5.0 and up, the INFORMATION_SCHEMA database contains the same kind of information in the SCHEMATA, TABLES, COLUMNS, and STATISTICS tables. See Chapter 19, *INFORMATION_SCHEMA Tables*.

Invoke mysql_tableinfo like this:

```
shell> mysql_tableinfo [options] db_name [db_like [tbl_like]]
```

The *db_name* argument indicates which database mysql_tableinfo should use as the location for the metadata tables. The database will be created if it does not exist. The tables will be named db, tbl (or tbl_status), col, and idx.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the *db_like* or *tbl_like* arguments are given, they are used as patterns and metadata is generated only for databases or tables that match the patterns. These arguments default to % if not given.

Examples:

```
mysql_tableinfo info
mysql_tableinfo info world
mysql_tableinfo info mydb tmp%
```

Each of the commands stores information into tables in the info database. The first stores information for all databases and tables. The second stores information for all tables in the world database. The third stores information for tables in the mydb database that have names matching the pattern tmp%.

mysql_tableinfo supports the following options:

Table 4.14 mysql_tableinfo Options

Format	Description
--clear	Before populating each metadata table, drop it if it exists
--clear-only	Similar to --clear, but exits after dropping the metadata tables to be populated.
--col	Generate column metadata into the col table
--help	Display help message and exit
--host	Connect to MySQL server on given host
--idx	Generate index metadata into the idx table
--password	Password to use when connecting to server -- not optional
--port	TCP/IP port number to use for connection
--prefix	Add prefix_str at the beginning of each metadata table name
--quiet	Be silent except for errors
--socket	For connections to localhost, the Unix socket file to use
--tbl-status	Use SHOW TABLE STATUS instead of SHOW TABLES
--user	The mysql_tableinfo user name to use when connecting to server

- `--help`

  Display a help message and exit.

- `--clear`

  Before populating each metadata table, drop it if it exists.

- `--clear-only`

  Similar to `--clear`, but exits after dropping the metadata tables to be populated.

- `--col`

  Generate column metadata into the `col` table.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--idx`

  Generate index metadata into the `idx` table.

- `--password=password`, `-ppassword`

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs. You can use an option file to avoid giving the password on the command line.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--prefix=prefix_str`

  Add *`prefix_str`* at the beginning of each metadata table name.

- `--quiet`, `-q`

  Be silent except for errors.

- `--socket=path`, `-S path`

  The Unix socket file to use for the connection.

- `--tbl-status`

  Use `SHOW TABLE STATUS` instead of `SHOW TABLES`. This provides more complete information, but is slower.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

4.6.17 `mysql_waitpid` — Kill Process and Wait for Its Termination

`mysql_waitpid` signals a process to terminate and waits for the process to exit. It uses the `kill()` system call and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_waitpid` like this:

```
shell> mysql_waitpid [options] pid wait_time
```

`mysql_waitpid` sends signal 0 to the process identified by *`pid`* and waits up to *`wait_time`* seconds for the process to terminate. *`pid`* and *`wait_time`* must be positive integers.

If process termination occurs within the wait time or the process does not exist, `mysql_waitpid` returns 0. Otherwise, it returns 1.

If the `kill()` system call cannot handle signal 0, `mysql_waitpid()` uses signal 1 instead.

`mysql_waitpid` supports the following options:

- `--help`, `-?`, `-I`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Display a help message and exit.

- `--verbose`, `-v`

  Verbose mode. Display a warning if signal 0 could not be used and signal 1 is used instead.

- `--version`, `-V`

  Display version information and exit.

4.6.18 mysql_zap — Kill Processes That Match a Pattern

`mysql_zap` kills processes that match a pattern. It uses the `ps` command and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_zap` like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the `ps` command contains the pattern. By default, `mysql_zap` asks for confirmation for each process. Respond `y` to kill the process, or `q` to exit `mysql_zap`. For any other response, `mysql_zap` does not attempt to kill the process.

If the *`-signal`* option is given, it specifies the name or number of the signal to send to each process. Otherwise, `mysql_zap` tries first with `TERM` (signal 15) and then with `KILL` (signal 9).

`mysql_zap` supports the following additional options:

- `--help`, `-?`, `-I`

  Display a help message and exit.

- `-f`

  Force mode. `mysql_zap` attempts to kill each process without confirmation.

- `-t`

  Test mode. Display information about each process but do not kill it.

4.7 MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

In shell scripts, you can use the `my_print_defaults` program to parse option files and see what options would be used by a given program. The following example shows the output that `my_print_defaults` might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for

This documentation is for an older version. If you're
This documentation is for an older version. If you're

programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.7.1 msql2mysql — Convert mSQL Programs for Use with MySQL

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The msql2mysql utility performs the conversion of mSQL C API function calls to their MySQL equivalents. msql2mysql converts the input file in place, so make a copy of the original before converting it. For example, use msql2mysql like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine client-prog.c and make any post-conversion revisions that may be necessary.

msql2mysql uses the replace utility to make the function name substitutions. See Section 4.8.2, “replace — A String-Replacement Utility”.

4.7.2 mysql_config — Display Options for Compiling Clients

mysql_config provides you with useful information for compiling your MySQL client and connecting it to MySQL. It is a shell script, so it is available only on Unix and Unix-like systems.

mysql_config supports the following options.

- --cflags

  Compiler flags to find include files and critical compiler flags and defines used when compiling the libmysqlclient library. The options returned are tied to the specific compiler that was used when the library was created and might clash with the settings for your own compiler. Use --include for more portable options that contain only include paths.

- --include

  Compiler options to find MySQL include files.

- --libmysqld-libs, --embedded

  Libraries and options required to link with the MySQL embedded server.

- --libs

  Libraries and options required to link with the MySQL client library.

- --libs_r

  Libraries and options required to link with the thread-safe MySQL client library.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--port`

  The default TCP/IP port number, defined when configuring MySQL.

- `--socket`

  The default Unix socket file, defined when configuring MySQL.

- `--version`

  Version number for the MySQL distribution.

If you invoke `mysql_config` with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
                    -lz -lcrypt -lnsl -lm]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lz -lpthread -lcrypt -lnsl -lm -lpthread]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [5.0.96]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld
                    -lz -lpthread -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use `mysql_config` within a command line using backticks to include the output that it produces for particular options. For example, to compile and link a MySQL client program, use `mysql_config` as follows:

```
gcc -c `mysql_config --cflags` progname.c
gcc -o progname progname.o `mysql_config --libs`
```

4.7.3 `my_print_defaults` — Display Options from Option Files

`my_print_defaults` displays the options that are present in option groups of option files. The output indicates what options will be used by programs that read the specified option groups. For example, the `mysqlcheck` program reads the `[mysqlcheck]` and `[client]` option groups. To see what options are present in those groups in the standard option files, invoke `my_print_defaults` like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=secret
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

`my_print_defaults` supports the following options.

- `--help`, `-?`

  Display a help message and exit.

- `--config-file=`*`file_name`*, `--defaults-file=`*`file_name`*, `-c` *`file_name`*

This documentation is for an older version. If you're

Read only the given option file.

- `--debug=debug_options`, `-# debug_options`

  Write a debugging log. A typical `debug_options` string is `d:t:o,file_name`. The default is `d:t:o,/tmp/my_print_defaults.trace`.

- `--defaults-extra-file=file_name`, `--extra-file=file_name`, `-e file_name`

  Read this option file after the global option file but (on Unix) before the user option file.

- `--defaults-group-suffix=suffix`, `-g suffix`

  In addition to the groups named on the command line, read groups that have the given suffix.

- `--no-defaults`, `-n`

  Return an empty string.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

4.7.4 `resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols

`resolve_stack_dump` resolves a numeric stack dump to symbols.

Invoke `resolve_stack_dump` like this:

```
shell> resolve_stack_dump [options] symbols_file [numeric_dump_file]
```

The symbols file should include the output from the `nm --numeric-sort mysqld` command. The numeric dump file should contain a numeric stack track from `mysqld`. If no numeric dump file is named on the command line, the stack trace is read from the standard input.

`resolve_stack_dump` supports the following options.

- `--help`, `-h`

  Display a help message and exit.

- `--numeric-dump-file=file_name`, `-n file_name`

  Read the stack trace from the given file.

- `--symbols-file=file_name`, `-s file_name`

  Use the given symbols file.

- `--version`, `-V`

  Display version information and exit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For more information, see Section 21.3.1.5, “Using a Stack Trace”.

4.8 Miscellaneous Programs

4.8.1 `perror` — Explain Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the `perror` utility.

`perror` prints a description for a system error code or for a storage engine (table handler) error code.

Invoke `perror` like this:

```
shell> perror [options] errorcode ...
```

Example:

```
shell> perror 13 64
OS error code  13:  Permission denied
OS error code  64:  Machine is not on the network
```

To obtain the error message for a MySQL Cluster error code, invoke `perror` with the `--ndb` option:

```
shell> perror --ndb errorcode
```

The meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

`perror` supports the following options.

- `--help`, `--info`, `-I`, `-?`

  Display a help message and exit.

- `--ndb`

  Print the error message for a MySQL Cluster error code.

- `--silent`, `-s`

  Silent mode. Print only the error message.

- `--verbose`, `-v`

  Verbose mode. Print error code and message. This is the default behavior.

- `--version`, `-V`

  Display version information and exit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4.8.2 replace — A String-Replacement Utility

The replace utility program changes strings in place in files or on the standard input.

Invoke replace in one of the following ways:

```
shell> replace from to [from to] ... -- file_name [file_name] ...
shell> replace from to [from to] ... < file_name
```

from represents a string to look for and *to* represents its replacement. There can be one or more pairs of strings.

Use the -- option to indicate where the string-replacement list ends and the file names begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. *replace* prints a message indicating which of the input files it actually modifies.

If the -- option is not given, replace reads the standard input and writes to the standard output.

replace uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps a and b in the given files, file1 and file2:

```
shell> replace a b b a -- file1 file2 ...
```

The replace program is used by msql2mysql. See Section 4.7.1, “msql2mysql — Convert mSQL Programs for Use with MySQL”.

replace supports the following options.

- -?, -I

  Display a help message and exit.

- *-#debug_options*

  Enable debugging.

- -s

  Silent mode. Print less information what the program does.

- -v

  Verbose mode. Print more information about what the program does.

- -V

  Display version information and exit.

4.8.3 resolveip — Resolve Host name to IP Address or Vice Versa

The resolveip utility resolves host names to IP addresses and vice versa.

Invoke resolveip like this:

```
shell> resolveip [options] {host_name|ip-addr} ...
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

resolveip supports the following options.

- --help, --info, -?, -I

  Display a help message and exit.

- --silent, -s

  Silent mode. Produce less output.

- --version, -V

  Display version information and exit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 5 MySQL Server Administration

Table of Contents

End of Product Lifecycle. Active development for MySQL Database Server version 5.0 has ended. Oracle offers various support offerings which may be of interest. For details and more information, see the MySQL section of the Lifetime Support Policy for Oracle Technology Products (http://www.oracle.com/us/support/lifetime-support/index.html). Please consider upgrading to a recent version.

MySQL Server (`mysqld`) is the main program that does most of the work in a MySQL installation. This chapter provides an overview of MySQL Server and covers general server administration:

- Server configuration
- The data directory, particularly the `mysql` system database
- The server log files
- Management of multiple servers on a single machine

For additional information on administrative topics, see also:

- Chapter 6, *Security*
- Chapter 7, *Backup and Recovery*
- Chapter 16, *Replication*

5.1 The MySQL Server

This documentation is for an older version. If you're

This documentation is for an older version. If you're

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports
- Server system variables
- Server status variables
- How to set the server SQL mode
- The server shutdown process

Note

Not all storage engines are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines your MySQL server installation supports, see Section 13.7.5.13, “SHOW ENGINES Syntax”.

5.1.1 Server Option and Variable Reference

The following table provides a list of all the command line options, server and status variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with notification of where each option/variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding server system or status variable, the variable name is noted immediately below the corresponding option. For status variables, the scope of the variable is shown (Scope) as either global, session, or both. Please see the corresponding sections for details on setting and using the options and variables. Where appropriate, a direct link to further information on the item as available.

For a version of this table that is specific to MySQL Cluster, see Section 17.3.2.5, “MySQL Cluster mysqld Option and Variable Reference”.

Table 5.1 Option/Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
abort-slave-event-count	Yes	Yes				
Aborted_clients				Yes	Global	No
Aborted_connects				Yes	Global	No
allow-suspicious-udfs	Yes	Yes				
ansi	Yes	Yes				
auto_increment_increment			Yes		Both	Yes
auto_increment_offset			Yes		Both	Yes
autocommit			Yes		Session	Yes
automatic_sp_privileges			Yes		Global	Yes
back_log			Yes		Global	No
basedir	Yes	Yes	Yes		Global	No
bdb_cache_size			Yes		Global	No
bdb-home	Yes	Yes			Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
- *Variable*: bdb_home			Yes		Global	No
bdb-lock-detect	Yes	Yes	Yes		Global	No
bdb_log_buffer_size			Yes		Global	No
bdb-logdir	Yes	Yes			Global	No
- *Variable*: bdb_logdir			Yes		Global	No
bdb_max_lock			Yes		Global	No
bdb-no-recover	Yes	Yes				
bdb-no-sync	Yes	Yes				
bdb-shared-data	Yes	Yes			Global	No
- *Variable*: bdb_shared_data			Yes		Global	No
bdb-tmpdir	Yes	Yes			Global	No
- *Variable*: bdb_tmpdir			Yes		Global	No
big-tables	Yes	Yes			Session	Yes
- *Variable*: big_tables			Yes		Session	Yes
bind-address	Yes	Yes				
Binlog_cache_disk_use				Yes	Global	No
binlog_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_cache_use				Yes	Global	No
binlog-do-db	Yes	Yes				
binlog-ignore-db	Yes	Yes				
bootstrap	Yes	Yes				
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
Bytes_received				Yes	Both	No
Bytes_sent				Yes	Both	No
character_set_client			Yes		Both	Yes
character-set-client-handshake	Yes	Yes				
character_set_connection			Yes		Both	Yes
character_set_database[a]			Yes		Both	Yes
character-set-filesystem	Yes	Yes			Both	Yes
- *Variable*: character_set_filesystem			Yes		Both	Yes
character_set_results			Yes		Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
character-set-server	Yes	Yes			Both	Yes
- *Variable*: character_set_server			Yes		Both	Yes
character_set_system			Yes		Global	No
character-sets-dir	Yes	Yes			Global	No
- *Variable*: character_sets_dir			Yes		Global	No
chroot	Yes	Yes				
collation_connection			Yes		Both	Yes
collation_database[b]			Yes		Both	Yes
collation-server	Yes	Yes			Both	Yes
- *Variable*: collation_server			Yes		Both	Yes
Com_admin_commands				Yes	Both	No
Com_alter_db				Yes	Both	No
Com_alter_event				Yes	Both	No
Com_alter_table				Yes	Both	No
Com_analyze				Yes	Both	No
Com_backup_table				Yes	Both	No
Com_begin				Yes	Both	No
Com_call_procedure				Yes	Both	No
Com_change_db				Yes	Both	No
Com_change_master				Yes	Both	No
Com_check				Yes	Both	No
Com_checksum				Yes	Both	No
Com_commit				Yes	Both	No
Com_create_db				Yes	Both	No
Com_create_event				Yes	Both	No
Com_create_function				Yes	Both	No
Com_create_index				Yes	Both	No
Com_create_table				Yes	Both	No
Com_create_user				Yes	Both	No
Com_dealloc_sql				Yes	Both	No
Com_delete				Yes	Both	No
Com_delete_multi				Yes	Both	No
Com_do				Yes	Both	No
Com_drop_db				Yes	Both	No
Com_drop_event				Yes	Both	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Com_drop_function				Yes	Both	No
Com_drop_index				Yes	Both	No
Com_drop_table				Yes	Both	No
Com_drop_user				Yes	Both	No
Com_execute_sql				Yes	Both	No
Com_flush				Yes	Both	No
Com_grant				Yes	Both	No
Com_ha_close				Yes	Both	No
Com_ha_open				Yes	Both	No
Com_ha_read				Yes	Both	No
Com_help				Yes	Both	No
Com_insert				Yes	Both	No
Com_insert_select				Yes	Both	No
Com_kill				Yes	Both	No
Com_load				Yes	Both	No
Com_lock_tables				Yes	Both	No
Com_optimize				Yes	Both	No
Com_preload_keys				Yes	Both	No
Com_prepare_sql				Yes	Both	No
Com_purge				Yes	Both	No
Com_purge_before_date				Yes	Both	No
Com_rename_table				Yes	Both	No
Com_repair				Yes	Both	No
Com_replace				Yes	Both	No
Com_replace_select				Yes	Both	No
Com_reset				Yes	Both	No
Com_restore_table				Yes	Both	No
Com_revoke				Yes	Both	No
Com_revoke_all				Yes	Both	No
Com_rollback				Yes	Both	No
Com_savepoint				Yes	Both	No
Com_select				Yes	Both	No
Com_set_option				Yes	Both	No
Com_show_binlog_events				Yes	Both	No
Com_show_binlogs				Yes	Both	No
Com_show_charsets				Yes	Both	No
Com_show_collations				Yes	Both	No
Com_show_column_types				Yes	Both	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Com_show_create_db				Yes	Both	No
Com_show_create_event				Yes	Both	No
Com_show_create_table				Yes	Both	No
Com_show_databases				Yes	Both	No
Com_show_engine_logs				Yes	Both	No
Com_show_engine_mutex				Yes	Both	No
Com_show_engine_status				Yes	Both	No
Com_show_errors				Yes	Both	No
Com_show_events				Yes	Both	No
Com_show_fields				Yes	Both	No
Com_show_grants				Yes	Both	No
Com_show_innodb_status				Yes	Both	No
Com_show_keys				Yes	Both	No
Com_show_logs				Yes	Both	No
Com_show_master_status				Yes	Both	No
Com_show_ndb_status				Yes	Both	No
Com_show_new_master				Yes	Both	No
Com_show_open_tables				Yes	Both	No
Com_show_plugins				Yes	Both	No
Com_show_privileges				Yes	Both	No
Com_show_processlist				Yes	Both	No
Com_show_slave_hosts				Yes	Both	No
Com_show_slave_status				Yes	Both	No
Com_show_status				Yes	Both	No
Com_show_storage_engines				Yes	Both	No
Com_show_tables				Yes	Both	No
Com_show_triggers				Yes	Both	No
Com_show_variables				Yes	Both	No
Com_show_warnings				Yes	Both	No
Com_slave_start				Yes	Both	No
Com_slave_stop				Yes	Both	No
Com_stmt_close				Yes	Both	No
Com_stmt_execute				Yes	Both	No
Com_stmt_fetch				Yes	Both	No
Com_stmt_prepare				Yes	Both	No
Com_stmt_reset				Yes	Both	No
Com_stmt_send_long_data				Yes	Both	No
Com_truncate				Yes	Both	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Com_unlock_tables				Yes	Both	No
Com_update				Yes	Both	No
Com_update_multi				Yes	Both	No
Com_xa_commit				Yes	Both	No
Com_xa_end				Yes	Both	No
Com_xa_prepare				Yes	Both	No
Com_xa_recover				Yes	Both	No
Com_xa_rollback				Yes	Both	No
Com_xa_start				Yes	Both	No
completion_type	Yes	Yes	Yes		Both	Yes
Compression				Yes	Session	No
concurrent_insert	Yes	Yes	Yes		Global	Yes
connect_timeout	Yes	Yes	Yes		Global	Yes
Connections				Yes	Global	No
console	Yes	Yes				
core-file	Yes	Yes				
Created_tmp_disk_tables				Yes	Both	No
Created_tmp_files				Yes	Global	No
Created_tmp_tables				Yes	Both	No
datadir	Yes	Yes	Yes		Global	No
date_format			Yes		Both	No
datetime_format			Yes		Both	No
debug	Yes	Yes	Yes		Both	Yes
default-character-set	Yes	Yes				
default-storage-engine	Yes	Yes	Yes		Both	Yes
default-table-type	Yes	Yes				
default-time-zone	Yes	Yes				
default_week_format	Yes	Yes	Yes		Both	Yes
defaults-extra-file	Yes					
defaults-file	Yes					
defaults-group-suffix	Yes					
delay-key-write	Yes	Yes			Global	Yes
- *Variable*: delay_key_write			Yes		Global	Yes
Delayed_errors				Yes	Global	No
delayed_insert_limit	Yes	Yes	Yes		Global	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Delayed_insert_threads				Yes	Global	No
delayed_insert_timeout	Yes	Yes	Yes		Global	Yes
delayed_queue_size	Yes	Yes	Yes		Global	Yes
Delayed_writes				Yes	Global	No
des-key-file	Yes	Yes				
disconnect-slave-event-count	Yes	Yes				
div_precision_increment	Yes	Yes	Yes		Both	Yes
enable-locking	Yes	Yes				
enable-named-pipe	Yes	Yes				
- *Variable*: named_pipe						
enable-pstack	Yes	Yes				
engine-condition-pushdown	Yes	Yes			Both	Yes
- *Variable*: engine_condition_pushdown			Yes		Both	Yes
error_count			Yes		Session	No
exit-info	Yes	Yes				
expire_logs_days	Yes	Yes	Yes		Global	Yes
external-locking	Yes	Yes				
- *Variable*: skip_external_locking						
flush	Yes	Yes	Yes		Global	Yes
Flush_commands				Yes	Global	No
flush_time	Yes	Yes	Yes		Global	Yes
foreign_key_checks			Yes		Session	Yes
ft_boolean_syntax	Yes	Yes	Yes		Global	Yes
ft_max_word_len	Yes	Yes	Yes		Global	No
ft_min_word_len	Yes	Yes	Yes		Global	No
ft_query_expansion_limit	Yes	Yes	Yes		Global	No
ft_stopword_file	Yes	Yes	Yes		Global	No
gdb	Yes	Yes				
group_concat_max_len	Yes	Yes	Yes		Both	Yes
Handler_commit				Yes	Both	No
Handler_delete				Yes	Both	No
Handler_discover				Yes	Both	No
Handler_prepare				Yes	Both	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Handler_read_first				Yes	Both	No
Handler_read_key				Yes	Both	No
Handler_read_next				Yes	Both	No
Handler_read_prev				Yes	Both	No
Handler_read_rnd				Yes	Both	No
Handler_read_rnd_next				Yes	Both	No
Handler_rollback				Yes	Both	No
Handler_savepoint				Yes	Both	No
Handler_savepoint_rollback				Yes	Both	No
Handler_update				Yes	Both	No
Handler_write				Yes	Both	No
have_archive			Yes		Global	No
have_bdb			Yes		Global	No
have_blackhole_engine			Yes		Global	No
have_community_features			Yes		Global	No
have_compress			Yes		Global	No
have_crypt			Yes		Global	No
have_csv			Yes		Global	No
have_example_engine			Yes		Global	No
have_federated_engine			Yes		Global	No
have_geometry			Yes		Global	No
have_innodb			Yes		Global	No
have_isam			Yes		Global	No
have_merge_engine			Yes		Global	No
have_ndbcluster			Yes		Global	No
have_openssl			Yes		Global	No
have_profiling			Yes		Global	No
have_query_cache			Yes		Global	No
have_raid			Yes		Global	No
have_rtree_keys			Yes		Global	No
have_ssl			Yes		Global	No
have_symlink			Yes		Global	No
help	Yes	Yes				
hostname			Yes		Global	No
identity			Yes		Session	Yes
init_connect	Yes	Yes	Yes		Global	Yes
init-file	Yes	Yes			Global	No
- *Variable*: init_file			Yes		Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
init-rpl-role	Yes	Yes				
init_slave	Yes	Yes	Yes		Global	Yes
innodb	Yes	Yes				
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	No
innodb_additional_mem_pool_size	Yes	Yes	Yes		Global	No
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_awe_mem_mb	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_ahead_seq				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_checksums	Yes	Yes	Yes		Global	No
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
innodb_file_io_threads	Yes	Yes	Yes		Global	No
innodb_file_per_table	Yes	Yes	Yes		Global	No
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_lock_wait_timeout	Yes	Yes	Yes		Global	No
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_arch_dir	Yes	Yes	Yes		Global	No
innodb_log_archive	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
Innodb_log_waits				Yes	Global	No
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_mirrored_log_groups	Yes	Yes	Yes		Global	No
innodb_open_files	Yes	Yes	Yes		Global	No
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
Innodb_page_size				Yes	Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
innodb-safe-binlog	Yes	Yes				
innodb-status-file	Yes	Yes				
innodb_support_xa	Yes	Yes	Yes		Both	Yes
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_use_legacy_cardinality_algorithm	Yes	Yes	Yes		Global	Yes
insert_id			Yes		Session	Yes
install	Yes					
install-manual	Yes					
interactive_timeout	Yes	Yes	Yes		Both	Yes
join_buffer_size	Yes	Yes	Yes		Both	Yes
keep_files_on_create	Yes	Yes	Yes		Both	Yes
Key_blocks_not_flushed				Yes	Global	No
Key_blocks_unused				Yes	Global	No
Key_blocks_used				Yes	Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes
key_cache_age_threshold	Yes	Yes	Yes		Global	Yes
key_cache_block_size	Yes	Yes	Yes		Global	Yes
key_cache_division_limit	Yes	Yes	Yes		Global	Yes
Key_read_requests				Yes	Global	No
Key_reads				Yes	Global	No
Key_write_requests				Yes	Global	No
Key_writes				Yes	Global	No
language	Yes	Yes	Yes		Global	No
large_files_support			Yes		Global	No
large_page_size			Yes		Global	No
large-pages	Yes	Yes			Global	No
- *Variable*: large_pages			Yes		Global	No
last_insert_id			Yes		Session	Yes
Last_query_cost				Yes	Session	No
lc_time_names			Yes		Both	Yes
license			Yes		Global	No
local_infile			Yes		Global	Yes
local-service	Yes					

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
locked_in_memory			Yes		Global	No
log	Yes	Yes	Yes		Global	No
log-bin	Yes	Yes	Yes		Global	No
log_bin			Yes		Global	No
log-bin-index	Yes	Yes				
log-bin-trust-function-creators	Yes	Yes			Global	Yes
- *Variable*: log_bin_trust_function_creators			Yes		Global	Yes
log-bin-trust-routine-creators	Yes	Yes			Global	Yes
- *Variable*: log_bin_trust_routine_creators			Yes		Global	Yes
log-error	Yes	Yes			Global	No
- *Variable*: log_error			Yes		Global	No
log-isam	Yes	Yes				
log-queries-not-using-indexes	Yes	Yes			Global	Yes
- *Variable*: log_queries_not_using_indexes			Yes		Global	Yes
log-short-format	Yes	Yes				
log-slave-updates	Yes	Yes			Global	No
- *Variable*: log_slave_updates			Yes		Global	No
log_slave_updates	Yes	Yes	Yes		Global	No
log-slow-admin-statements	Yes	Yes				
log-slow-queries	Yes	Yes			Global	No
- *Variable*: log_slow_queries			Yes		Global	No
log-tc	Yes	Yes				
log-tc-size	Yes	Yes				
log-warnings	Yes	Yes			Both	Yes
- *Variable*: log_warnings			Yes		Both	Yes
long_query_time	Yes	Yes	Yes		Both	Yes
low-priority-updates	Yes	Yes			Both	Yes
- *Variable*: low_priority_updates			Yes		Both	Yes
lower_case_file_system			Yes		Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
lower_case_table_names	Yes	Yes	Yes		Global	No
master-connect-retry	Yes	Yes				
master-host	Yes	Yes				
master-info-file	Yes	Yes				
master-password	Yes	Yes				
master-port	Yes	Yes				
master-retry-count	Yes	Yes				
master-ssl	Yes	Yes				
master-ssl-ca	Yes	Yes				
master-ssl-capath	Yes	Yes				
master-ssl-cert	Yes	Yes				
master-ssl-cipher	Yes	Yes				
master-ssl-key	Yes	Yes				
master-user	Yes	Yes				
max_allowed_packet	Yes	Yes	Yes		Both	Yes
max_binlog_cache_size	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events	Yes	Yes				
max_binlog_size	Yes	Yes	Yes		Global	Yes
max_connect_errors	Yes	Yes	Yes		Global	Yes
max_connections	Yes	Yes	Yes		Global	Yes
max_delayed_threads	Yes	Yes	Yes		Both	Yes
max_error_count	Yes	Yes	Yes		Both	Yes
max_heap_table_size	Yes	Yes	Yes		Both	Yes
max_insert_delayed_threads			Yes		Both	Yes
max_join_size	Yes	Yes	Yes		Both	Yes
max_length_for_sort_data	Yes	Yes	Yes		Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes		Global	Yes
max_relay_log_size	Yes	Yes	Yes		Global	Yes
max_seeks_for_key	Yes	Yes	Yes		Both	Yes
max_sort_length	Yes	Yes	Yes		Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes		Both	Yes
max_tmp_tables			Yes		Both	Yes
Max_used_connections				Yes	Global	No
max_user_connections	Yes	Yes	Yes		Varies	Yes
max_write_lock_count	Yes	Yes	Yes		Global	Yes
memlock	Yes	Yes				

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
- *Variable*: locked_in_memory						
merge	Yes	Yes				
multi_range_count	Yes	Yes	Yes		Both	Yes
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Yes
myisam_max_extra_sort_file_size	Yes	Yes	Yes		Global	No
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Yes
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam-recover	Yes	Yes				
- *Variable*: myisam_recover_options						
myisam_recover_options			Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Yes
myisam_stats_method	Yes	Yes	Yes		Both	Yes
named_pipe			Yes		Global	No
ndb_autoincrement_prefetch_sz	Yes	Yes	Yes		Both	Yes
ndb_cache_check_time	Yes	Yes	Yes		Global	Yes
Ndb_cluster_node_id				Yes	Both	No
Ndb_config_from_host				Yes	Both	No
Ndb_config_from_port				Yes	Both	No
ndb-connectstring	Yes	Yes				
ndb_force_send	Yes	Yes	Yes		Both	Yes
ndb_index_stat_cache_entries	Yes	Yes	Yes		Both	Yes
ndb_index_stat_enable	Yes	Yes	Yes		Both	Yes
ndb_index_stat_update_freq	Yes	Yes	Yes		Both	Yes
ndb-mgmd-host	Yes	Yes				
ndb-nodeid	Yes	Yes		Yes	Global	No
ndb_optimized_node_selection	Yes	Yes	Yes		Global	No
ndb_report_thresh_binlog_epoch_slip	Yes					
ndb_report_thresh_binlog_mem_usage	Yes					
ndb_use_exact_count			Yes		Both	Yes
ndb_use_transactions	Yes	Yes	Yes		Both	Yes
ndbcluster	Yes	Yes				
- *Variable*: have_ndbcluster						
net_buffer_length	Yes	Yes	Yes		Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
net_read_timeout	Yes	Yes	Yes		Both	Yes
net_retry_count	Yes	Yes	Yes		Both	Yes
net_write_timeout	Yes	Yes	Yes		Both	Yes
new	Yes	Yes	Yes		Both	Yes
no-defaults	Yes					
Not_flushed_delayed_rows				Yes	Global	No
old_passwords			Yes		Both	Yes
old-style-user-limits	Yes	Yes				
one-thread	Yes	Yes				
Open_files				Yes	Global	No
open-files-limit	Yes	Yes			Global	No
- *Variable*: open_files_limit			Yes		Global	No
Open_streams				Yes	Global	No
Open_tables				Yes	Both	No
Opened_tables				Yes	Both	No
optimizer_prune_level	Yes	Yes	Yes		Both	Yes
optimizer_search_depth	Yes	Yes	Yes		Both	Yes
pid-file	Yes	Yes			Global	No
- *Variable*: pid_file			Yes		Global	No
plugin_dir	Yes	Yes	Yes		Global	No
port	Yes	Yes	Yes		Global	No
port-open-timeout	Yes	Yes				
preload_buffer_size	Yes	Yes	Yes		Both	Yes
Prepared_stmt_count				Yes	Global	No
prepared_stmt_count			Yes		Global	No
print-defaults	Yes					
profiling			Yes		Session	Yes
profiling_history_size	Yes	Yes	Yes		Both	Yes
protocol_version			Yes		Global	No
pseudo_thread_id			Yes		Session	Yes
Qcache_free_blocks				Yes	Global	No
Qcache_free_memory				Yes	Global	No
Qcache_hits				Yes	Global	No
Qcache_inserts				Yes	Global	No
Qcache_lowmem_prunes				Yes	Global	No
Qcache_not_cached				Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Qcache_queries_in_cache				Yes	Global	No
Qcache_total_blocks				Yes	Global	No
Queries				Yes	Both	No
query_alloc_block_size	Yes	Yes	Yes		Both	Yes
query_cache_limit	Yes	Yes	Yes		Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes		Global	Yes
query_cache_size	Yes	Yes	Yes		Global	Yes
query_cache_type	Yes	Yes	Yes		Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes		Both	Yes
query_prealloc_size	Yes	Yes	Yes		Both	Yes
Questions				Yes	Both	No
rand_seed1			Yes		Session	Yes
rand_seed2			Yes		Session	Yes
range_alloc_block_size	Yes	Yes	Yes		Both	Yes
read_buffer_size	Yes	Yes	Yes		Both	Yes
read_only	Yes	Yes	Yes		Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes		Both	Yes
relay-log	Yes	Yes			Global	No
- *Variable*: relay_log			Yes		Global	No
relay-log-index	Yes	Yes			Global	No
- *Variable*: relay_log_index			Yes		Global	No
relay_log_index	Yes	Yes	Yes		Global	No
relay-log-info-file	Yes	Yes				
- *Variable*: relay_log_info_file						
relay_log_info_file	Yes	Yes	Yes		Global	No
relay_log_purge	Yes	Yes	Yes		Global	Yes
relay_log_space_limit	Yes	Yes	Yes		Global	No
remove	Yes					
replicate-do-db	Yes	Yes				
replicate-do-table	Yes	Yes				
replicate-ignore-db	Yes	Yes				
replicate-ignore-table	Yes	Yes				
replicate-rewrite-db	Yes	Yes				

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
replicate-same-server-id	Yes	Yes				
replicate-wild-do-table	Yes	Yes				
replicate-wild-ignore-table	Yes	Yes				
report-host	Yes	Yes				
report-password	Yes	Yes				
report-port	Yes	Yes				
report-user	Yes	Yes				
rpl_recovery_rank			Yes		Global	Yes
Rpl_status				Yes	Global	No
safe-mode	Yes	Yes				
safe-show-database	Yes	Yes				
safe-user-create	Yes	Yes				
safemalloc-mem-limit	Yes	Yes				
secure-auth	Yes	Yes			Global	Yes
- *Variable*: secure_auth			Yes		Global	Yes
secure-file-priv	Yes	Yes			Global	No
- *Variable*: secure_file_priv			Yes		Global	No
Select_full_join				Yes	Both	No
Select_full_range_join				Yes	Both	No
Select_range				Yes	Both	No
Select_range_check				Yes	Both	No
Select_scan				Yes	Both	No
server-id	Yes	Yes			Global	Yes
- *Variable*: server_id			Yes		Global	Yes
set-variable	Yes	Yes				
shared_memory	Yes	Yes	Yes		Global	No
shared_memory_base_name	Yes	Yes	Yes		Global	No
show-slave-auth-info	Yes	Yes				
skip-bdb	Yes	Yes				
skip-character-set-client-handshake	Yes	Yes				

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
skip-concurrent-insert	Yes	Yes				
- *Variable*: concurrent_insert						
skip_external_locking	Yes	Yes	Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-host-cache	Yes	Yes				
skip-locking	Yes	Yes				
skip-merge	Yes	Yes				
skip-name-resolve	Yes	Yes				
skip-ndbcluster	Yes	Yes				
skip-networking	Yes	Yes			Global	No
- *Variable*: skip_networking			Yes		Global	No
skip-new	Yes	Yes				
skip-safemalloc	Yes	Yes				
skip-show-database	Yes	Yes			Global	No
- *Variable*: skip_show_database			Yes		Global	No
skip-slave-start	Yes	Yes				
skip-ssl	Yes	Yes				
skip-stack-trace	Yes	Yes				
skip-symbolic-links	Yes					
skip-symlink	Yes	Yes				
skip-sync-bdb-logs	Yes	Yes	Yes		Global	No
skip-thread-priority	Yes	Yes				
slave_compressed_protocol	Yes	Yes	Yes		Global	Yes
slave-load-tmpdir	Yes	Yes			Global	No
- *Variable*: slave_load_tmpdir			Yes		Global	No
slave-net-timeout	Yes	Yes			Global	Yes
- *Variable*: slave_net_timeout			Yes		Global	Yes
Slave_open_temp_tables				Yes	Global	No
Slave_retried_transactions				Yes	Global	No
Slave_running				Yes	Global	No
slave-skip-errors	Yes	Yes			Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
- *Variable*: slave_skip_errors			Yes		Global	No
slave_transaction_retries	Yes	Yes	Yes		Global	Yes
Slow_launch_threads				Yes	Both	No
slow_launch_time	Yes	Yes	Yes		Global	Yes
Slow_queries				Yes	Both	No
socket	Yes	Yes	Yes		Global	No
sort_buffer_size	Yes	Yes	Yes		Both	Yes
Sort_merge_passes				Yes	Both	No
Sort_range				Yes	Both	No
Sort_rows				Yes	Both	No
Sort_scan				Yes	Both	No
sporadic-binlog-dump-fail	Yes	Yes				
sql_auto_is_null			Yes		Session	Yes
sql_big_selects			Yes		Session	Yes
sql_big_tables			Yes		Session	Yes
sql_buffer_result			Yes		Session	Yes
sql_log_bin			Yes		Session	Yes
sql_log_off			Yes		Session	Yes
sql_log_update			Yes		Session	Yes
sql_low_priority_updates			Yes		Both	Yes
sql_max_join_size			Yes		Both	Yes
sql-mode	Yes	Yes			Both	Yes
- *Variable*: sql_mode			Yes		Both	Yes
sql_notes			Yes		Session	Yes
sql_quote_show_create			Yes		Session	Yes
sql_safe_updates			Yes		Session	Yes
sql_select_limit			Yes		Both	Yes
sql_slave_skip_counter			Yes		Global	Yes
sql_warnings			Yes		Session	Yes
ssl	Yes	Yes				
Ssl_accept_renegotiates				Yes	Global	No
Ssl_accepts				Yes	Global	No
ssl-ca	Yes	Yes			Global	No
- *Variable*: ssl_ca			Yes		Global	No
Ssl_callback_cache_hits				Yes	Global	No
ssl-capath	Yes	Yes			Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
- *Variable*: ssl_capath			Yes		Global	No
ssl-cert	Yes	Yes			Global	No
- *Variable*: ssl_cert			Yes		Global	No
Ssl_cipher				Yes	Both	No
ssl-cipher	Yes	Yes			Global	No
- *Variable*: ssl_cipher			Yes		Global	No
Ssl_cipher_list				Yes	Both	No
Ssl_client_connects				Yes	Global	No
Ssl_connect_renegotiates				Yes	Global	No
Ssl_ctx_verify_depth				Yes	Global	No
Ssl_ctx_verify_mode				Yes	Global	No
Ssl_default_timeout				Yes	Both	No
Ssl_finished_accepts				Yes	Global	No
Ssl_finished_connects				Yes	Global	No
ssl-key	Yes	Yes			Global	No
- *Variable*: ssl_key			Yes		Global	No
Ssl_session_cache_hits				Yes	Global	No
Ssl_session_cache_misses				Yes	Global	No
Ssl_session_cache_mode				Yes	Global	No
Ssl_session_cache_overflows				Yes	Global	No
Ssl_session_cache_size				Yes	Global	No
Ssl_session_cache_timeouts				Yes	Global	No
Ssl_sessions_reused				Yes	Both	No
Ssl_used_session_cache_entries				Yes	Global	No
Ssl_verify_depth				Yes	Both	No
Ssl_verify_mode				Yes	Both	No
Ssl_version				Yes	Both	No
standalone	Yes	Yes				
storage_engine			Yes		Both	Yes
symbolic-links	Yes	Yes				
sync-bdb-logs	Yes	Yes	Yes		Global	No
sync_binlog	Yes	Yes	Yes		Global	Yes
sync_frm	Yes	Yes	Yes		Global	Yes
sysdate-is-now	Yes	Yes				
system_time_zone			Yes		Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
table_cache	Yes	Yes	Yes		Global	Yes
table_lock_wait_timeout	Yes	Yes	Yes		Global	Yes
Table_locks_immediate				Yes	Global	No
Table_locks_waited				Yes	Global	No
table_type			Yes		Both	Yes
tc-heuristic-recover	Yes	Yes				
Tc_log_max_pages_used				Yes	Global	No
Tc_log_page_size				Yes	Global	No
Tc_log_page_waits				Yes	Global	No
temp-pool	Yes	Yes				
thread_cache_size	Yes	Yes	Yes		Global	Yes
thread_concurrency	Yes	Yes	Yes		Global	No
thread_stack	Yes	Yes	Yes		Global	No
Threads_cached				Yes	Global	No
Threads_connected				Yes	Global	No
Threads_created				Yes	Global	No
Threads_running				Yes	Global	No
time_format			Yes		Both	No
time_zone			Yes		Both	Yes
timed_mutexes	Yes	Yes	Yes		Global	Yes
timestamp			Yes		Session	Yes
tmp_table_size	Yes	Yes	Yes		Both	Yes
tmpdir	Yes	Yes	Yes		Global	No
transaction_alloc_block_size	Yes	Yes	Yes		Both	Yes
transaction-isolation	Yes	Yes				
- *Variable*: tx_isolation						
transaction_prealloc_size	Yes	Yes	Yes		Both	Yes
tx_isolation			Yes		Both	Yes
unique_checks			Yes		Session	Yes
updatable_views_with_limit	Yes	Yes	Yes		Both	Yes
Uptime				Yes	Global	No
Uptime_since_flush_status				Yes	Global	No
user	Yes	Yes				
verbose	Yes	Yes				
version			Yes		Global	No
version_comment			Yes		Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
version_compile_machine			Yes		Global	No
version_compile_os			Yes		Global	No
wait_timeout	Yes	Yes	Yes		Both	Yes
warning_count			Yes		Session	No
warnings	Yes	Yes				

[a] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

[b] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

5.1.2 Server Configuration Defaults

The MySQL server has many operating parameters, which you can change at server startup using command-line options or configuration files (option files). It is also possible to change many parameters at runtime. For general instructions on setting parameters at startup or runtime, see Section 5.1.3, “Server Command Options”, and Section 5.1.4, “Server System Variables”.

MySQL provides a number of preconfigured option files that can be used as a basis for tuning the MySQL server. Look for files named `my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, and `my-huge.cnf`, which are sample option files for small, medium, large, and very large systems. On Windows, the extension is `.ini` rather than `.cnf`.

Note

On Windows, the `.ini` or `.cnf` option file extension might not be displayed.

For a binary distribution, look for the sample files in or under your installation directory. If you have a source distribution, look in the `support-files` directory. To use a sample file as a base configuration file, rename a copy of it and place the copy in the appropriate location. Regarding names and appropriate location, see the general information provided in Section 4.2.6, “Using Option Files”. That section also describes option file format and syntax.

5.1.3 Server Command Options

When you start the `mysqld` server, you can specify program options using any of the methods described in Section 4.2.3, “Specifying Program Options”. The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See Section 4.2.6, “Using Option Files”. That section also describes option file format and syntax.

`mysqld` reads options from the `[mysqld]` and `[server]` groups. `mysqld_safe` reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. `mysql.server` reads options from the `[mysqld]` and `[mysql.server]` groups.

An embedded MySQL server usually reads options from the `[server]`, `[embedded]`, and `[`*`xxxxx`*`_SERVER]` groups, where *`xxxxx`* is the name of the application into which the server is embedded.

`mysqld` accepts many command options. For a brief summary, execute `mysqld --help`. To see the full list, use `mysqld --verbose --help`.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See Section 6.1.4, “Security-Related mysqld Options and Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- SSL-related options: See Section 6.3.6.5, “Command Options for Secure Connections”.
- Binary log control options: See Section 16.1.2.4, “Binary Log Options and Variables”.
- Replication-related options: See Section 16.1.2, “Replication and Binary Logging Options and Variables”.
- Options specific to particular storage engines: See Section 14.1.1, “MyISAM Startup Options”, Section 14.5.3, “BDB Startup Options”, Section 14.2.2, “InnoDB Startup Options and System Variables”, and mysqld Command Options for MySQL Cluster.

Some options control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to an option that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to an option for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some options take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued option is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

You can also set the values of server system variables at server startup by using variable names as options. To assign a value to a server system variable, use an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` sets the `key_buffer_size` variable to a value of 32MB.

When you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest permissible value if only certain values are permitted.

If you want to restrict the maximum value to which a variable can be set at runtime with `SET`, you can define this by using the `--maximum-var_name=value` command-line option.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated*.

You can change the values of most system variables for a running server with the `SET` statement. See Section 13.7.4, “SET Syntax”.

Section 5.1.4, “Server System Variables”, provides a full description for all variables, and additional information for setting them at server startup and runtime. Section 8.12.2, “Tuning Server Parameters”, includes information on optimizing the server by tuning system variables.

- `--help`, `-?`

Command-Line Format	`--help`

  Display a short help message and exit. Use both the `--verbose` and `--help` options to see the full message.

- `--allow-suspicious-udfs`

Introduced	5.0.3

This documentation is for an older version. If you're

Command-Line Format	--allow-suspicious-udfs	
Permitted Values	Type	boolean
	Default	FALSE

This option controls whether user-defined functions that have only an xxx symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. This option was added in version 5.0.3. See Section 21.2.2.6, “UDF Security Precautions”.

- --ansi

Command-Line Format	--ansi

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the --sql-mode option instead. See Section 1.8, “MySQL Standards Compliance”, and Section 5.1.7, “Server SQL Modes”.

- --basedir=*dir_name*, -b *dir_name*

Command-Line Format	--basedir=dir_name	
System Variable	Name	basedir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name

The path to the MySQL installation directory. All paths are usually resolved relative to this directory.

- --big-tables

Command-Line Format	--big-tables	
System Variable	Name	big_tables
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	OFF

Enable large result sets by saving all temporary sets in files. This option prevents most “table full” errors, but also slows down queries for which in-memory tables would suffice. Since MySQL 3.23.2, the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk tables where necessary.

- --bind-address=*addr*

Command-Line Format	--bind-address=addr	
Permitted Values	Type	string

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	`0.0.0.0`

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. The default address is `0.0.0.0`. To specify an address explicitly, use the `--bind-address=addr` option at server startup, where *`addr`* is an IPv4 address or a host name. If *`addr`* is a host name, the server resolves the name to an IPv4 address and binds to that address.

The server treats different types of addresses as follows:

- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces.
- If the address is a "regular" IPv4 address (such as `127.0.0.1`), the server accepts TCP/IP connections only for that particular IPv4 address.

If you intend to bind the server to a specific address, be sure that the `mysql.user` grant table contains an account with administrative privileges that you can use connect to that address. Otherwise, you will not be able to shut down the server. For example, if you bind to `0.0.0.0`, you can connect to the server using all existing accounts. But if you bind to `127.0.0.1`, the server accepts connections only on that address. In this case, first make sure that the `'root'@'127.0.0.1'` account is present in the `mysql.user` table so that you can still connect to the server to shut it down.

- `--bootstrap`

Command-Line Format	`--bootstrap`

This option is used by the `mysql_install_db` program to create the MySQL privilege tables without having to start a full MySQL server.

This option is unavailable if MySQL was configured with the `--disable-grant-options` option. See Section 2.17.3, "MySQL Source-Configuration Options".

- `--character-sets-dir=dir_name`

Command-Line Format		`--character-sets-dir=dir_name`
System Variable	**Name**	`character_sets_dir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

The directory where character sets are installed. See Section 10.5, "Character Set Configuration".

- `--character-set-client-handshake`

Command-Line Format		`--character-set-client-handshake`
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

Do not ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake`; this makes MySQL behave like MySQL 4.0.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--character-set-filesystem=charset_name`

Introduced	5.0.19	
Command-Line Format	`--character-set-filesystem=name`	
System Variable	Name	`character_set_filesystem`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`string`
	Default	`binary`

The file system character set. This option sets the `character_set_filesystem` system variable. It was added in MySQL 5.0.19.

- `--character-set-server=charset_name`, `-C charset_name`

Command-Line Format	`--character-set-server`	
System Variable	Name	`character_set_server`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`string`
	Default	`latin1`

Use *`charset_name`* as the default server character set. See Section 10.5, “Character Set Configuration”. If you use this option to specify a nondefault character set, you should also use `--collation-server` to specify the collation.

- `--chroot=dir_name`, `-r dir_name`

Command-Line Format	`--chroot=dir_name`	
Permitted Values	Type	`directory name`

Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure. Use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

- `--collation-server=collation_name`

Command-Line Format	`--collation-server`	
System Variable	Name	`collation_server`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`string`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	`latin1_swedish_ci`

Use *`collation_name`* as the default server collation. See Section 10.5, “Character Set Configuration”.

- `--console`

Command-Line Format	`--console`
Platform Specific	Windows

(Windows only.) Write error log messages to `stderr` and `stdout`. `mysqld` does not close the console window if this option is used.

If both `--log-error` and `--console` are specified, whichever option is given last takes precedence.

- `--core-file`

Command-Line Format	`--core-file`	
Permitted Values	**Type**	`boolean`
	Default	`OFF`

Write a core file if `mysqld` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.`*`pid`* is written to the current working directory of the process, which for `mysqld` is the data directory. *`pid`* represents the process ID of the server process. On OS X, a core file named `core.`*`pid`* is written to the `/cores` directory. On Solaris, use the `coreadm` command to specify where to write the core file and how to name it.

For some systems, to get a core file you must also specify the `--core-file-size` option to `mysqld_safe`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. On some systems, such as Solaris, you do not get a core file if you are also using the `--user` option. There might be additional restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation.

- `--datadir=`*`dir_name`*, `-h` *`dir_name`*

Command-Line Format	`--datadir=dir_name`	
System Variable	**Name**	`datadir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

The path to the data directory.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

Command-Line Format	`--debug[=debug_options]`	
System Variable	**Name**	`debug`
	Variable Scope	Global, Session

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values (Unix)	**Type**	`string`
	Default	`d:t:i:o,/tmp/mysqld.trace`
Permitted Values (Windows)	**Type**	`string`
	Default	`d:t:i:O,\mysqld.trace`

If MySQL is configured with `--with-debug`, you can use this option to get a trace file of what `mysqld` is doing. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows.

As of MySQL 5.0.25, using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

For more information, see Section 21.3.3, “The DBUG Package”.

- `--default-character-set=`*`charset_name`*

Deprecated	5.0.0	
Command-Line Format	`--default-character-set=name`	
Permitted Values	**Type**	`string`

Use *`charset_name`* as the default character set. This option is deprecated in favor of `--character-set-server`. See Section 10.5, “Character Set Configuration”. `--default-character-set` is removed in MySQL 5.5.

- `--default-collation=`*`collation_name`*

Deprecated	4.1.3	
Command-Line Format	`--default-collation=name`	
Permitted Values	**Type**	`string`

Use *`collation_name`* as the default collation. This option is deprecated in favor of `--collation-server`. See Section 10.5, “Character Set Configuration”. `--default-collation` is removed in MySQL 5.5.

- `--default-storage-engine=`*`type`*

Command-Line Format	`--default-storage-engine=name`	
System Variable	**Name**	`storage_engine`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`enumeration`
	Default	`MyISAM`

Set the default storage engine (table type) for tables. See Chapter 14, *Storage Engines*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--default-table-type=type`

Deprecated	5.0.0, by default-storage-engine	
Command-Line Format	`--default-table-type=name`	
Permitted Values	Type	`string`

This option is a deprecated synonym for `--default-storage-engine`.

- `--default-time-zone=timezone`

Command-Line Format	`--default-time-zone=name`	
Permitted Values	Type	`string`

Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-file=file_name`

Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is the full path name to the file.

- `--defaults-group-suffix=str`

Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqld` normally reads the `[mysqld]` group. If the `--defaults-group-suffix=_other` option is given, `mysqld` also reads the `[mysqld_other]` group. This option was added in MySQL 5.0.10.

- `--delay-key-write[={OFF|ON|ALL}]`

Command-Line Format	`--delay-key-write[=name]`	
System Variable	Name	`delay_key_write`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`enumeration`
	Default	`ON`
	Valid Values	`ON`
		`OFF`
		`ALL`

Specify how to use delayed key writes. Delayed key writing causes key buffers not to be flushed between writes for `MyISAM` tables. `OFF` disables delayed key writes. `ON` enables delayed key writes for

This documentation is for an older version. If you're
This documentation is for an older version. If you're

those tables that were created with the `DELAY_KEY_WRITE` option. `ALL` delays key writes for all `MyISAM` tables. See Section 8.12.2, “Tuning Server Parameters”, and Section 14.1.1, “MyISAM Startup Options”.

Note

If you set this variable to `ALL`, you should not use `MyISAM` tables from within another program (such as another MySQL server or `myisamchk`) when the tables are in use. Doing so leads to index corruption.

- `--des-key-file=file_name`

Command-Line Format	`--des-key-file=file_name`

Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` and `DES_DECRYPT()` functions.

- `--enable-locking`

This option is deprecated. Use `--external-locking` instead.

- `--enable-named-pipe`

Command-Line Format	`--enable-named-pipe`
Platform Specific	Windows

Enable support for named pipes. This option can be used only with the `mysqld-nt` and `mysqld-debug` servers that support named-pipe connections.

- `--enable-pstack`

Command-Line Format	`--enable-pstack`	
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Print a symbolic stack trace on failure. This capability is available only on Intel Linux systems, and only if MySQL was configured with the `--with-pstack` option.

- `--engine-condition-pushdown={ON|OFF}`

Introduced	5.0.3	
Command-Line Format	`--engine-condition-pushdown`	
System Variable	**Name**	`engine_condition_pushdown`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (>= 5.0.3)	**Type**	`boolean`
	Default	`OFF`

Sets the `engine_condition_pushdown` system variable. For more information, see Section 8.2.1.5, “Engine Condition Pushdown Optimization”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This variable was added in MySQL 5.0.3.

- `--exit-info[=flags]`, `-T [flags]`

Command-Line Format	`--exit-info[=flags]`	
Permitted Values	Type	integer

This is a bit mask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know *exactly* what it does!

- `--external-locking`

Command-Line Format	`--external-locking`	
Permitted Values	Type	boolean
	Default	FALSE

Enable external locking (system locking), which is disabled by default. If you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock. This option previously was named `--enable-locking`.

To disable external locking explicitly, use `--skip-external-locking`.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.11.4, “External Locking”.

- `--flush`

Command-Line Format	`--flush`	
System Variable	Name	flush
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	OFF

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”.

- `--gdb`

Command-Line Format	`--gdb`	
Permitted Values	Type	boolean
	Default	FALSE

Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See Section 21.3, “Debugging and Porting MySQL”.

Command-Line Format	--init-file=file_name	
System Variable	Name	init_file
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	file name

Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

This option is unavailable if MySQL was configured with the `--disable-grant-options` option. See Section 2.17.3, “MySQL Source-Configuration Options”.

- `--innodb-safe-binlog`

Deprecated	5.0.3	
Removed	5.0.3	
Command-Line Format	--innodb-safe-binlog	
Permitted Values	Type	boolean

If this option is given, then after a crash recovery by `InnoDB`, `mysqld` truncates the binary log after the last not-rolled-back transaction in the log. The option also causes `InnoDB` to print an error if the binary log is smaller or shorter than it should be. See Section 5.4.3, “The Binary Log”. This option was removed in MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support.

- `--innodb-`*`xxx`*

Set an option for the `InnoDB` storage engine. The `InnoDB` options are listed in Section 14.2.2, “InnoDB Startup Options and System Variables”.

- `--install [`*`service_name`*`]`

Command-Line Format	--install [service_name]
Platform Specific	Windows

(Windows only) Install the server as a Windows service that starts automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.10.4.7, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install` options, `--install` must be first.

- `--install-manual [`*`service_name`*`]`

Command-Line Format	--install-manual [service_name]
Platform Specific	Windows

This documentation is for an older version. If you're

(Windows only) Install the server as a Windows service that must be started manually. It does not start automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.10.4.7, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install-manual` options, `--install-manual` must be first.

- `--language=`*`lang_name`*`, -L` *`lang_name`*

Command-Line Format	`--language=name`	
System Variable	**Name**	`language`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`
	Default	`/usr/local/mysql/share/mysql/english/`

The language to use for error messages. *`lang_name`* can be given as the language name or as the full path name to the directory where the language files are installed. See Section 10.2, “Setting the Error Message Language”.

- `--large-pages`

Introduced	5.0.3	
Command-Line Format	`--large-pages`	
System Variable	**Name**	`large_pages`
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Linux	
Permitted Values (Linux)	**Type**	`boolean`
	Default	`FALSE`

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

MySQL supports only the Linux implementation of large page support (which is called HugeTLB in Linux). See Section 8.12.5.2, “Enabling Large Page Support”.

`--large-pages` is disabled by default. It was added in MySQL 5.0.3.

- `--local-service`

Command-Line Format	`--local-service`

This documentation is for an older version. If you're

(Windows only) A `--local-service` option following the service name causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order. See Section 2.10.4.7, “Starting MySQL as a Windows Service”.

- `--log[=file_name]`, `-l [file_name]`

Command-Line Format	`--log[=file_name]`	
System Variable	Name	`log`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`file name`

Log connections and SQL statements received from clients to this file. See Section 5.4.2, “The General Query Log”. If you omit the file name, MySQL uses *`host_name`*`.log` as the file name.

- `--log-error[=file_name]`

Command-Line Format	`--log-error[=file_name]`	
System Variable	Name	`log_error`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`file name`

Log errors and startup messages to this file. See Section 5.4.1, “The Error Log”. If you omit the file name, MySQL uses *`host_name`*`.err`. If the file name has no extension, the server adds an extension of `.err`.

- `--log-isam[=file_name]`

Command-Line Format	`--log-isam[=file_name]`	
Permitted Values	Type	`file name`

Log all `MyISAM` changes to this file (used only when debugging `MyISAM`).

- `--log-long-format`

Deprecated	4.1.0
Command-Line Format	`--log-long-format`

Log extra information to the update log, binary update log, and slow query log, if they have been activated. For example, the user name and timestamp are logged for all queries. This option is deprecated, as it now represents the default logging behavior. (See the description for `--log-short-format`.) The `--log-queries-not-using-indexes` option is available for the purpose of logging queries that do not use indexes to the slow query log. `--log-long-format` is removed in MySQL 5.5.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--log-queries-not-using-indexes`

Command-Line Format	`--log-queries-not-using-indexes`	
System Variable (>= 5.0.23)	Name	`log_queries_not_using_indexes`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`boolean`
	Default	`OFF`

If you are using this option with the slow query log enabled, queries that are expected to retrieve all rows are logged. See Section 5.4.4, “The Slow Query Log”. This option does not necessarily mean that no index is used. For example, a query that uses a full index scan uses an index but would be logged because the index would not limit the number of rows.

- `--log-short-format`

Command-Line Format	`--log-short-format`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

Originally intended to log less information to the update log and slow query log, if they have been activated. However, this option is not operational.

- `--log-slow-admin-statements`

Command-Line Format	`--log-slow-admin-statements`	
Permitted Values	Type	`boolean`
	Default	`OFF`

Include slow administrative statements in the statements written to the slow query log. Administrative statements include `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`.

- `--log-slow-queries[=file_name]`

Command-Line Format	`--log-slow-queries[=name]`	
System Variable	Name	`log_slow_queries`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`boolean`

Log all queries that have taken more than `long_query_time` seconds to execute to this file. See Section 5.4.4, “The Slow Query Log”. See the descriptions of the `--log-long-format` and `--log-short-format` options for details.

- `--log-tc=file_name`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Introduced	5.0.3	
Command-Line Format	`--log-tc=file_name`	
Permitted Values	Type	`file name`
	Default	`tc.log`

The name of the memory-mapped transaction coordinator log file (for XA transactions that affect multiple storage engines when the binary log is disabled). The default name is `tc.log`. The file is created under the data directory if not given as a full path name. This option is unused. Added in MySQL 5.0.3.

- `--log-tc-size=size`

Introduced	5.0.3	
Command-Line Format	`--log-tc-size=#`	
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`24576`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	`integer`
	Default	`24576`
	Max Value	`18446744073709547520`

The size in bytes of the memory-mapped transaction coordinator log. The default size is 24KB. Added in MySQL 5.0.3.

- `--log-warnings[=level]`, `-W [level]`

Command-Line Format	`--log-warnings[=#]`	
System Variable	Name	`log_warnings`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`1`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	`integer`
	Default	`1`
	Min Value	`0`
	Max Value	`18446744073709547520`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Print out warnings such as `Aborted connection...` to the error log. This option is enabled (1) by default. To disable it, use `--log-warnings=0`. Specifying the option without a *`level`* value increments the current value by 1. Enabling this option by setting it greater than 0 is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). If the value is greater than 1, aborted connections are written to the error log. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

If a slave server was started with `--log-warnings` enabled, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth.

- `--low-priority-updates`

Command-Line Format	`--low-priority-updates`	
System Variable	**Name**	`low_priority_updates`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Give table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) lower priority than selects. This can also be done using `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`). See Section 8.11.2, “Table Locking Issues”.

- `--memlock`

Command-Line Format	`--memlock`	
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Lock the `mysqld` process in memory. This option might help if you have a problem where the operating system is causing `mysqld` to swap to disk.

`--memlock` works on systems that support the `mlockall()` system call; this includes Solaris, most Linux distributions that use a 2.4 or newer kernel, and perhaps other Unix systems. On Linux systems, you can tell whether or not `mlockall()` (and thus this option) is supported by checking to see whether or not it is defined in the system `mman.h` file, like this:

```
shell> grep mlockall /usr/include/sys/mman.h
```

If `mlockall()` is supported, you should see in the output of the previous command something like the following:

```
extern int mlockall (int __flags) __THROW;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Important

Use of this option may require you to run the server as `root`, which, for reasons of security, is normally not a good idea. See Section 6.1.5, “How to Run MySQL as a Normal User”.

On Linux and perhaps other systems, you can avoid the need to run the server as `root` by changing the `limits.conf` file. See the notes regarding the memlock limit in Section 8.12.5.2, “Enabling Large Page Support”.

You must not try to use this option on a system that does not support the `mlockall()` system call; if you do so, `mysqld` will very likely crash as soon as you try to start it.

- `--myisam-block-size=N`

Command-Line Format	`--myisam-block-size=#`	
Permitted Values	Type	`integer`
	Default	`1024`
	Min Value	`1024`
	Max Value	`16384`

The block size to be used for `MyISAM` index pages.

- `--myisam-recover[=option[,option]...]]`

Command-Line Format	`--myisam-recover[=name]`	
Permitted Values	Type	`enumeration`
	Default	`OFF`
	Valid Values	`OFF`
		`DEFAULT`
		`BACKUP`
		`FORCE`
		`QUICK`

Set the `MyISAM` storage engine recovery mode. The option value is any combination of the values of `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`. If you specify multiple values, separate them by commas. Specifying the option with no argument is the same as specifying `DEFAULT`, and specifying with an explicit value of `""` disables recovery (same as not giving the option). If recovery is enabled, each time `mysqld` opens a `MyISAM` table, it checks whether the table is marked as crashed or was not closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, `mysqld` runs a check on the table. If the table was corrupted, `mysqld` attempts to repair it.

The following options affect how the repair works.

Option	Description
`DEFAULT`	Recovery without backup, forcing, or quick checking.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option	Description
BACKUP	If the data file was changed during recovery, save a backup of the *tbl_name*.MYD file as *tbl_name-datetime*.BAK.
FORCE	Run recovery even if we would lose more than one row from the .MYD file.
QUICK	Do not check the rows in the table if there are not any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options BACKUP,FORCE. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

See Section 14.1.1, “MyISAM Startup Options”.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

- --old-style-user-limits

Introduced	5.0.3	
Command-Line Format	--old-style-user-limits	
Permitted Values	Type	boolean
	Default	FALSE

  Enable old-style user limits. (Before MySQL 5.0.3, account resource limits were counted separately for each host from which a user connected rather than per account row in the user table.) See Section 6.3.4, “Setting Account Resource Limits”. This option was added in MySQL 5.0.3.

- --one-thread

Command-Line Format	--one-thread

  Only use one thread (for debugging under Linux). This option is available only if the server is built with debugging enabled. See Section 21.3, “Debugging and Porting MySQL”.

- --open-files-limit=*count*

Command-Line Format	--open-files-limit=#	
System Variable	Name	open_files_limit
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	integer
	Default	0
	Min Value	0
	Max Value	platform dependent

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Changes the number of file descriptors available to `mysqld`. You should try increasing the value of this option if `mysqld` gives you the error `Too many open files`. `mysqld` uses the option value to reserve descriptors with `setrlimit()`. Internally, the maximum value for this option is the maximum unsigned integer value, but the actual maximum is platform dependent. If the requested number of file descriptors cannot be allocated, `mysqld` writes a warning to the error log.

`mysqld` may attempt to allocate more than the requested number of descriptors (if they are available), using the values of `max_connections` and `table_cache` to estimate whether more descriptors will be needed.

On Unix, the value cannot be set less than `ulimit -n`.

- `--pid-file=file_name`

Command-Line Format	`--pid-file=file_name`	
System Variable	**Name**	`pid_file`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

The path name of the process ID file. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. This file is used by other programs such as `mysqld_safe` to determine the server's process ID.

- `--port=port_num`, `-P port_num`

Command-Line Format	`--port=#`	
System Variable	**Name**	`port`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`3306`
	Min Value	`0`
	Max Value	`65535`

The port number to use when listening for TCP/IP connections. On Unix and Unix-like systems, the port number must be 1024 or higher unless the server is started by the `root` system user.

- `--port-open-timeout=num`

Introduced	5.0.19	
Command-Line Format	`--port-open-timeout=#`	
Permitted Values	**Type**	`integer`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	`0`

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait. This option was added in MySQL 5.0.19.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--remove [service_name]`

Command-Line Format	`--remove [service_name]`
Platform Specific	Windows

(Windows only) Remove a MySQL Windows service. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.10.4.7, “Starting MySQL as a Windows Service”.

- `--safe-mode`

Command-Line Format	`--safe-mode`

Skip some optimization stages. This option is deprecated and is removed in MySQL 5.6.

- `--safe-show-database`

Deprecated	4.0.2	
Command-Line Format	`--safe-show-database`	
Permitted Values	**Type**	`boolean`

This option is deprecated and does not do anything because there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See Section 6.2.1, “Privileges Provided by MySQL”. `--safe-show-database` is removed in MySQL 5.5.

- `--safe-user-create`

Command-Line Format	`--safe-user-create`	
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` table or any column in the table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command-Line Format	--secure-auth	
System Variable	Name	secure_auth
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	OFF

This option causes the server to block connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format. Use it to prevent all use of passwords employing the old format (and hence insecure communication over the network).

Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format. See Section B.5.2.4, “Client does not support authentication protocol”.

The mysql client also has a --secure-auth option, which prevents connections to a server if the server requires a password in old format for the client account.

- --secure-file-priv=*dir_name*

Introduced	5.0.38	
Command-Line Format	--secure-file-priv=dir_name	
System Variable	Name	secure_file_priv
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string
	Default	empty
	Valid Values	empty
		dirname

This option limits the effect of the LOAD DATA and SELECT ... INTO OUTFILE statements and the LOAD_FILE() function to work only with files in the specified directory.

This option was added in MySQL 5.0.38.

- --shared-memory

Command-Line Format	--shared_memory[={0,1}]	
System Variable	Name	shared_memory
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Windows	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Enable shared-memory connections by local clients. This option is available only on Windows.

- `--shared-memory-base-name=`*`name`*

Command-Line Format	`--shared_memory_base_name=name`	
System Variable	**Name**	`shared_memory_base_name`
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Windows	
Permitted Values	**Type**	`string`
	Default	`MYSQL`

The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive.

- `--skip-bdb`

Disable the `BDB` storage engine. This saves memory and might speed up some operations. Do not use this option if you require `BDB` tables.

- `--skip-concurrent-insert`

Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.) See Section 8.11.3, “Concurrent Inserts”.

- `--skip-grant-tables`

This option causes the server to start without using the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement after connecting to the server. This option also suppresses loading of user-defined functions (UDFs).

`FLUSH PRIVILEGES` might be executed implicitly by other actions performed after startup. For example, `mysql_upgrade` flushes the privileges during the upgrade procedure.

This option is unavailable if MySQL was configured with the `--disable-grant-options` option. See Section 2.17.3, “MySQL Source-Configuration Options”.

- `--skip-host-cache`

Disable use of the internal host cache for faster name-to-IP resolution. In this case, the server performs a DNS lookup every time a client connects. See Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”.

- `--skip-innodb`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Disable the `InnoDB` storage engine. In this case, the server will not start if the default storage engine is set to `InnoDB`. Use `--default-storage-engine` to set the default to some other engine if necessary.

- `--skip-merge`

  Disable the `MERGE` storage engine. This option was added in MySQL 5.0.24. It can be used if the following behavior is undesirable: If a user has access to `MyISAM` table *`t`*, that user can create a `MERGE` table *`m`* that accesses *`t`*. However, if the user's privileges on *`t`* are subsequently revoked, the user can continue to access *`t`* by doing so through *`m`*.

- `--skip-name-resolve`

  Do not resolve host names when checking client connections. Use only IP addresses. If you use this option, all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”.

  Depending on the network configuration of your system and the `Host` values for your accounts, clients may need to connect using an explicit `--host` option, such as `--host=localhost` or `--host=127.0.0.1`.

- `--skip-networking`

  Do not listen for TCP/IP connections at all. All interaction with `mysqld` must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. See Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”.

- `--ssl*`

  Options that begin with `--ssl` specify whether to permit clients to connect using SSL and indicate where to find SSL keys and certificates. See Section 6.3.6.5, “Command Options for Secure Connections”.

- `--standalone`

Command-Line Format	`--standalone`
Platform Specific	Windows

  Instructs the MySQL server not to run as a service.

- `--symbolic-links`, `--skip-symbolic-links`

Command-Line Format	`--symbolic-links`

  Enable or disable symbolic link support. This option has different effects on Windows and Unix:

  - On Windows, enabling symbolic links enables you to establish a symbolic link to a database directory by creating a *`db_name`*`.sym` file that contains the path to the real directory. See Section 8.12.4.3, “Using Symbolic Links for Databases on Windows”.

  - On Unix, enabling symbolic links means that you can link a `MyISAM` index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` options of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”.

- `--skip-safemalloc`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command-Line Format	--skip-safemalloc

If MySQL is configured with `--with-debug=full`, all MySQL programs check for memory overruns during each memory allocation and memory freeing operation. This checking is very slow, so for the server you can avoid it when you do not need it by using the `--skip-safemalloc` option.

- `--skip-show-database`

Command-Line Format	--skip-show-database	
System Variable	Name	skip_show_database
	Variable Scope	Global
	Dynamic Variable	No

This option sets the `skip_show_database` system variable that controls who is permitted to use the `SHOW DATABASES` statement. See Section 5.1.4, “Server System Variables”.

- `--skip-stack-trace`

Command-Line Format	--skip-stack-trace

Do not write stack traces. This option is useful when you are running `mysqld` under a debugger. On some systems, you also must use this option to get a core file. See Section 21.3, “Debugging and Porting MySQL”.

- `--skip-thread-priority`

Command-Line Format	--skip-thread-priority

Disable using thread priorities for faster response time.

`mysqld` makes a large number of invalid calls to thread scheduling routines on Linux. These calls do not affect performance noticeably but may be a source of “noise” for debugging tools. For example, they can overwhelm other information of more interest in kernel logs. To avoid these calls, start the server with the `--skip-thread-priority` option.

- `--socket=`*`path`*

Command-Line Format	--socket={file_name|pipe_name}	
System Variable	Name	socket
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string
	Default	/tmp/mysql.sock

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. If this option is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory. On Windows, the option specifies the pipe

This documentation is for an older version. If you're
This documentation is for an older version. If you're

name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case sensitive).

- `--sql-mode=value[,value[,value...]]`

Command-Line Format	`--sql-mode=name`	
System Variable	Name	`sql_mode`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`set`
	Default	`''`
	Valid Values	`ALLOW_INVALID_DATES`
		`ANSI_QUOTES`
		`ERROR_FOR_DIVISION_BY_ZERO`
		`HIGH_NOT_PRECEDENCE`
		`IGNORE_SPACE`
		`NO_AUTO_CREATE_USER`
		`NO_AUTO_VALUE_ON_ZERO`
		`NO_BACKSLASH_ESCAPES`
		`NO_DIR_IN_CREATE`
		`NO_ENGINE_SUBSTITUTION`
		`NO_FIELD_OPTIONS`
		`NO_KEY_OPTIONS`
		`NO_TABLE_OPTIONS`
		`NO_UNSIGNED_SUBTRACTION`
		`NO_ZERO_DATE`
		`NO_ZERO_IN_DATE`
		`ONLY_FULL_GROUP_BY`
		`PAD_CHAR_TO_FULL_LENGTH`
		`PIPES_AS_CONCAT`
		`REAL_AS_FLOAT`
		`STRICT_ALL_TABLES`
		`STRICT_TRANS_TABLES`

Set the SQL mode. See Section 5.1.7, “Server SQL Modes”.

Note

MySQL installation programs may configure the SQL mode during the installation process. If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--sysdate-is-now`

Introduced	5.0.20	
Command-Line Format	`--sysdate-is-now`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

As of MySQL 5.0.12, `SYSDATE()` by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of `NOW()`. This option causes `SYSDATE()` to be an alias for `NOW()`. For information about the implications for binary logging and replication, see the description for `SYSDATE()` in Section 12.7, “Date and Time Functions” and for `SET TIMESTAMP` in Section 5.1.4, “Server System Variables”.

This option was added in MySQL 5.0.20.

- `--tc-heuristic-recover={COMMIT|ROLLBACK}`

Introduced	5.0.3	
Command-Line Format	`--tc-heuristic-recover=name`	
Permitted Values	Type	`enumeration`
	Default	`COMMIT`
	Valid Values	`COMMIT`
		`ROLLBACK`

The type of decision to use in the heuristic recovery process. This option is unused. Added in MySQL 5.0.3.

- `--temp-pool`

Command-Line Format	`--temp-pool`	
Permitted Values	Type	`boolean`
	Default	`TRUE`

This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache.

- `--transaction-isolation=`*`level`*

Command-Line Format	`--transaction-isolation=name`	
Permitted Values	Type	`enumeration`
	Default	`REPEATABLE-READ`
	Valid Values	`READ-UNCOMMITTED`
		`READ-COMMITTED`
		`REPEATABLE-READ`
		`SERIALIZABLE`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Sets the default transaction isolation level. The `level` value can be `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. See Section 13.3.6, “SET TRANSACTION Syntax”.

The default transaction isolation level can also be set at runtime using the `SET TRANSACTION` statement or by setting the `tx_isolation` system variable.

- `--tmpdir=dir_name`, `-t dir_name`

Command-Line Format	`--tmpdir=dir_name`	
System Variable	**Name**	`tmpdir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (“`:`”) on Unix and semicolon characters (“`;`”) on Windows, NetWare, and OS/2. If the MySQL server is acting as a replication slave, you should not set `--tmpdir` to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. For more information about the storage location of temporary files, see Section B.5.3.5, “Where MySQL Stores Temporary Files”. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- `--user={user_name|user_id}`, `-u {user_name|user_id}`

Command-Line Format	`--user=name`	
Permitted Values	**Type**	`string`

Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting `mysqld` as `root`. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See Section 6.1.1, “Security Guidelines”.

To avoid a possible security hole where a user adds a `--user=root` option to a `my.cnf` file (thus causing the server to run as `root`), `mysqld` uses only the first `--user` option specified and produces a warning if there are multiple `--user` options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` option is found.

- `--verbose`, `-v`

Use this option with the `--help` option for detailed help.

- `--version`, `-V`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variable	Name	version
	Variable Scope	Global
	Dynamic Variable	No

Display version information and exit.

An attempt to connect to the host 127.0.0.1 normally resolves to the localhost account. However, this fails if the server is run with the --skip-name-resolve option, so make sure that an account exists that can accept a connection. For example, to be able to connect as root using --host=127.0.0.1 or --host=::1, create these accounts:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@'::1' IDENTIFIED BY 'root-password';
```

5.1.4 Server System Variables

The MySQL server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the SET statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server will use based on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the SHOW VARIABLES statement.

This section provides a description of each system variable. Variables with no version indicated are present in all MySQL 5.0 releases.

The following table lists all available system variables.

Table 5.2 System Variable Summary

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
auto_increment_increment			Yes	Both	Yes
auto_increment_offset			Yes	Both	Yes
autocommit			Yes	Session	Yes
automatic_sp_privileges			Yes	Global	Yes
back_log			Yes	Global	No
basedir	Yes	Yes	Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
bdb_cache_size			Yes	Global	No
bdb-home	Yes	Yes			No
- *Variable*: bdb_home			Yes	Global	No
bdb-lock-detect	Yes	Yes	Yes	Global	No
bdb_log_buffer_size			Yes	Global	No
bdb-logdir	Yes	Yes			No
- *Variable*: bdb_logdir			Yes	Global	No
bdb_max_lock			Yes	Global	No
bdb-shared-data	Yes	Yes			No
- *Variable*: bdb_shared_data			Yes	Global	No
bdb-tmpdir	Yes	Yes			No
- *Variable*: bdb_tmpdir			Yes	Global	No
big-tables	Yes	Yes			Yes
- *Variable*: big_tables			Yes	Session	Yes
binlog_cache_size	Yes	Yes	Yes	Global	Yes
bulk_insert_buffer_size	Yes	Yes	Yes	Both	Yes
character_set_client			Yes	Both	Yes
character_set_connection			Yes	Both	Yes
character_set_database[a]			Yes	Both	Yes
character-set-filesystem	Yes	Yes			Yes
- *Variable*: character_set_filesystem			Yes	Both	Yes
character_set_results			Yes	Both	Yes
character-set-server	Yes	Yes			Yes
- *Variable*: character_set_server			Yes	Both	Yes
character_set_system			Yes	Global	No
character-sets-dir	Yes	Yes			No
- *Variable*: character_sets_dir			Yes	Global	No
collation_connection			Yes	Both	Yes
collation_database[b]			Yes	Both	Yes
collation-server	Yes	Yes			Yes
- *Variable*: collation_server			Yes	Both	Yes
completion_type	Yes	Yes	Yes	Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
concurrent_insert	Yes	Yes	Yes	Global	Yes
connect_timeout	Yes	Yes	Yes	Global	Yes
datadir	Yes	Yes	Yes	Global	No
date_format			Yes	Both	No
datetime_format			Yes	Both	No
debug	Yes	Yes	Yes	Both	Yes
default-storage-engine	Yes	Yes	Yes	Both	Yes
default_week_format	Yes	Yes	Yes	Both	Yes
delay-key-write	Yes	Yes			Yes
- *Variable*: delay_key_write			Yes	Global	Yes
delayed_insert_limit	Yes	Yes	Yes	Global	Yes
delayed_insert_timeout	Yes	Yes	Yes	Global	Yes
delayed_queue_size	Yes	Yes	Yes	Global	Yes
div_precision_increment	Yes	Yes	Yes	Both	Yes
engine-condition-pushdown	Yes	Yes			Yes
- *Variable*: engine_condition_pushdown			Yes	Both	Yes
error_count			Yes	Session	No
expire_logs_days	Yes	Yes	Yes	Global	Yes
flush	Yes	Yes	Yes	Global	Yes
flush_time	Yes	Yes	Yes	Global	Yes
foreign_key_checks			Yes	Session	Yes
ft_boolean_syntax	Yes	Yes	Yes	Global	Yes
ft_max_word_len	Yes	Yes	Yes	Global	No
ft_min_word_len	Yes	Yes	Yes	Global	No
ft_query_expansion_limit	Yes	Yes	Yes	Global	No
ft_stopword_file	Yes	Yes	Yes	Global	No
group_concat_max_len	Yes	Yes	Yes	Both	Yes
have_archive			Yes	Global	No
have_bdb			Yes	Global	No
have_blackhole_engine			Yes	Global	No
have_community_features			Yes	Global	No
have_compress			Yes	Global	No
have_crypt			Yes	Global	No
have_csv			Yes	Global	No
have_example_engine			Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
have_federated_engine			Yes	Global	No
have_geometry			Yes	Global	No
have_innodb			Yes	Global	No
have_isam			Yes	Global	No
have_merge_engine			Yes	Global	No
have_ndbcluster			Yes	Global	No
have_openssl			Yes	Global	No
have_profiling			Yes	Global	No
have_query_cache			Yes	Global	No
have_raid			Yes	Global	No
have_rtree_keys			Yes	Global	No
have_ssl			Yes	Global	No
have_symlink			Yes	Global	No
hostname			Yes	Global	No
identity			Yes	Session	Yes
init_connect	Yes	Yes	Yes	Global	Yes
init-file	Yes	Yes			No
- *Variable*: init_file			Yes	Global	No
init_slave	Yes	Yes	Yes	Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes	Global	No
innodb_additional_mem_pool_size	Yes	Yes	Yes	Global	No
innodb_autoextend_increment	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_awe_mem_mb	Yes	Yes	Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes	Global	No
innodb_checksums	Yes	Yes	Yes	Global	No
innodb_commit_concurrency	Yes	Yes	Yes	Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes	Global	Yes
innodb_data_file_path	Yes	Yes	Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes	Global	No
innodb_doublewrite	Yes	Yes	Yes	Global	No
innodb_fast_shutdown	Yes	Yes	Yes	Global	Yes
innodb_file_io_threads	Yes	Yes	Yes	Global	No
innodb_file_per_table	Yes	Yes	Yes	Global	No
innodb_flush_log_at_trx_commit	Yes	Yes	Yes	Global	Yes
innodb_flush_method	Yes	Yes	Yes	Global	No
innodb_force_recovery	Yes	Yes	Yes	Global	No
innodb_lock_wait_timeout	Yes	Yes	Yes	Global	No
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_log_arch_dir	Yes	Yes	Yes	Global	No
innodb_log_archive	Yes	Yes	Yes	Global	No
innodb_log_buffer_size	Yes	Yes	Yes	Global	No
innodb_log_file_size	Yes	Yes	Yes	Global	No
innodb_log_files_in_group	Yes	Yes	Yes	Global	No
innodb_log_group_home_dir	Yes	Yes	Yes	Global	No
innodb_max_dirty_pages_pct	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes	Global	Yes
innodb_mirrored_log_groups	Yes	Yes	Yes	Global	No
innodb_open_files	Yes	Yes	Yes	Global	No
innodb_rollback_on_timeout	Yes	Yes	Yes	Global	No
innodb_support_xa	Yes	Yes	Yes	Both	Yes
innodb_sync_spin_loops	Yes	Yes	Yes	Global	Yes
innodb_table_locks	Yes	Yes	Yes	Both	Yes
innodb_thread_concurrency	Yes	Yes	Yes	Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_use_legacy_cardinality_algorithm	Yes	Yes	Yes	Global	Yes
insert_id			Yes	Session	Yes
interactive_timeout	Yes	Yes	Yes	Both	Yes
join_buffer_size	Yes	Yes	Yes	Both	Yes
keep_files_on_create	Yes	Yes	Yes	Both	Yes
key_buffer_size	Yes	Yes	Yes	Global	Yes
key_cache_age_threshold	Yes	Yes	Yes	Global	Yes
key_cache_block_size	Yes	Yes	Yes	Global	Yes
key_cache_division_limit	Yes	Yes	Yes	Global	Yes
language	Yes	Yes	Yes	Global	No
large_files_support			Yes	Global	No
large_page_size			Yes	Global	No
large-pages	Yes	Yes			No
- *Variable*: large_pages			Yes	Global	No
last_insert_id			Yes	Session	Yes
lc_time_names			Yes	Both	Yes
license			Yes	Global	No
local_infile			Yes	Global	Yes
locked_in_memory			Yes	Global	No
log	Yes	Yes	Yes	Global	No
log-bin	Yes	Yes	Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
log_bin			Yes	Global	No
log-bin-trust-function-creators	Yes	Yes			Yes
- *Variable*: log_bin_trust_function_creators			Yes	Global	Yes
log-bin-trust-routine-creators	Yes	Yes			Yes
- *Variable*: log_bin_trust_routine_creators			Yes	Global	Yes
log-error	Yes	Yes			No
- *Variable*: log_error			Yes	Global	No
log-queries-not-using-indexes	Yes	Yes			Yes
- *Variable*: log_queries_not_using_indexes			Yes	Global	Yes
log-slave-updates	Yes	Yes			No
- *Variable*: log_slave_updates			Yes	Global	No
log_slave_updates	Yes	Yes	Yes	Global	No
log-slow-queries	Yes	Yes			No
- *Variable*: log_slow_queries			Yes	Global	No
log-warnings	Yes	Yes			Yes
- *Variable*: log_warnings			Yes	Both	Yes
long_query_time	Yes	Yes	Yes	Both	Yes
low-priority-updates	Yes	Yes			Yes
- *Variable*: low_priority_updates			Yes	Both	Yes
lower_case_file_system			Yes	Global	No
lower_case_table_names	Yes	Yes	Yes	Global	No
max_allowed_packet	Yes	Yes	Yes	Both	Yes
max_binlog_cache_size	Yes	Yes	Yes	Global	Yes
max_binlog_size	Yes	Yes	Yes	Global	Yes
max_connect_errors	Yes	Yes	Yes	Global	Yes
max_connections	Yes	Yes	Yes	Global	Yes
max_delayed_threads	Yes	Yes	Yes	Both	Yes
max_error_count	Yes	Yes	Yes	Both	Yes
max_heap_table_size	Yes	Yes	Yes	Both	Yes
max_insert_delayed_threads			Yes	Both	Yes
max_join_size	Yes	Yes	Yes	Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
max_length_for_sort_data	Yes	Yes	Yes	Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes	Global	Yes
max_relay_log_size	Yes	Yes	Yes	Global	Yes
max_seeks_for_key	Yes	Yes	Yes	Both	Yes
max_sort_length	Yes	Yes	Yes	Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes	Both	Yes
max_tmp_tables			Yes	Both	Yes
max_user_connections	Yes	Yes	Yes	Varies	Yes
max_write_lock_count	Yes	Yes	Yes	Global	Yes
multi_range_count	Yes	Yes	Yes	Both	Yes
myisam_data_pointer_size	Yes	Yes	Yes	Global	Yes
myisam_max_extra_sort_file_size	Yes	Yes	Yes	Global	No
myisam_max_sort_file_size	Yes	Yes	Yes	Global	Yes
myisam_mmap_size	Yes	Yes	Yes	Global	No
myisam_recover_options			Yes	Global	No
myisam_repair_threads	Yes	Yes	Yes	Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes	Both	Yes
myisam_stats_method	Yes	Yes	Yes	Both	Yes
named_pipe			Yes	Global	No
ndb_autoincrement_prefetch_sz	Yes	Yes	Yes	Both	Yes
ndb_cache_check_time	Yes	Yes	Yes	Global	Yes
ndb_force_send	Yes	Yes	Yes	Both	Yes
ndb_index_stat_cache_entries	Yes	Yes	Yes	Both	Yes
ndb_index_stat_enable	Yes	Yes	Yes	Both	Yes
ndb_index_stat_update_freq	Yes	Yes	Yes	Both	Yes
ndb_optimized_node_selection	Yes	Yes	Yes	Global	No
ndb_use_exact_count			Yes	Both	Yes
ndb_use_transactions	Yes	Yes	Yes	Both	Yes
net_buffer_length	Yes	Yes	Yes	Both	Yes
net_read_timeout	Yes	Yes	Yes	Both	Yes
net_retry_count	Yes	Yes	Yes	Both	Yes
net_write_timeout	Yes	Yes	Yes	Both	Yes
new	Yes	Yes	Yes	Both	Yes
old_passwords			Yes	Both	Yes
open-files-limit	Yes	Yes			No
- *Variable*: open_files_limit			Yes	Global	No
optimizer_prune_level	Yes	Yes	Yes	Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
optimizer_search_depth	Yes	Yes	Yes	Both	Yes
pid-file	Yes	Yes			No
- *Variable*: pid_file			Yes	Global	No
plugin_dir	Yes	Yes	Yes	Global	No
port	Yes	Yes	Yes	Global	No
preload_buffer_size	Yes	Yes	Yes	Both	Yes
prepared_stmt_count			Yes	Global	No
profiling			Yes	Session	Yes
profiling_history_size	Yes	Yes	Yes	Both	Yes
protocol_version			Yes	Global	No
pseudo_thread_id			Yes	Session	Yes
query_alloc_block_size	Yes	Yes	Yes	Both	Yes
query_cache_limit	Yes	Yes	Yes	Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes	Global	Yes
query_cache_size	Yes	Yes	Yes	Global	Yes
query_cache_type	Yes	Yes	Yes	Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes	Both	Yes
query_prealloc_size	Yes	Yes	Yes	Both	Yes
rand_seed1			Yes	Session	Yes
rand_seed2			Yes	Session	Yes
range_alloc_block_size	Yes	Yes	Yes	Both	Yes
read_buffer_size	Yes	Yes	Yes	Both	Yes
read_only	Yes	Yes	Yes	Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes	Both	Yes
relay-log	Yes	Yes			No
- *Variable*: relay_log			Yes	Global	No
relay-log-index	Yes	Yes			No
- *Variable*: relay_log_index			Yes	Global	No
relay_log_index	Yes	Yes	Yes	Global	No
relay_log_info_file	Yes	Yes	Yes	Global	No
relay_log_purge	Yes	Yes	Yes	Global	Yes
relay_log_space_limit	Yes	Yes	Yes	Global	No
rpl_recovery_rank			Yes	Global	Yes
secure-auth	Yes	Yes			Yes
- *Variable*: secure_auth			Yes	Global	Yes
secure-file-priv	Yes	Yes			No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
- *Variable*: secure_file_priv			Yes	Global	No
server-id	Yes	Yes			Yes
- *Variable*: server_id			Yes	Global	Yes
shared_memory	Yes	Yes	Yes	Global	No
shared_memory_base_name	Yes	Yes	Yes	Global	No
skip_external_locking	Yes	Yes	Yes	Global	No
skip-networking	Yes	Yes			No
- *Variable*: skip_networking			Yes	Global	No
skip-show-database	Yes	Yes			No
- *Variable*: skip_show_database			Yes	Global	No
skip-sync-bdb-logs	Yes	Yes	Yes	Global	No
slave_compressed_protocol	Yes	Yes	Yes	Global	Yes
slave-load-tmpdir	Yes	Yes			No
- *Variable*: slave_load_tmpdir			Yes	Global	No
slave-net-timeout	Yes	Yes			Yes
- *Variable*: slave_net_timeout			Yes	Global	Yes
slave-skip-errors	Yes	Yes			No
- *Variable*: slave_skip_errors			Yes	Global	No
slave_transaction_retries	Yes	Yes	Yes	Global	Yes
slow_launch_time	Yes	Yes	Yes	Global	Yes
socket	Yes	Yes	Yes	Global	No
sort_buffer_size	Yes	Yes	Yes	Both	Yes
sql_auto_is_null			Yes	Session	Yes
sql_big_selects			Yes	Session	Yes
sql_big_tables			Yes	Session	Yes
sql_buffer_result			Yes	Session	Yes
sql_log_bin			Yes	Session	Yes
sql_log_off			Yes	Session	Yes
sql_log_update			Yes	Session	Yes
sql_low_priority_updates			Yes	Both	Yes
sql_max_join_size			Yes	Both	Yes
sql-mode	Yes	Yes			Yes
- *Variable*: sql_mode			Yes	Both	Yes
sql_notes			Yes	Session	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
sql_quote_show_create			Yes	Session	Yes
sql_safe_updates			Yes	Session	Yes
sql_select_limit			Yes	Both	Yes
sql_slave_skip_counter			Yes	Global	Yes
sql_warnings			Yes	Session	Yes
ssl-ca	Yes	Yes			No
- *Variable*: ssl_ca			Yes	Global	No
ssl-capath	Yes	Yes			No
- *Variable*: ssl_capath			Yes	Global	No
ssl-cert	Yes	Yes			No
- *Variable*: ssl_cert			Yes	Global	No
ssl-cipher	Yes	Yes			No
- *Variable*: ssl_cipher			Yes	Global	No
ssl-key	Yes	Yes			No
- *Variable*: ssl_key			Yes	Global	No
storage_engine			Yes	Both	Yes
sync-bdb-logs	Yes	Yes	Yes	Global	No
sync_binlog	Yes	Yes	Yes	Global	Yes
sync_frm	Yes	Yes	Yes	Global	Yes
system_time_zone			Yes	Global	No
table_cache	Yes	Yes	Yes	Global	Yes
table_lock_wait_timeout	Yes	Yes	Yes	Global	Yes
table_type			Yes	Both	Yes
thread_cache_size	Yes	Yes	Yes	Global	Yes
thread_concurrency	Yes	Yes	Yes	Global	No
thread_stack	Yes	Yes	Yes	Global	No
time_format			Yes	Both	No
time_zone			Yes	Both	Yes
timed_mutexes	Yes	Yes	Yes	Global	Yes
timestamp			Yes	Session	Yes
tmp_table_size	Yes	Yes	Yes	Both	Yes
tmpdir	Yes	Yes	Yes	Global	No
transaction_alloc_block_size	Yes	Yes	Yes	Both	Yes
transaction_prealloc_size	Yes	Yes	Yes	Both	Yes
tx_isolation			Yes	Both	Yes
unique_checks			Yes	Session	Yes
updatable_views_with_limit	Yes	Yes	Yes	Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
version			Yes	Global	No
version_comment			Yes	Global	No
version_compile_machine			Yes	Global	No
version_compile_os			Yes	Global	No
wait_timeout	Yes	Yes	Yes	Both	Yes
warning_count			Yes	Session	No

[a] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.
[b] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

For additional system variable information, see these sections:

- Section 5.1.5, “Using System Variables”, discusses the syntax for setting and displaying system variable values.
- Section 5.1.5.2, “Dynamic System Variables”, lists the variables that can be set at runtime.
- Information on tuning system variables can be found in Section 8.12.2, “Tuning Server Parameters”.
- Section 14.2.2, “InnoDB Startup Options and System Variables”, lists `InnoDB` system variables.
- MySQL Cluster System Variables, lists system variables which are specific to MySQL Cluster.
- For information on server system variables specific to replication, see Section 16.1.2, “Replication and Binary Logging Options and Variables”.

Note

Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` works but `--delay_key_write=ON` does not.

Some system variables control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to a system variable that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to a variable for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some system variables take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued variable is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `autocommit`

System Variable	Name	`autocommit`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`ON`

The autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you must use `COMMIT` to accept a transaction or `ROLLBACK` to cancel it. If `autocommit` is 0 and you change it to 1, MySQL performs an automatic `COMMIT` of any open transaction. Another way to begin a transaction is to use a `START TRANSACTION` or `BEGIN` statement. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

By default, client connections begin with `autocommit` set to 1. To cause clients to begin with a default of 0, set the server's `init_connect` system variable:

```
SET GLOBAL init_connect='SET autocommit=0';
```

The `init_connect` variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]
init_connect='SET autocommit=0'
```

The content of `init_connect` is not executed for users that have the `SUPER` privilege.

- `automatic_sp_privileges`

Introduced	5.0.3	
System Variable	**Name**	`automatic_sp_privileges`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

When this variable has a value of 1 (the default), the server automatically grants the `EXECUTE` and `ALTER ROUTINE` privileges to the creator of a stored routine, if the user cannot already execute and alter or drop the routine. (The `ALTER ROUTINE` privilege is required to drop the routine.) The server also automatically drops those privileges from the creator when the routine is dropped. If `automatic_sp_privileges` is 0, the server does not automatically add or drop these privileges.

The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

See also Section 18.2.2, “Stored Routines and MySQL Privileges”.

This variable was added in MySQL 5.0.3.

- `back_log`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variable	Name	back_log
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	integer
	Default	50
	Min Value	1
	Max Value	65535

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The back_log value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix listen() system call should have more details. Check your OS documentation for the maximum value for this variable. back_log cannot be set higher than your operating system limit.

- basedir

Command-Line Format	--basedir=dir_name	
System Variable	Name	basedir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name

The MySQL installation base directory. This variable can be set with the --basedir option. Relative path names for other variables usually are resolved relative to the base directory.

- bdb_cache_size

System Variable	Name	bdb_cache_size
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	integer
	Min Value	20480

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The size of the buffer that is allocated for caching indexes and rows for BDB tables. If you do not use BDB tables, you should start mysqld with --skip-bdb to not allocate memory for this cache.

- bdb_home

Command-Line Format	--bdb-home=dir_name	
System Variable	Name	bdb_home
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name

The base directory for BDB tables. This should be assigned the same value as the datadir variable.

- bdb_log_buffer_size

System Variable	Name	bdb_log_buffer_size
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	integer
	Min Value	262144
	Max Value	4294967295

The size of the buffer that is allocated for caching indexes and rows for BDB tables. If you do not use BDB tables, you should set this to 0 or start mysqld with --skip-bdb to not allocate memory for this cache.

- bdb_logdir

Command-Line Format	--bdb-logdir=file_name	
System Variable	Name	bdb_logdir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name

The directory where the BDB storage engine writes its log files. This variable can be set with the --bdb-logdir option.

- bdb_max_lock

System Variable	Name	bdb_max_lock

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`10000`

The maximum number of locks that can be active for a `BDB` table (10,000 by default). You should increase this value if errors such as the following occur when you perform long transactions or when `mysqld` has to examine many rows to calculate a query:

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

For more information, see Section 21.3.3, “The DBUG Package”.

- `bdb_shared_data`

Command-Line Format	`--bdb-shared-data`	
System Variable	**Name**	`bdb_shared_data`
	Variable Scope	Global
	Dynamic Variable	No

This is `ON` if you are using `--bdb-shared-data` to start Berkeley DB in multi-process mode. (Do not use `DB_PRIVATE` when initializing Berkeley DB.)

- `bdb_tmpdir`

Command-Line Format	`--bdb-tmpdir=dir_name`	
System Variable	**Name**	`bdb_tmpdir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

The `BDB` temporary file directory.

- `big_tables`

If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error `The table tbl_name is full` does not occur for `SELECT` operations that require a large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). Normally, you should never need to set this variable, because in-memory tables are automatically converted to disk-based tables as required.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

This variable was formerly named `sql_big_tables`.

- `binlog_cache_size`

Command-Line Format		`--binlog_cache_size=#`
System Variable	**Name**	`binlog_cache_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`32768`
	Min Value	`4096`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`32768`
	Min Value	`4096`
	Max Value	`18446744073709547520`

The size of the cache to hold the SQL statements for the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin` option). If you often use large, multiple-statement transactions, you can increase this cache size to get better performance. The `Binlog_cache_use` and `Binlog_cache_disk_use` status variables can be useful for tuning the size of this variable. See Section 5.4.3, “The Binary Log”.

- `bulk_insert_buffer_size`

Command-Line Format		`--bulk_insert_buffer_size=#`
System Variable	**Name**	`bulk_insert_buffer_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`8388608`
	Min Value	`0`
	Max Value	`4294967295`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`8388608`
	Min Value	`0`
	Max Value	`18446744073709547520`

`MyISAM` uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA INFILE` when adding data to nonempty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

- `character_set_client`

System Variable	**Name**	`character_set_client`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`string`

The character set for statements that arrive from the client. The session value of this variable is set using the character set requested by the client when the client connects to the server. (Many clients support a `--default-character-set` option to enable this character set to be specified explicitly. See also Section 10.1.4, “Connection Character Sets and Collations”.) The global value of the variable is used to set the session value in cases when the client-requested value is unknown or not available, or the server is configured to ignore client requests:

- The client is from a version of MySQL older than MySQL 4.1, and thus does not request a character set.
- The client requests a character set not known to the server. For example, a Japanese-enabled client requests `sjis` when connecting to a server not configured with `sjis` support.
- `mysqld` was started with the `--skip-character-set-client-handshake` option, which causes it to ignore client character set configuration. This reproduces MySQL 4.0 behavior and is useful should you wish to upgrade the server without upgrading all the clients.

`ucs2` cannot be used as a client character set, which means that it also does not work for `SET NAMES` or `SET CHARACTER SET`.

- `character_set_connection`

System Variable	**Name**	`character_set_connection`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`string`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The character set used for literals that do not have a character set introducer and for number-to-string conversion.

- `character_set_database`

System Variable	Name	character_set_database
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.	
Permitted Values	Type	string

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server`.

- `character_set_filesystem`

Introduced	5.0.19	
Command-Line Format	--character-set-filesystem=name	
System Variable	Name	character_set_filesystem
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	string
	Default	binary

The file system character set. This variable is used to interpret string literals that refer to file names, such as in the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. Such file names are converted from `character_set_client` to `character_set_filesystem` before the file opening attempt occurs. The default value is `binary`, which means that no conversion occurs. For systems on which multibyte file names are permitted, a different value may be more appropriate. For example, if the system represents file names using UTF-8, set `character_set_filesystem` to `'utf8'`. This variable was added in MySQL 5.0.19.

- `character_set_results`

System Variable	Name	character_set_results
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	string

The character set used for returning query results such as result sets or error messages to the client.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `character_set_server`

Command-Line Format	`--character-set-server`	
System Variable	Name	`character_set_server`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`string`
	Default	`latin1`

The server's default character set.

- `character_set_system`

System Variable	Name	`character_set_system`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`string`
	Default	`utf8`

The character set used by the server for storing identifiers. The value is always `utf8`.

- `character_sets_dir`

Command-Line Format	`--character-sets-dir=dir_name`	
System Variable	Name	`character_sets_dir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`directory name`

The directory where character sets are installed.

- `collation_connection`

System Variable	Name	`collation_connection`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`string`

The collation of the connection character set.

- `collation_database`

This documentation is for an older version. If you're

System Variable	Name	collation_database
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.	
Permitted Values	Type	string

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as collation_server.

- collation_server

Command-Line Format	--collation-server	
System Variable	Name	collation_server
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	string
	Default	latin1_swedish_ci

The server's default collation.

- completion_type

Introduced	5.0.3	
Command-Line Format	--completion_type=#	
System Variable	Name	completion_type
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (>= 5.0.3)	Type	integer
	Default	0
	Valid Values	0
		1
		2

The transaction completion type. This variable can take the values shown in the following table.

Value	Description
0	COMMIT and ROLLBACK are unaffected. This is the default value.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Value	Description
1	`COMMIT` and `ROLLBACK` are equivalent to `COMMIT AND CHAIN` and `ROLLBACK AND CHAIN`, respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
2	`COMMIT` and `ROLLBACK` are equivalent to `COMMIT RELEASE` and `ROLLBACK RELEASE`, respectively. (The server disconnects after terminating the transaction.)

`completion_type` affects transactions that begin with `START TRANSACTION` or `BEGIN` and end with `COMMIT` or `ROLLBACK`. It does not apply to implicit commits resulting from execution of the statements listed in Section 13.3.3, “Statements That Cause an Implicit Commit”. It also does not apply for `XA COMMIT`, `XA ROLLBACK`, or when `autocommit=1`.

This variable was added in MySQL 5.0.3.

- `concurrent_insert`

Command-Line Format	`--concurrent_insert[=#]`	
System Variable	**Name**	`concurrent_insert`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (<= 5.0.5)	**Type**	`boolean`
	Default	`TRUE`
Permitted Values (>= 5.0.6)	**Type**	`integer`
	Default	`1`
	Valid Values	`0`
		`1`
		`2`

If 1 (the default), MySQL permits `INSERT` and `SELECT` statements to run concurrently for `MyISAM` tables that have no free blocks in the middle of the data file. If you start `mysqld` with `--skip-new`, this variable is set to 0.

In MySQL 5.0.6, this variable was changed to take three integer values:

Value	Description
0	Disables concurrent inserts
1	(Default) Enables concurrent insert for `MyISAM` tables that do not have holes
2	Enables concurrent inserts for all `MyISAM` tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.

See also Section 8.11.3, “Concurrent Inserts”.

- `connect_timeout`

Command-Line Format	`--connect_timeout=#`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variable	Name	connect_timeout
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (<= 5.0.51)	Type	integer
	Default	5
	Min Value	2
	Max Value	31536000
Permitted Values (>= 5.0.52)	Type	integer
	Default	10
	Min Value	2
	Max Value	31536000

The number of seconds that the `mysqld` server waits for a connect packet before responding with `Bad handshake`. The default value is 10 seconds as of MySQL 5.0.52 and 5 seconds before that.

Increasing the `connect_timeout` value might help if clients frequently encounter errors of the form `Lost connection to MySQL server at '`*`XXX`*`', system error: `*`errno`*.

- `datadir`

Command-Line Format	--datadir=dir_name	
System Variable	Name	datadir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name

The MySQL data directory. This variable can be set with the `--datadir` option.

- `date_format`

This variable is unused.

- `datetime_format`

This variable is unused.

- `default_week_format`

Command-Line Format	--default_week_format=#	
System Variable	Name	default_week_format

This documentation is for an older version. If you're

	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`7`

The default mode value to use for the `WEEK()` function. See Section 12.7, “Date and Time Functions”.

- `delay_key_write`

Command-Line Format	`--delay-key-write[=name]`	
System Variable	**Name**	`delay_key_write`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`enumeration`
	Default	`ON`
	Valid Values	`ON`
		`OFF`
		`ALL`

This option applies only to `MyISAM` tables. It can have one of the following values to affect handling of the `DELAY_KEY_WRITE` table option that can be used in `CREATE TABLE` statements.

Option	Description
`OFF`	`DELAY_KEY_WRITE` is ignored.
`ON`	MySQL honors any `DELAY_KEY_WRITE` option specified in `CREATE TABLE` statements. This is the default value.
`ALL`	All new opened tables are treated as if they were created with the `DELAY_KEY_WRITE` option enabled.

If `DELAY_KEY_WRITE` is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all `MyISAM` tables by starting the server with the `--myisam-recover` option (for example, `--myisam-recover=BACKUP,FORCE`). See Section 5.1.3, “Server Command Options”, and Section 14.1.1, “MyISAM Startup Options”.

Warning

If you enable external locking with `--external-locking`, there is no protection against index corruption for tables that use delayed key writes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `delayed_insert_limit`

Command-Line Format	`--delayed_insert_limit=#`	
System Variable	Name	`delayed_insert_limit`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`100`
	Min Value	`1`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	`integer`
	Default	`100`
	Min Value	`1`
	Max Value	`18446744073709547520`

After inserting `delayed_insert_limit` delayed rows, the `INSERT DELAYED` handler thread checks whether there are any `SELECT` statements pending. If so, it permits them to execute before continuing to insert delayed rows.

- `delayed_insert_timeout`

Command-Line Format	`--delayed_insert_timeout=#`	
System Variable	Name	`delayed_insert_timeout`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`300`

How many seconds an `INSERT DELAYED` handler thread should wait for `INSERT` statements before terminating.

- `delayed_queue_size`

Command-Line Format	`--delayed_queue_size=#`	
System Variable	Name	`delayed_queue_size`
	Variable Scope	Global
	Dynamic Variable	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values (32-bit platforms)	**Type**	integer
	Default	1000
	Min Value	1
	Max Value	4294967295
Permitted Values (64-bit platforms)	**Type**	integer
	Default	1000
	Min Value	1
	Max Value	18446744073709547520

This is a per-table limit on the number of rows to queue when handling `INSERT DELAYED` statements. If the queue becomes full, any client that issues an `INSERT DELAYED` statement waits until there is room in the queue again.

- `div_precision_increment`

Introduced	5.0.6	
Command-Line Format	`--div_precision_increment=#`	
System Variable	**Name**	`div_precision_increment`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	4
	Min Value	0
	Max Value	30

This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the `/` operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+----------------+
| 1/7            |
+----------------+
| 0.142857142857 |
+----------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This variable was added in MySQL 5.0.6.

- `engine_condition_pushdown`

Introduced	5.0.3	
Command-Line Format	`--engine-condition-pushdown`	
System Variable	**Name**	`engine_condition_pushdown`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (>= 5.0.3)	**Type**	`boolean`
	Default	`OFF`

The engine condition pushdown optimization enables processing for certain comparisons to be “pushed down” to the storage engine level for more efficient execution. For more information, see Section 8.2.1.5, “Engine Condition Pushdown Optimization”.

Engine condition pushdown is used only by the `NDBCLUSTER` storage engine. Enabling this optimization on a MySQL Server acting as a MySQL Cluster SQL node causes `WHERE` conditions on unindexed columns to be evaluated on the cluster's data nodes and only the rows that match to be sent back to the SQL node that issued the query. This greatly reduces the amount of cluster data that must be sent over the network, increasing the efficiency with which results are returned.

The `engine_condition_pushdown` variable controls whether engine condition pushdown is enabled. By default, this variable is `OFF` (0). Setting it to `ON` (1) enables pushdown.

This variable was added in MySQL 5.0.3.

- `error_count`

The number of errors that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.5.14, “SHOW ERRORS Syntax”.

- `expire_logs_days`

Command-Line Format	`--expire_logs_days=#`	
System Variable	**Name**	`expire_logs_days`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`99`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of days for automatic binary log file removal. The default is 0, which means “no automatic removal.” Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in Section 5.4, “MySQL Server Logs”.

To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

- `flush`

Command-Line Format	`--flush`	
System Variable	Name	`flush`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`boolean`
	Default	`OFF`

If `ON`, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”. This variable is set to `ON` if you start `mysqld` with the `--flush` option.

- `flush_time`

Command-Line Format	`--flush_time=#`	
System Variable	Name	`flush_time`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`0`
	Min Value	`0`
Permitted Values (Windows)	Type	`integer`
	Default	`1800`
	Min Value	`0`

If this is set to a nonzero value, all tables are closed every `flush_time` seconds to free up resources and synchronize unflushed data to disk. This option is best used only on systems with minimal resources.

- `foreign_key_checks`

If set to 1 (the default), foreign key constraints for `InnoDB` tables are checked. If set to 0, foreign key constraints are ignored, with a couple of exceptions. When re-creating a table that was dropped, an error is returned if the table definition does not conform to the foreign key constraints referencing the table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Likewise, an `ALTER TABLE` operation returns an error if a foreign key definition is incorrectly formed. For more information, see Section 13.1.10.3, “Using FOREIGN KEY Constraints”.

Disabling foreign key checking can be useful for reloading `InnoDB` tables in an order different from that required by their parent/child relationships. See Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”.

Setting `foreign_key_checks` to 0 also affects data definition statements: `DROP DATABASE` drops a database even if it contains tables that have foreign keys that are referred to by tables outside the database, and `DROP TABLE` drops tables that have foreign keys that are referred to by other tables.

Note

Setting `foreign_key_checks` to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while `foreign_key_checks = 0` will not be verified for consistency.

- `ft_boolean_syntax`

Command-Line Format	`--ft_boolean_syntax=name`	
System Variable	**Name**	`ft_boolean_syntax`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`string`
	Default	`+ -><()~*:""&|`

The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See Section 12.9.2, “Boolean Full-Text Searches”.

The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

- Operator function is determined by position within the string.
- The replacement value must be 14 characters.
- Each character must be an ASCII nonalphanumeric character.
- Either the first or second character must be a space.
- No duplicates are permitted except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to “`:`”, “`&`”, and “`|`”) are reserved for future extensions.

- `ft_max_word_len`

Command-Line Format	`--ft_max_word_len=#`	
System Variable	**Name**	`ft_max_word_len`
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Min Value	`10`

The maximum length of the word to be included in a `FULLTEXT` index.

Note

`FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len`

Command-Line Format	`--ft_min_word_len=#`	
System Variable	**Name**	`ft_min_word_len`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`4`
	Min Value	`1`

The minimum length of the word to be included in a `FULLTEXT` index.

Note

`FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit`

Command-Line Format	`--ft_query_expansion_limit=#`	
System Variable	**Name**	`ft_query_expansion_limit`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`20`
	Min Value	`0`
	Max Value	`1000`

The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `ft_stopword_file`

Command-Line Format	`--ft_stopword_file=file_name`	
System Variable	Name	`ft_stopword_file`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`file name`

The file from which to read the list of stopwords for full-text searches. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `myisam/ft_static.c` file). Setting this variable to the empty string (`''`) disables stopword filtering. See also Section 12.9.4, “Full-Text Stopwords”.

Note

`FULLTEXT` indexes must be rebuilt after changing this variable or the contents of the stopword file. Use `REPAIR TABLE tbl_name QUICK`.

- `group_concat_max_len`

Command-Line Format	`--group_concat_max_len=#`	
System Variable	Name	`group_concat_max_len`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`1024`
	Min Value	`4`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	`integer`
	Default	`1024`
	Min Value	`4`
	Max Value	`18446744073709547520`

The maximum permitted result length in bytes for the `GROUP_CONCAT()` function. The default is 1024.

- `have_archive`

`YES` if `mysqld` supports `ARCHIVE` tables, `NO` if not.

- `have_bdb`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`YES` if `mysqld` supports `BDB` tables. `DISABLED` if `--skip-bdb` is used.

- `have_blackhole_engine`

  `YES` if `mysqld` supports `BLACKHOLE` tables, `NO` if not.

- `have_community_features`

  `YES` if statement profiling capability is present, `NO` if not. If present, the `profiling` system variable controls whether this capability is enabled or disabled. See Section 13.7.5.29, “SHOW PROFILES Syntax”.

  This variable was added in MySQL 5.0.82.

- `have_compress`

  `YES` if the `zlib` compression library is available to the server, `NO` if not. If not, the `COMPRESS()` and `UNCOMPRESS()` functions cannot be used.

- `have_crypt`

  `YES` if the `crypt()` system call is available to the server, `NO` if not. If not, the `ENCRYPT()` function cannot be used.

- `have_csv`

  `YES` if `mysqld` supports `CSV` tables, `NO` if not.

- `have_example_engine`

  `YES` if `mysqld` supports `EXAMPLE` tables, `NO` if not.

- `have_federated_engine`

  `YES` if `mysqld` supports `FEDERATED` tables, `NO` if not. This variable was added in MySQL 5.0.3.

- `have_geometry`

  `YES` if the server supports spatial data types, `NO` if not.

- `have_innodb`

  `YES` if `mysqld` supports `InnoDB` tables. `DISABLED` if `--skip-innodb` is used.

- `have_isam`

  This variable appears only for reasons of backward compatibility. It is always `NO` because `ISAM` tables are no longer supported.

- `have_merge_engine`

  `YES` if `mysqld` supports `MERGE` tables. `DISABLED` if `--skip-merge` is used. This variable was added in MySQL 5.0.24.

- `have_openssl`

  `YES` if `mysqld` supports SSL connections, `NO` if not. As of MySQL 5.0.38, this variable is an alias for `have_ssl`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `have_profiling`

  `YES` if statement profiling capability is present, `NO` if not. If present, the `profiling` system variable controls whether this capability is enabled or disabled. See Section 13.7.5.29, “SHOW PROFILES Syntax”.

  This variable was added in MySQL 5.0.82.

- `have_query_cache`

  `YES` if `mysqld` supports the query cache, `NO` if not.

- `have_raid`

  This variable appears only for reasons of backward compatibility. It is always `NO` because `RAID` tables are no longer supported.

- `have_rtree_keys`

  `YES` if `RTREE` indexes are available, `NO` if not. (These are used for spatial indexes in `MyISAM` tables.)

- `have_ssl`

  `YES` if `mysqld` supports SSL connections, `NO` if not. `DISABLED` indicates that the server was compiled with SSL support, but but was not started with the appropriate `--ssl-`*`xxx`* options. For more information, see Section 6.3.6.2, “Building MySQL with Support for Secure Connections”.

  This variable was added in MySQL 5.0.38. Before that, use `have_openssl`.

- `have_symlink`

  `YES` if symbolic link support is enabled, `NO` if not. This is required on Unix for support of the `DATA DIRECTORY` and `INDEX DIRECTORY` table options, and on Windows for support of data directory symlinks. If the server is started with the `--skip-symbolic-links` option, the value is `DISABLED`.

- `hostname`

Introduced	5.0.38	
System Variable	**Name**	`hostname`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`string`

  The server sets this variable to the server host name at startup. This variable was added in MySQL 5.0.38.

- `identity`

  This variable is a synonym for the `last_insert_id` variable. It exists for compatibility with other database systems. You can read its value with `SELECT @@identity`, and set it using `SET identity`.

- `init_connect`

Command-Line Format	`--init-connect=name`

This documentation is for an older version. If you're

System Variable	Name	init_connect
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string

A string to be executed by the server for each client that connects. The string consists of one or more SQL statements, separated by semicolon characters. For example, each client session begins by default with autocommit mode enabled. There is no global autocommit system variable to specify that autocommit should be disabled by default, but init_connect can be used to achieve the same effect:

```
SET GLOBAL init_connect='SET autocommit=0';
```

The init_connect variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]
init_connect='SET autocommit=0'
```

The content of init_connect is not executed for users that have the SUPER privilege. This is done so that an erroneous value for init_connect does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing init_connect for users that have the SUPER privilege enables them to open a connection and fix the init_connect value.

The server discards any result sets produced by statements in the value of of init_connect.

- init_file

Command-Line Format	--init-file=file_name	
System Variable	Name	init_file
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	file name

The name of the file specified with the --init-file option when you start the server. This should be a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments. For more information, see the description of --init-file.

Note that the --init-file option is unavailable if MySQL was configured with the --disable-grant-options option. See Section 2.17.3, “MySQL Source-Configuration Options”.

- innodb_*xxx*

  InnoDB system variables are listed in Section 14.2.2, “InnoDB Startup Options and System Variables”.

- insert_id

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The value to be used by the following `INSERT` or `ALTER TABLE` statement when inserting an `AUTO_INCREMENT` value. This is mainly used with the binary log.

- `interactive_timeout`

Command-Line Format	`--interactive_timeout=#`	
System Variable	Name	`interactive_timeout`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`28800`
	Min Value	`1`

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout`.

- `join_buffer_size`

Command-Line Format	`--join_buffer_size=#`	
System Variable	Name	`join_buffer_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`131072`
	Min Value	`8200`
	Max Value	`4294967295`

The minimum size of the buffer that is used for plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

There is no gain from setting the buffer larger than required to hold each matching row, and all joins allocate at least the minimum size, so use caution in setting this variable to a large value globally. It is better to keep the global setting small and change to a larger setting only in sessions that are doing large joins. Memory allocation time can cause substantial performance drops if the global size is larger than needed by most queries that use it.

The maximum permissible setting for `join_buffer_size` is 4GB−1.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For additional information about join buffering, see Section 8.2.1.8, “Nested-Loop Join Algorithms”.

- `keep_files_on_create`

Introduced	5.0.48	
Command-Line Format	`--keep_files_on_create=#`	
System Variable	**Name**	`keep_files_on_create`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`OFF`

If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, set the `keep_files_on_create` variable to `ON` (1), in which case `MyISAM` will not overwrite existing files and returns an error instead. The default value is `OFF` (0).

If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, MyISAM always returns an error. It will not overwrite a file in the specified directory.

This variable was added in MySQL 5.0.48.

- `key_buffer_size`

Command-Line Format	`--key_buffer_size=#`	
System Variable	**Name**	`key_buffer_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`8388608`
	Min Value	`8`
	Max Value	`4294967295`

Index blocks for `MyISAM` tables are buffered and are shared by all threads. `key_buffer_size` is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum permissible setting for `key_buffer_size` is 4GB−1 on 32-bit platforms. As of MySQL 5.0.52, larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB−1 with a warning). The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware

This documentation is for an older version. If you're
This documentation is for an older version. If you're

platform. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

You can increase the value to get better index handling for all reads and multiple writes; on a system whose primary function is to run MySQL using the `MyISAM` storage engine, 25% of the machine's total memory is an acceptable value for this variable. However, you should be aware that, if you make the value too large (for example, more than 50% of the machine's total memory), your system might start to page and become extremely slow. This is because MySQL relies on the operating system to perform file system caching for data reads, so you must leave some room for the file system cache. You should also consider the memory requirements of any other storage engines that you may be using in addition to `MyISAM`.

For even more speed when writing many rows at the same time, use `LOCK TABLES`. See Section 8.2.2.1, “Speed of INSERT Statements”.

You can check the performance of the key buffer by issuing a `SHOW STATUS` statement and examining the `Key_read_requests`, `Key_reads`, `Key_write_requests`, and `Key_writes` status variables. (See Section 13.7.5, “SHOW Syntax”.) The `Key_reads/Key_read_requests` ratio should normally be less than 0.01. The `Key_writes/Key_write_requests` ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the `DELAY_KEY_WRITE` table option.

The fraction of the key buffer in use can be determined using `key_buffer_size` in conjunction with the `Key_blocks_unused` status variable and the buffer block size, which is available from the `key_cache_block_size` system variable:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer is allocated internally for administrative structures. Factors that influence the amount of overhead for these structures include block size and pointer size. As block size increases, the percentage of the key buffer lost to overhead tends to decrease. Larger blocks results in a smaller number of read operations (because more keys are obtained per read), but conversely an increase in reads of keys that are not examined (if not all keys in a block are relevant to a query).

It is possible to create multiple `MyISAM` key caches. The size limit of 4GB applies to each cache individually, not as a group. See Section 8.10.1, “The MyISAM Key Cache”.

- `key_cache_age_threshold`

Command-Line Format	`--key_cache_age_threshold=#`	
System Variable	**Name**	`key_cache_age_threshold`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`300`
	Min Value	`100`
	Max Value	`4294967295`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`300`
	Min Value	`100`
	Max Value	`18446744073709547520`

This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See Section 8.10.1, “The MyISAM Key Cache”.

- `key_cache_block_size`

Command-Line Format	`--key_cache_block_size=#`	
System Variable	**Name**	`key_cache_block_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`1024`
	Min Value	`512`
	Max Value	`16384`

The size in bytes of blocks in the key cache. The default value is 1024. See Section 8.10.1, “The MyISAM Key Cache”.

- `key_cache_division_limit`

Command-Line Format	`--key_cache_division_limit=#`	
System Variable	**Name**	`key_cache_division_limit`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`100`
	Min Value	`1`
	Max Value	`100`

The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100. See Section 8.10.1, “The MyISAM Key Cache”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `language`

Command-Line Format	`--language=name`	
System Variable	Name	`language`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`directory name`
	Default	`/usr/local/mysql/share/mysql/english/`

The directory where error messages are located. See Section 10.2, “Setting the Error Message Language”.

- `large_files_support`

System Variable	Name	`large_files_support`
	Variable Scope	Global
	Dynamic Variable	No

Whether `mysqld` was compiled with options for large file support.

- `large_pages`

Introduced	5.0.3	
Command-Line Format	`--large-pages`	
System Variable	Name	`large_pages`
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Linux	
Permitted Values (Linux)	Type	`boolean`
	Default	`FALSE`

Whether large page support is enabled (via the `--large-pages` option). See Section 8.12.5.2, “Enabling Large Page Support”. This variable was added in MySQL 5.0.3.

For more information, see the entry for the `--large-pages` server option.

- `large_page_size`

Introduced	5.0.3	
System Variable	Name	`large_page_size`
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	No
Permitted Values (Linux)	**Type**	`integer`
	Default	`0`

If large page support is enabled, this shows the size of memory pages. Large memory pages are supported only on Linux; on other platforms, the value of this variable is always 0. This variable was added in MySQL 5.0.3.

For more information, see the entry for the `--large-pages` server option.

- `last_insert_id`

  The value to be returned from `LAST_INSERT_ID()`. This is stored in the binary log when you use `LAST_INSERT_ID()` in a statement that updates a table. Setting this variable does not update the value returned by the `mysql_insert_id()` C API function.

- `lc_time_names`

Introduced	5.0.25	
System Variable	**Name**	`lc_time_names`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`string`

This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting. For further information, see Section 10.7, “MySQL Server Locale Support”. This variable was added in MySQL 5.0.25.

- `license`

System Variable	**Name**	`license`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`string`
	Default	`GPL`

The type of license the server has.

- `local_infile`

System Variable	**Name**	`local_infile`
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`true`

Whether `LOCAL` is supported for `LOAD DATA INFILE` statements. If this variable is disabled, clients cannot use `LOCAL` in `LOAD DATA` statements. While the default for this variable is `true`, whether `LOAD DATA INFILE LOCAL` is actually permitted depends on how MySQL was compiled, as well as a number of settings on both the server and the client; see Section 6.1.6, “Security Issues with LOAD DATA LOCAL”, for details.

- `locked_in_memory`

System Variable	**Name**	`locked_in_memory`
	Variable Scope	Global
	Dynamic Variable	No

Whether `mysqld` was locked in memory with `--memlock`.

- `log`

Command-Line Format	`--log[=file_name]`	
System Variable	**Name**	`log`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

Whether logging of all statements to the general query log is enabled. See Section 5.4.2, “The General Query Log”.

- `log_bin_trust_function_creators`

Introduced	5.0.16	
Command-Line Format	`--log-bin-trust-function-creators`	
System Variable	**Name**	`log_bin_trust_function_creators`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that will cause unsafe events to be written to the binary log. If set to 0 (the default), users are not permitted to create or alter stored functions unless they have the `SUPER`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

privilege in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege. A setting of 0 also enforces the restriction that a function must be declared with the `DETERMINISTIC` characteristic, or with the `READS SQL DATA` or `NO SQL` characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. This variable also applies to trigger creation. See Section 18.6, "Binary Logging of Stored Programs".

This variable was added in MySQL 5.0.16.

- `log_bin_trust_routine_creators`

  This is the old name for `log_bin_trust_function_creators`. Before MySQL 5.0.16, it also applies to stored procedures, not just stored functions. As of 5.0.16, this variable is deprecated. It is recognized for backward compatibility but its use results in a warning.

  This variable was added in MySQL 5.0.6. It is removed in MySQL 5.5.

- `log_error`

Command-Line Format	`--log-error[=file_name]`	
System Variable	**Name**	`log_error`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

  The location of the error log, or empty if the server is writing error message to the standard error output. See Section 5.4.1, "The Error Log".

- `log_queries_not_using_indexes`

Command-Line Format	`--log-queries-not-using-indexes`	
System Variable (>= 5.0.23)	**Name**	`log_queries_not_using_indexes`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`OFF`

  Whether queries that do not use indexes are logged to the slow query log. See Section 5.4.4, "The Slow Query Log". This variable was added in MySQL 5.0.23.

- `log_slow_queries`

Command-Line Format	`--log-slow-queries[=name]`	
System Variable	**Name**	`log_slow_queries`
	Variable Scope	Global
	Dynamic Variable	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	Type	boolean

Whether slow queries should be logged. “Slow” is determined by the value of the `long_query_time` variable. See Section 5.4.4, “The Slow Query Log”.

- `log_warnings`

Command-Line Format	--log-warnings[=#]	
System Variable	Name	log_warnings
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	1
	Min Value	0
	Max Value	4294967295
Permitted Values (64-bit platforms)	Type	integer
	Default	1
	Min Value	0
	Max Value	18446744073709547520

Whether to produce additional warning messages to the error log. This variable is enabled (1) by default and can be disabled by setting it to 0. Aborted connections and access-denied errors for new connection attempts are logged if the value is greater than 1.

- `long_query_time`

Command-Line Format	--long_query_time=#	
System Variable	Name	long_query_time
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (<= 5.0.20)	Type	integer
	Default	10
	Min Value	1
Permitted Values (>= 5.0.21)	Type	numeric
	Default	10
	Min Value	0

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If a query takes longer than this many seconds, the server increments the `Slow_queries` status variable. If you are using the `--log-slow-queries` option, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum value is 1. The default is 10. See Section 5.4.4, “The Slow Query Log”.

- `low_priority_updates`

Command-Line Format	`--low-priority-updates`	
System Variable	Name	`low_priority_updates`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`boolean`
	Default	`FALSE`

If set to `1`, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`). This variable previously was named `sql_low_priority_updates`.

- `lower_case_file_system`

System Variable	Name	`lower_case_file_system`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`boolean`

This variable describes the case sensitivity of file names on the file system where the data directory is located. `OFF` means file names are case sensitive, `ON` means they are not case sensitive. This variable is read only because it reflects a file system attribute and setting it would have no effect on the file system.

- `lower_case_table_names`

Command-Line Format	`--lower_case_table_names[=#]`	
System Variable	Name	`lower_case_table_names`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`integer`
	Default	`0`
	Min Value	`0`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Max Value	2

If set to 0, table names are stored as specified and comparisons are case sensitive. If set to 1, table names are stored in lowercase on disk and comparisons are not case sensitive. If set to 2, table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. For additional information, see Section 9.2.2, “Identifier Case Sensitivity”.

On Windows the default value is 1. On OS X, the default value is 2.

You should *not* set `lower_case_table_names` to 0 if you are running MySQL on a system where the data directory resides on a case-insensitive file system (such as on Windows or OS X). It is an unsupported combination that could result in a hang condition when running an `INSERT INTO ... SELECT ... FROM tbl_name` operation with the wrong `tbl_name` letter case. With `MyISAM`, accessing table names using different letter cases could cause index corruption.

If you are using `InnoDB` or MySQL Cluster (`NDB`) tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

The setting of this variable has no effect on replication filtering options. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”, for more information.

You should not use different settings for `lower_case_table_names` on replication masters and slaves. In particular, you should not do this when the slave uses a case-sensitive file system, as this can cause replication to fail. For more information, see Section 16.4.1.29, “Replication and Variables”.

- `max_allowed_packet`

Command-Line Format	`--max_allowed_packet=#`	
System Variable	**Name**	`max_allowed_packet`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`1048576`
	Min Value	`1024`
	Max Value	`1073741824`

The maximum size of one packet or any generated/intermediate string.

The packet message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for `max_allowed_packet` is 1GB. The value should be a multiple of 1024; nonmultiples are rounded down to the nearest multiple.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When you change the message buffer size by changing the value of the `max_allowed_packet` variable, you should also change the buffer size on the client side if your client program permits it. The default `max_allowed_packet` value built in to the client library is 1GB, but individual client programs might override this. For example, `mysql` and `mysqldump` have defaults of 16MB and 24MB, respectively. They also enable you to change the client-side value by setting `max_allowed_packet` on the command line or in an option file.

As of MySQL 5.0.84, the session value of this variable is read only. Before 5.0.84, setting the session value is permitted but has no effect. The client can receive up to as many bytes as the session value. However, the server will not send to the client more bytes than the current global `max_allowed_packet` value. (The global value could be less than the session value if the global value is changed after the client connects.)

- `max_connect_errors`

Command-Line Format	`--max_connect_errors=#`	
System Variable	**Name**	`max_connect_errors`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`10`
	Min Value	`1`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`10`
	Min Value	`1`
	Max Value	`18446744073709547520`

If more than this many successive connection requests from a host are interrupted without a successful connection, the server blocks that host from further connections. You can unblock blocked hosts by flushing the host cache. To do so, issue a `FLUSH HOSTS` statement or execute a `mysqladmin flush-hosts` command. If a connection is established successfully within fewer than `max_connect_errors` attempts after a previous connection was interrupted, the error count for the host is cleared to zero. However, once a host is blocked, flushing the host cache is the only way to unblock it.

- `max_connections`

Command-Line Format	`--max_connections=#`	
System Variable	**Name**	`max_connections`
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`100`
	Min Value	`1`
	Max Value	`16384`

The maximum permitted number of simultaneous client connections. By default, this is 100. See Section B.5.2.7, “Too many connections”, for more information.

Increasing this value increases the number of file descriptors that `mysqld` requires. See Section 8.4.3.1, “How MySQL Opens and Closes Tables”, for comments on file descriptor limits.

- `max_delayed_threads`

Command-Line Format	`--max_delayed_threads=#`	
System Variable	**Name**	`max_delayed_threads`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`20`
	Min Value	`0`
	Max Value	`16384`

Do not start more than this number of threads to handle `INSERT DELAYED` statements. If you try to insert data into a new table after all `INSERT DELAYED` threads are in use, the row is inserted as if the `DELAYED` attribute was not specified. If you set this to 0, MySQL never creates a thread to handle `DELAYED` rows; in effect, this disables `DELAYED` entirely.

For the `SESSION` value of this variable, the only valid values are 0 or the `GLOBAL` value.

- `max_error_count`

Command-Line Format	`--max_error_count=#`	
System Variable	**Name**	`max_error_count`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`64`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Min Value	`0`
	Max Value	`65535`

The maximum number of error, warning, and note messages to be stored for display by the `SHOW ERRORS` and `SHOW WARNINGS` statements.

- `max_heap_table_size`

Command-Line Format		`--max_heap_table_size=#`
System Variable	**Name**	`max_heap_table_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`16777216`
	Min Value	`16384`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`16777216`
	Min Value	`16384`
	Max Value	`18446744073709551615`

This variable sets the maximum size to which user-created `MEMORY` tables are permitted to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE` or altered with `ALTER TABLE` or `TRUNCATE TABLE`. A server restart also sets the maximum size of existing `MEMORY` tables to the global `max_heap_table_size` value.

This variable is also used in conjunction with `tmp_table_size` to limit the size of internal in-memory tables. See Section 8.4.4, “Internal Temporary Table Use in MySQL”.

`max_heap_table_size` is not replicated. See Section 16.4.1.15, “Replication and MEMORY Tables”, and Section 16.4.1.29, “Replication and Variables”, for more information.

- `max_insert_delayed_threads`

System Variable	**Name**	`max_insert_delayed_threads`
	Variable Scope	Global, Session
	Dynamic Variable	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	Type	integer

This variable is a synonym for `max_delayed_threads`.

- `max_join_size`

Command-Line Format	`--max_join_size=#`	
System Variable	Name	`max_join_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`4294967295`
	Min Value	`1`
	Max Value	`4294967295`

Do not permit statements that probably need to examine more than `max_join_size` rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` disk seeks. By setting this value, you can catch statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than `DEFAULT` resets the value of `sql_big_selects` to `0`. If you set the `sql_big_selects` value again, the `max_join_size` variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

This variable previously was named `sql_max_join_size`.

- `max_length_for_sort_data`

Command-Line Format	`--max_length_for_sort_data=#`	
System Variable	Name	`max_length_for_sort_data`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`1024`
	Min Value	`4`
	Max Value	`8388608`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The cutoff on the size of index values that determines which `filesort` algorithm to use. See Section 8.2.1.11, “ORDER BY Optimization”.

- `max_prepared_stmt_count`

Introduced	5.0.21	
Command-Line Format	`--max_prepared_stmt_count=#` (>= 5.0.21)	
System Variable	**Name**	`max_prepared_stmt_count`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`16382`
	Min Value	`0`
	Max Value	`1048576`

This variable limits the total number of prepared statements in the server. (The sum of the number of prepared statements across all sessions.) It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. The default value is 16,382. The permissible range of values is from 0 to 1 million. Setting the value to 0 disables prepared statements. This variable was added in MySQL 5.0.21.

- `max_relay_log_size`

Command-Line Format	`--max_relay_log_size=#`	
System Variable	**Name**	`max_relay_log_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`1073741824`

If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` is 0, the server uses `max_binlog_size` for both the binary log and the relay log. If `max_relay_log_size` is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` to between 4096 bytes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

and 1GB (inclusive), or to 0. The default value is 0. See Section 16.2.1, “Replication Implementation Details”.

- `max_seeks_for_key`

Command-Line Format	`--max_seeks_for_key=#`	
System Variable	Name	`max_seeks_for_key`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`4294967295`
	Min Value	`1`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	`integer`
	Default	`18446744073709547520`
	Min Value	`1`
	Max Value	`18446744073709547520`

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see Section 13.7.5.18, “SHOW INDEX Syntax”). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- `max_sort_length`

Command-Line Format	`--max_sort_length=#`	
System Variable	Name	`max_sort_length`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`1024`
	Min Value	`4`
	Max Value	`8388608`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of bytes to use when sorting data values. The server uses only the first `max_sort_length` bytes of each value and ignores the rest. Consequently, values that differ only after the first `max_sort_length` bytes compare as equal for `GROUP BY`, `ORDER BY`, and `DISTINCT` operations.

- `max_sp_recursion_depth`

Introduced	5.0.17	
Command-Line Format	`--max_sp_recursion_depth[=#]`	
System Variable	**Name**	`max_sp_recursion_depth`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Max Value	`255`

The number of times that any given stored procedure may be called recursively. The default value for this option is 0, which completely disables recursion in stored procedures. The maximum value is 255.

Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth`, it may be necessary to increase thread stack size by increasing the value of `thread_stack` at server startup.

This variable was added in MySQL 5.0.17.

- `max_tmp_tables`

This variable is unused.

- `max_user_connections`

Command-Line Format	`--max_user_connections=#`	
System Variable (<= 5.0.3)	**Name**	`max_user_connections`
	Variable Scope	Global
	Dynamic Variable	Yes
System Variable (>= 5.0.3)	**Name**	`max_user_connections`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Max Value	4294967295

The maximum number of simultaneous connections permitted to any given MySQL user account. A value of 0 (the default) means “no limit.”

Before MySQL 5.0.3, this variable has only a global value that can be set at server startup or runtime. As of MySQL 5.0.3, it also has a read-only session value that indicates the effective simultaneous-connection limit that applies to the account associated with the current session. The session value is initialized as follows:

- If the user account has a nonzero `MAX_USER_CONNECTIONS` resource limit, the session `max_user_connections` value is set to that limit.
- Otherwise, the session `max_user_connections` value is set to the global value.

Account resource limits are specified using the `GRANT` statement. See Section 6.3.4, “Setting Account Resource Limits”, and Section 13.7.1.3, “GRANT Syntax”.

- `max_write_lock_count`

Command-Line Format	`--max_write_lock_count=#`	
System Variable	**Name**	`max_write_lock_count`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`4294967295`
	Min Value	`1`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`18446744073709547520`
	Min Value	`1`
	Max Value	`18446744073709547520`

After this many write locks, permit some pending read lock requests to be processed in between.

- `multi_range_count`

Introduced	5.0.3	
Command-Line Format	`--multi_range_count=#`	
System Variable	**Name**	`multi_range_count`
	Variable Scope	Global, Session

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`256`
	Min Value	`1`
	Max Value	`4294967295`

The maximum number of ranges to send to a table handler at once during range selects. The default value is 256. Sending multiple ranges to a handler at once can improve the performance of certain selects dramatically. This is especially true for the `NDBCLUSTER` table handler, which needs to send the range requests to all nodes. Sending a batch of those requests at once reduces communication costs significantly.

This variable was added in MySQL 5.0.3.

- `myisam_data_pointer_size`

Command-Line Format	`--myisam_data_pointer_size=#`	
System Variable	**Name**	`myisam_data_pointer_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (<= 5.0.5)	**Type**	`integer`
	Default	`4`
	Min Value	`2`
	Max Value	`8`
Permitted Values (>= 5.0.6)	**Type**	`integer`
	Default	`6`
	Min Value	`2`
	Max Value	`7`

The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6 (4 before MySQL 5.0.6). See Section B.5.2.12, “The table is full”.

- `myisam_max_extra_sort_file_size` (*DEPRECATED*)

This variable is unused as of MySQL 5.0.6.

- `myisam_max_sort_file_size`

Command-Line Format	`--myisam_max_sort_file_size=#`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variable	Name	myisam_max_sort_file_size
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	2147483648
Permitted Values (64-bit platforms)	Type	integer
	Default	9223372036854775807

The maximum size of the temporary file that MySQL is permitted to use while re-creating a MyISAM index (during REPAIR TABLE, ALTER TABLE, or LOAD DATA INFILE). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

If MyISAM index files exceed this size and disk space is available, increasing the value may help performance. The space must be available in the file system containing the directory where the original index file is located.

- myisam_mmap_size

Introduced	5.0.90	
Command-Line Format	--myisam_mmap_size=#	
System Variable	Name	myisam_mmap_size
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (32-bit platforms)	Type	integer
	Default	4294967295
	Min Value	7
	Max Value	4294967295
Permitted Values (64-bit platforms)	Type	integer
	Default	18446744073709547520
	Min Value	7
	Max Value	18446744073709547520

The maximum amount of memory to use for memory mapping compressed MyISAM files. If many compressed MyISAM tables are used, the value can be decreased to reduce the likelihood of memory-swapping problems. This variable was added in MySQL 5.0.90.

- myisam_recover_options

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variable	Name	myisam_recover_options
	Variable Scope	Global
	Dynamic Variable	No

The value of the `--myisam-recover` option. See Section 5.1.3, “Server Command Options”.

- `myisam_repair_threads`

Command-Line Format		--myisam_repair_threads=#
System Variable	Name	myisam_repair_threads
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	1
	Min Value	1
	Max Value	4294967295
Permitted Values (64-bit platforms)	Type	integer
	Default	1
	Min Value	1
	Max Value	18446744073709547520

If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1.

Note

Multi-threaded repair is still *beta-quality* code.

- `myisam_sort_buffer_size`

Command-Line Format		--myisam_sort_buffer_size=#
System Variable	Name	myisam_sort_buffer_size
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (Windows)	Type	integer
	Default	8388608

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Min Value	4096
	Max Value	4294967295
Permitted Values (Other, 32-bit platforms)	**Type**	integer
	Default	8388608
	Min Value	4096
	Max Value	4294967295
Permitted Values (Other, 64-bit platforms)	**Type**	integer
	Default	8388608
	Min Value	4096
	Max Value	18446744073709547520

The size of the buffer that is allocated when sorting MyISAM indexes during a REPAIR TABLE or when creating indexes with CREATE INDEX or ALTER TABLE.

The maximum permissible setting for myisam_sort_buffer_size is 4GB−1.

- myisam_stats_method

Introduced	5.0.14	
Command-Line Format	--myisam_stats_method=name	
System Variable	**Name**	myisam_stats_method
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (>= 5.0.14)	**Type**	enumeration
	Default	nulls_unequal
	Valid Values	nulls_equal
		nulls_unequal
		nulls_ignored

How the server treats NULL values when collecting statistics about the distribution of index values for MyISAM tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.7, “MyISAM Index Statistics Collection”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Any unique prefix of a valid value may be used to set the value of this variable.

This variable was added in MySQL 5.0.14. For older versions, the statistics collection method is equivalent to `nulls_equal`.

- `named_pipe`

System Variable	Name	named_pipe
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Windows	
Permitted Values (Windows)	Type	boolean
	Default	OFF

(Windows only.) Indicates whether the server supports connections over named pipes.

- `net_buffer_length`

Command-Line Format	--net_buffer_length=#	
System Variable	Name	net_buffer_length
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	16384
	Min Value	1024
	Max Value	1048576

Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by `net_buffer_length` but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` after each SQL statement.

This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which `net_buffer_length` can be set is 1MB.

As of MySQL 5.0.84, the session value of this variable is read only. Before 5.0.84, setting the session value is permitted but has no effect.

- `net_read_timeout`

Command-Line Format	--net_read_timeout=#	
System Variable	Name	net_read_timeout

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`30`
	Min Value	`1`

The number of seconds to wait for more data from a connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made through Unix socket files, named pipes, or shared memory. When the server is reading from the client, `net_read_timeout` is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` is the timeout value controlling when to abort. See also `slave_net_timeout`.

On Linux, the `NO_ALARM` build flag affects timeout behavior as indicated in the description of the `net_retry_count` system variable.

- `net_retry_count`

Command-Line Format	`--net_retry_count=#`	
System Variable	**Name**	`net_retry_count`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`10`
	Min Value	`1`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`10`
	Min Value	`1`
	Max Value	`18446744073709547520`

If a read or write on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

On Linux, the `NO_ALARM` build flag (`-DNO_ALARM`) modifies how the binary treats both `net_read_timeout` and `net_write_timeout`. With this flag enabled, neither timer cancels the current statement until after the failing connection has been waited on an additional `net_retry_count` times. This means that the effective timeout value becomes `(timeout setting) × (net_retry_count+1)`.

- `net_write_timeout`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command-Line Format	--net_write_timeout=#	
System Variable	Name	net_write_timeout
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	60
	Min Value	1

The number of seconds to wait for a block to be written to a connection before aborting the write. This timeout applies only to TCP/IP connections, not to connections made using Unix socket files, named pipes, or shared memory. See also net_read_timeout.

On Linux, the NO_ALARM build flag affects timeout behavior as indicated in the description of the net_retry_count system variable.

- new

Command-Line Format	--new	
System Variable	Name	new
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Disabled by	skip-new	
Permitted Values	Type	boolean
	Default	FALSE

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. Its value is always OFF.

- old_passwords

System Variable	Name	old_passwords
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	0

This variable controls the password hashing method used by the PASSWORD() function. It also influences password hashing performed by CREATE USER and GRANT statements that specify a password using an IDENTIFIED BY clause.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The value determines whether or not to use “old” native MySQL password hashing. A value of 0 (or `OFF`) causes passwords to be encrypted using the format available from MySQL 4.1 on. A value of 1 (or `ON`) causes password encryption to use the older pre-4.1 format.

If `old_passwords=1`, `PASSWORD(str)` returns the same value as `OLD_PASSWORD(str)`. The latter function is not affected by the value of `old_passwords`.

For information about hashing formats, see Section 6.1.2.4, “Password Hashing in MySQL”.

- `one_shot`

  This is not a variable, but it can be used when setting some variables. It is described in Section 13.7.4, “SET Syntax”.

- `open_files_limit`

Command-Line Format	`--open-files-limit=#`	
System Variable	**Name**	`open_files_limit`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`platform dependent`

The number of files that the operating system permits `mysqld` to open. This is the real value permitted by the system and might be different from the value you gave using the `--open-files-limit` option to `mysqld` or `mysqld_safe`. The value is 0 on systems where MySQL cannot change the number of open files.

- `optimizer_prune_level`

Introduced	5.0.1	
Command-Line Format	`--optimizer_prune_level[=#]`	
System Variable	**Name**	`optimizer_prune_level`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`1`

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive

This documentation is for an older version. If you're

search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans. This variable was added in MySQL 5.0.1.

- `optimizer_search_depth`

Introduced	5.0.1	
Command-Line Format	`--optimizer_search_depth[=#]`	
System Variable	**Name**	`optimizer_search_depth`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`62`
	Min Value	`0`
	Max Value	`63`

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value. If set to 63, the optimizer switches to the algorithm used in MySQL 5.0.0 (and previous versions) for performing searches. This variable was added in MySQL 5.0.1.

- `pid_file`

Command-Line Format	`--pid-file=file_name`	
System Variable	**Name**	`pid_file`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

The path name of the process ID (PID) file. This variable can be set with the `--pid-file` option.

- `plugin_dir`

Introduced	5.0.67	
Command-Line Format	`--plugin_dir=dir_name`	
System Variable	**Name**	`plugin_dir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	

The path name of the plugin directory. This variable was added in MySQL 5.0.67. If the value is nonempty, user-defined function object files must be located in this directory. If the value is empty, the behavior that is used before 5.0.67 applies: The UDF object files must be located in a directory that is searched by your system's dynamic linker.

If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

- `port`

Command-Line Format	`--port=#`	
System Variable	**Name**	`port`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`3306`
	Min Value	`0`
	Max Value	`65535`

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

Command-Line Format	`--preload_buffer_size=#`	
System Variable	**Name**	`preload_buffer_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`32768`
	Min Value	`1024`
	Max Value	`1073741824`

The size of the buffer that is allocated when preloading indexes.

- `prepared_stmt_count`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Introduced	5.0.21	
Removed	5.0.31	
System Variable (>= 5.0.21, <= 5.0.31)	**Name**	`prepared_stmt_count`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`

The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` system variable.) This variable was added in MySQL 5.0.21. In MySQL 5.0.32, it was converted to the global `Prepared_stmt_count` status variable.

- `profiling`

  If set to 0 or `OFF` (the default), statement profiling is disabled. If set to 1 or `ON`, statement profiling is enabled and the `SHOW PROFILE` and `SHOW PROFILES` statements provide access to profiling information. See Section 13.7.5.29, “SHOW PROFILES Syntax”. This variable was added in MySQL 5.0.37. *Note*: This option does not apply to MySQL Enterprise Server users.

- `profiling_history_size`

  The number of statements for which to maintain profiling information if `profiling` is enabled. The default value is 15. The maximum value is 100. Setting the value to 0 effectively disables profiling. See Section 13.7.5.29, “SHOW PROFILES Syntax”. This variable was added in MySQL 5.0.37. *Note*: This option does not apply to MySQL Enterprise Server users.

- `protocol_version`

System Variable	**Name**	`protocol_version`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`

The version of the client/server protocol used by the MySQL server.

- `pseudo_thread_id`

System Variable	**Name**	`pseudo_thread_id`
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`

This variable is for internal server use.

- `query_alloc_block_size`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command-Line Format	--query_alloc_block_size=#	
System Variable	Name	query_alloc_block_size
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	8192
	Min Value	1024
	Max Value	4294967295
	Block Size	1024
Permitted Values (64-bit platforms)	Type	integer
	Default	8192
	Min Value	1024
	Max Value	18446744073709547520
	Block Size	1024

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this parameter.

- query_cache_limit

Command-Line Format	--query_cache_limit=#	
System Variable	Name	query_cache_limit
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	1048576
	Min Value	0
	Max Value	4294967295
Permitted Values (64-bit platforms)	Type	integer
	Default	1048576
	Min Value	0

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Max Value	18446744073709547520

Do not cache results that are larger than this number of bytes. The default value is 1MB.

- `query_cache_min_res_unit`

Command-Line Format	`--query_cache_min_res_unit=#`	
System Variable	**Name**	`query_cache_min_res_unit`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`4096`
	Min Value	`512`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`4096`
	Min Value	`512`
	Max Value	`18446744073709547520`

The minimum size (in bytes) for blocks allocated by the query cache. The default value is 4096 (4KB). Tuning information for this variable is given in Section 8.10.3.3, “Query Cache Configuration”.

- `query_cache_size`

Command-Line Format	`--query_cache_size=#`	
System Variable	**Name**	`query_cache_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`0`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Min Value	0
	Max Value	18446744073709547520

The amount of memory allocated for caching query results. The default value is 0, which disables the query cache. The permissible values are multiples of 1024; other values are rounded down to the nearest multiple. query_cache_size bytes of memory are allocated even if query_cache_type is set to 0. See Section 8.10.3.3, “Query Cache Configuration”, for more information.

The query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value of query_cache_size too small, a warning will occur, as described in Section 8.10.3.3, “Query Cache Configuration”.

- query_cache_type

Command-Line Format	--query_cache_type=#	
System Variable	**Name**	query_cache_type
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	enumeration
	Default	1
	Valid Values	0
		1
		2

Set the query cache type. Setting the GLOBAL value sets the type for all clients that connect thereafter. Individual clients can set the SESSION value to affect their own use of the query cache. Possible values are shown in the following table.

Option	Description
0 or OFF	Do not cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set query_cache_size to 0.
1 or ON	Cache all cacheable query results except for those that begin with SELECT SQL_NO_CACHE.
2 or DEMAND	Cache results only for cacheable queries that begin with SELECT SQL_CACHE.

This variable defaults to ON.

Any unique prefix of a valid value may be used to set the value of this variable.

- query_cache_wlock_invalidate

Command-Line Format	--query_cache_wlock_invalidate	
System Variable	**Name**	query_cache_wlock_invalidate

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Normally, when one client acquires a `WRITE` lock on a `MyISAM` table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting this variable to 1 causes acquisition of a `WRITE` lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.

- `query_prealloc_size`

Command-Line Format	`--query_prealloc_size=#`	
System Variable	**Name**	`query_prealloc_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`8192`
	Min Value	`8192`
	Max Value	`4294967295`
	Block Size	`1024`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`8192`
	Min Value	`8192`
	Max Value	`18446744073709547520`
	Block Size	`1024`

The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- `rand_seed1`

The `rand_seed1` and `rand_seed2` variables exist as session variables only, and can be set but not read. They are not shown in the output of `SHOW VARIABLES`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The purpose of these variables is to support replication of the `RAND()` function. For statements that invoke `RAND()`, the master passes two values to the slave, where they are used to seed the random number generator. The slave uses these values to set the session variables `rand_seed1` and `rand_seed2` so that `RAND()` on the slave generates the same value as on the master.

- `rand_seed2`

  See the description for `rand_seed1`.

- `range_alloc_block_size`

Command-Line Format	`--range_alloc_block_size=#`	
System Variable	**Name**	`range_alloc_block_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`4096`
	Min Value	`4096`
	Max Value	`4294967295`
	Block Size	`1024`

The size of blocks that are allocated when doing range optimization.

- `read_buffer_size`

Command-Line Format	`--read_buffer_size=#`	
System Variable	**Name**	`read_buffer_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`131072`
	Min Value	`8200`
	Max Value	`2147479552`

Each thread that does a sequential scan allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072. The value of this variable should be a multiple of 4KB. If it is set to a value that is not a multiple of 4KB, its value will be rounded down to the nearest multiple of 4KB.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The maximum permissible setting for `read_buffer_size` is 2GB.

`read_buffer_size` and `read_rnd_buffer_size` are not specific to any storage engine and apply in a general manner for optimization. See Section 8.12.5.1, "How MySQL Uses Memory", for example.

- `read_only`

Command-Line Format	`--read_only`	
System Variable	Name	`read_only`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`boolean`
	Default	`OFF`

When the `read_only` system variable is enabled, the server permits no client updates except from users who have the `SUPER` privilege. This variable is disabled by default.

Even with `read_only` enabled, the server permits these operations:

- Updates performed by slave threads, if the server is a replication slave. In replication setups, it can be useful to enable `read_only` on slave servers to ensure that slaves accept updates only from the master server and not from clients.
- Use of `ANALYZE TABLE` or `OPTIMIZE TABLE` statements. The purpose of read-only mode is to prevent changes to table structure or contents. Analysis and optimization do not qualify as such changes. This means, for example, that consistency checks on read-only replication slaves can be performed with `mysqlcheck --all-databases --analyze`.
- Operations on `TEMPORARY` tables, as of MySQL 5.0.16.

`read_only` exists only as a `GLOBAL` variable, so changes to its value require the `SUPER` privilege. Changes to `read_only` on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.

Important

As of MySQL 5.1, enabling `read_only` prevents users not having the `SUPER` privilege from using account-management statements such as `CREATE USER` or `SET PASSWORD`. This is not the case for MySQL 5.0. When replicating from a MySQL 5.0 master to a MySQL 5.1 or later slave, check whether this will have an impact on your applications.

- `read_rnd_buffer_size`

Command-Line Format	`--read_rnd_buffer_size=#`	
System Variable	Name	`read_rnd_buffer_size`
	Variable Scope	Global, Session

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`262144`
	Min Value	`8200`
	Max Value	`2147483647`

When reading rows in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See Section 8.2.1.11, “ORDER BY Optimization”. Setting the variable to a large value can improve `ORDER BY` performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

The maximum permissible setting for `read_rnd_buffer_size` is 2GB.

`read_buffer_size` and `read_rnd_buffer_size` are not specific to any storage engine and apply in a general manner for optimization. See Section 8.12.5.1, “How MySQL Uses Memory”, for example.

- `relay_log_purge`

Command-Line Format	`--relay_log_purge`	
System Variable	**Name**	`relay_log_purge`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (`ON`).

- `relay_log_space_limit`

Command-Line Format	`--relay_log_space_limit=#`	
System Variable	**Name**	`relay_log_space_limit`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`18446744073709547520`

The maximum amount of space to use for all relay logs.

- `secure_auth`

Command-Line Format	`--secure-auth`	
System Variable	**Name**	`secure_auth`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`OFF`

If this variable is enabled, the server blocks connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format.

Enable this variable to prevent all use of passwords employing the old format (and hence insecure communication over the network).

Server startup fails with an error if this variable is enabled and the privilege tables are in pre-4.1 format. See Section B.5.2.4, “Client does not support authentication protocol”.

- `secure_file_priv`

Introduced	5.0.38	
Command-Line Format	`--secure-file-priv=dir_name`	
System Variable	**Name**	`secure_file_priv`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`string`
	Default	`empty`
	Valid Values	`empty`
		`dirname`

This variable is used to limit the effect of data import and export operations, such as those performed by the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. By default, this variable is empty. If set to the name of a directory, it limits import and export operations to work only with files in that directory.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This variable was added in MySQL 5.0.38.

- `server_id`

Command-Line Format	`--server-id=#`	
System Variable	Name	`server_id`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`

The server ID, used in replication to give each master and slave a unique identity. This variable is set by the `--server-id` option. For each server participating in replication, you should pick a positive integer in the range from 1 to $2^{32} - 1$ to act as that server's ID.

- `shared_memory`

Command-Line Format	`--shared_memory[={0,1}]`	
System Variable	Name	`shared_memory`
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Windows	
Permitted Values	Type	`boolean`
	Default	`FALSE`

(Windows only.) Whether the server permits shared-memory connections.

- `shared_memory_base_name`

Command-Line Format	`--shared_memory_base_name=name`	
System Variable	Name	`shared_memory_base_name`
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Windows	
Permitted Values	Type	`string`
	Default	`MYSQL`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is `MYSQL`. The name is case sensitive.

- `skip_external_locking`

Command-Line Format	`--skip-external-locking`	
System Variable	Name	`skip_external_locking`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`boolean`
	Default	`ON`

This is `OFF` if `mysqld` uses external locking (system locking), `ON` if external locking is disabled. This affects only `MyISAM` table access.

This variable is set by the `--external-locking` or `--skip-external-locking` option. External locking is disabled by default.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.11.4, “External Locking”.

- `skip_networking`

Command-Line Format	`--skip-networking`	
System Variable	Name	`skip_networking`
	Variable Scope	Global
	Dynamic Variable	No

This is `ON` if the server permits only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. On NetWare, only TCP/IP connections are supported, so do not set this variable to `ON`. This variable can be set to `ON` with the `--skip-networking` option.

- `skip_show_database`

Command-Line Format	`--skip-show-database`	
System Variable	Name	`skip_show_database`
	Variable Scope	Global
	Dynamic Variable	No

This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the `SHOW DATABASES` privilege: If the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

variable value is ON, the SHOW DATABASES statement is permitted only to users who have the SHOW DATABASES privilege, and the statement displays all database names. If the value is OFF, SHOW DATABASES is permitted to all users, but displays the names of only those databases for which the user has the SHOW DATABASES or other privilege. (Note that *any* global privilege is considered a privilege for the database.)

- slow_launch_time

Command-Line Format	--slow_launch_time=#	
System Variable	Name	slow_launch_time
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	2

If creating a thread takes longer than this many seconds, the server increments the Slow_launch_threads status variable.

- socket

Command-Line Format	--socket={file_name|pipe_name}	
System Variable	Name	socket
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string
	Default	/tmp/mysql.sock

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is /tmp/mysql.sock. (For some distribution formats, the directory might be different, such as /var/lib/mysql for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is MySQL (not case sensitive).

- sort_buffer_size

Command-Line Format	--sort_buffer_size=#	
System Variable	Name	sort_buffer_size
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	2097144

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Min Value	32768
	Max Value	4294967295

Each session that needs to do a sort allocates a buffer of this size. `sort_buffer_size` is not specific to any storage engine and applies in a general manner for optimization. See Section 8.2.1.11, “ORDER BY Optimization”, for example.

If you see many `Sort_merge_passes` per second in `SHOW GLOBAL STATUS` output, you can consider increasing the `sort_buffer_size` value to speed up `ORDER BY` or `GROUP BY` operations that cannot be improved with query optimization or improved indexing. The entire buffer is allocated even if it is not all needed, so setting it larger than required globally will slow down most queries that sort. It is best to increase it as a session setting, and only for the sessions that need a larger size. On Linux, there are thresholds of 256KB and 2MB where larger values may significantly slow down memory allocation, so you should consider staying below one of those values. Experiment to find the best value for your workload. See Section B.5.3.5, “Where MySQL Stores Temporary Files”.

The maximum permissible setting for `sort_buffer_size` is 4GB−1.

- `sql_auto_is_null`

System Variable	**Name**	`sql_auto_is_null`
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`1`

If this variable is set to 1 (the default), then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` function. For details, including the return value after a multiple-row insert, see Section 12.13, “Information Functions”. If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` comparison is used by some ODBC programs, such as Access. See Obtaining Auto-Increment Values. This behavior can be disabled by setting `sql_auto_is_null` to 0.

- `sql_big_selects`

System Variable	**Name**	`sql_big_selects`
	Variable Scope	Session
	Dynamic Variable	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	**Type**	`boolean`
	Default	`1`

If set to 0, MySQL aborts `SELECT` statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of `max_join_size`). This is useful when an inadvisable `WHERE` statement has been issued. The default value for a new connection is 1, which permits all `SELECT` statements.

If you set the `max_join_size` system variable to a value other than `DEFAULT`, `sql_big_selects` is set to 0.

- `sql_buffer_result`

System Variable	**Name**	`sql_buffer_result`
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`0`

If set to 1, `sql_buffer_result` forces results from `SELECT` statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is 0.

- `sql_log_bin`

System Variable	**Name**	`sql_log_bin`
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`

If set to 0, no logging is done to the binary log for the client. The client must have the `SUPER` privilege to set this option. The default value is 1.

- `sql_log_off`

System Variable	**Name**	`sql_log_off`
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`0`

If set to 1, no logging is done to the general query log for this client. The client must have the `SUPER` privilege to set this option. The default value is 0.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `sql_log_update`

Deprecated	5.0.0, by sql_log_bin	
System Variable	**Name**	`sql_log_update`
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`

This variable is deprecated, and is mapped to `sql_log_bin`. It is removed in MySQL 5.5.

- `sql_mode`

Command-Line Format	`--sql-mode=name`	
System Variable	**Name**	`sql_mode`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`set`
	Default	`''`
	Valid Values	`ALLOW_INVALID_DATES`
		`ANSI_QUOTES`
		`ERROR_FOR_DIVISION_BY_ZERO`
		`HIGH_NOT_PRECEDENCE`
		`IGNORE_SPACE`
		`NO_AUTO_CREATE_USER`
		`NO_AUTO_VALUE_ON_ZERO`
		`NO_BACKSLASH_ESCAPES`
		`NO_DIR_IN_CREATE`
		`NO_ENGINE_SUBSTITUTION`
		`NO_FIELD_OPTIONS`
		`NO_KEY_OPTIONS`
		`NO_TABLE_OPTIONS`
		`NO_UNSIGNED_SUBTRACTION`
		`NO_ZERO_DATE`
		`NO_ZERO_IN_DATE`
		`ONLY_FULL_GROUP_BY`
		`PAD_CHAR_TO_FULL_LENGTH`
		`PIPES_AS_CONCAT`
		`REAL_AS_FLOAT`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

		STRICT_ALL_TABLES
		STRICT_TRANS_TABLES

The current server SQL mode, which can be set dynamically. For details, see Section 5.1.7, “Server SQL Modes”.

Note

MySQL installation programs may configure the SQL mode during the installation process. If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- sql_notes

  If set to 1 (the default), warnings of Note level increment warning_count and the server records them. If set to 0, Note warnings do not increment warning_count and the server does not record them. mysqldump includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation. sql_notes was added in MySQL 5.0.3.

- sql_quote_show_create

  If set to 1 (the default), the server quotes identifiers for SHOW CREATE TABLE and SHOW CREATE DATABASE statements. If set to 0, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See Section 13.7.5.9, “SHOW CREATE TABLE Syntax”, and Section 13.7.5.6, “SHOW CREATE DATABASE Syntax”.

- sql_safe_updates

  If set to 1, MySQL aborts UPDATE or DELETE statements that do not use a key in the WHERE clause or a LIMIT clause. (Specifically, UPDATE statements must have a WHERE clause that uses a key or a LIMIT clause, or both. DELETE statements must have both.) This makes it possible to catch UPDATE or DELETE statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is 0.

- sql_select_limit

System Variable	**Name**	sql_select_limit
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	integer

  The maximum number of rows to return from SELECT statements. The default value for a new connection is the maximum number of rows that the server permits per table, which depends on the server configuration and may be affected if the server build was configured with --with-big-tables. Typical default values are $(2^{32})-1$ or $(2^{64})-1$. If you have changed the limit, the default value can be restored by assigning a value of DEFAULT.

  If a SELECT has a LIMIT clause, the LIMIT takes precedence over the value of sql_select_limit.

- sql_warnings

This documentation is for an older version. If you're

This variable controls whether single-row `INSERT` statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string.

- `ssl_ca`

Introduced	5.0.23	
Command-Line Format	`--ssl-ca=file_name`	
System Variable	**Name**	`ssl_ca`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

The path to a file with a list of trusted SSL CAs. This variable was added in MySQL 5.0.23.

- `ssl_capath`

Introduced	5.0.23	
Command-Line Format	`--ssl-capath=dir_name`	
System Variable	**Name**	`ssl_capath`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

The path to a directory that contains trusted SSL CA certificates in PEM format. This variable was added in MySQL 5.0.23.

- `ssl_cert`

Introduced	5.0.23	
Command-Line Format	`--ssl-cert=file_name`	
System Variable	**Name**	`ssl_cert`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

The name of the SSL certificate file to use for establishing a secure connection. This variable was added in MySQL 5.0.23.

- `ssl_cipher`

Introduced	5.0.23
Command-Line Format	`--ssl-cipher=name`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variable	Name	ssl_cipher
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string

A list of permissible ciphers to use for SSL encryption. This variable was added in MySQL 5.0.23.

- ssl_key

Introduced	5.0.23	
Command-Line Format	--ssl-key=file_name	
System Variable	Name	ssl_key
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	file name

The name of the SSL key file to use for establishing a secure connection. This variable was added in MySQL 5.0.23.

- storage_engine

System Variable	Name	storage_engine
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	enumeration
	Default	MyISAM

The default storage engine (table type). To set the storage engine at server startup, use the --default-storage-engine option. See Section 5.1.3, “Server Command Options”.

To see which storage engines are available and enabled, use the SHOW ENGINES statement.

- sync_frm

Command-Line Format	--sync-frm	
System Variable	Name	sync_frm
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	TRUE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If this variable is set to 1, when any nontemporary table is created its `.frm` file is synchronized to disk (using `fdatasync()`). This is slower but safer in case of a crash. The default is 1.

- `system_time_zone`

System Variable	Name	system_time_zone
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `system_time_zone`. Typically the time zone is specified by the `TZ` environment variable. It also can be specified using the `--timezone` option of the `mysqld_safe` script.

The `system_time_zone` variable differs from `time_zone`. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See Section 10.6, “MySQL Server Time Zone Support”.

- `table_cache`

Command-Line Format	--table_cache=#	
System Variable	Name	table_cache
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	64
	Min Value	1
	Max Value	524288

The number of open tables for all threads. Increasing this value increases the number of file descriptors that `mysqld` requires. You can check whether you need to increase the table cache by checking the `Opened_tables` status variable. See Section 5.1.6, “Server Status Variables”. If the value of `Opened_tables` is large and you do not use `FLUSH TABLES` often (which just forces all tables to be closed and reopened), then you should increase the value of the `table_cache` variable. For more information about the table cache, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

- `table_lock_wait_timeout`

Introduced	5.0.10	
Command-Line Format	--table_lock_wait_timeout=#	
System Variable	Name	table_lock_wait_timeout

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	50
	Min Value	1
	Max Value	1073741824

This variable is unused.

- table_type

System Variable	**Name**	table_type
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	enumeration

This variable is a synonym for storage_engine, which is the preferred name; table_type is deprecated and is removed in MySQL 5.5.

- thread_cache_size

Command-Line Format	--thread_cache_size=#	
System Variable	**Name**	thread_cache_size
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	0
	Min Value	0
	Max Value	16384

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than thread_cache_size threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second you should normally set thread_cache_size high enough so that most new connections use cached threads. By

This documentation is for an older version. If you're
This documentation is for an older version. If you're

examining the difference between the `Connections` and `Threads_created` status variables, you can see how efficient the thread cache is. For details, see Section 5.1.6, “Server Status Variables”.

- `thread_concurrency`

Command-Line Format	--thread_concurrency=#	
System Variable	Name	thread_concurrency
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	integer
	Default	10
	Min Value	1
	Max Value	512

This variable is specific to Solaris systems, for which `mysqld` invokes the `thr_setconcurrency()` with the variable value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time.

- `thread_stack`

Command-Line Format	--thread_stack=#	
System Variable	Name	thread_stack
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (32-bit platforms)	Type	integer
	Default	196608
	Min Value	131072
	Max Value	4294967295
	Block Size	1024
Permitted Values (64-bit platforms)	Type	integer
	Default	262144
	Min Value	131072
	Max Value	18446744073709547520
	Block Size	1024

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. See Section 8.13.2, “The MySQL Benchmark Suite”. The default of 192KB (256KB for 64-bit systems) is large enough for normal operation. If the thread stack size is too small, it limits the complexity of the SQL statements that the server can handle, the recursion depth of stored procedures, and other memory-consuming actions.

- `time_format`

  This variable is unused.

- `time_zone`

System Variable	Name	`time_zone`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	Type	`string`

The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is `'SYSTEM'` (which means, “use the value of `system_time_zone`”). The value can be specified explicitly at server startup with the `--default-time-zone` option. See Section 10.6, “MySQL Server Time Zone Support”.

- `timed_mutexes`

Introduced	5.0.3	
Command-Line Format	`--timed_mutexes`	
System Variable	Name	`timed_mutexes`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`boolean`
	Default	`OFF`

This variable controls whether `InnoDB` mutexes are timed. If this variable is set to 0 or `OFF` (the default), mutex timing is disabled. If the variable is set to 1 or `ON`, mutex timing is enabled. With timing enabled, the `os_wait_times` value in the output from `SHOW ENGINE INNODB MUTEX` indicates the amount of time (in ms) spent in operating system waits. Otherwise, the value is 0. This variable was added in MySQL 5.0.3.

- `timestamp`

System Variable	Name	`timestamp`
	Variable Scope	Session
	Dynamic Variable	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	Type	numeric

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. *`timestamp_value`* should be a Unix epoch timestamp (a value like that returned by `UNIX_TIMESTAMP()`, not a value in `'YYYY-MM-DD hh:mm:ss'` format) or `DEFAULT`.

Setting `timestamp` to a constant value causes it to retain that value until it is changed again. Setting `timestamp` to `DEFAULT` causes its value to be the current date and time as of the time it is accessed.

`SET timestamp` affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. The server can be started with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`, in which case `SET timestamp` affects both functions.

- `tmp_table_size`

Command-Line Format		`--tmp_table_size=#`
System Variable	**Name**	`tmp_table_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (<= 5.0.85)	**Type**	`integer`
	Default	`33554432`
	Min Value	`1024`
	Max Value	`4294967295`
Permitted Values (>= 5.0.86)	**Type**	`integer`
	Default	`33554432`
	Min Value	`1024`
	Max Value	`9223372036854775807`

The maximum size of internal in-memory temporary tables. This variable does not apply to user-created `MEMORY` tables.

The actual limit is determined as the minimum of `tmp_table_size` and `max_heap_table_size`. If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk `MyISAM` table. Increase the value of `tmp_table_size` (and `max_heap_table_size` if necessary) if you do many advanced `GROUP BY` queries and you have lots of memory.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `tmpdir`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command-Line Format	--tmpdir=dir_name	
System Variable	Name	tmpdir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	directory name

The directory used for temporary files and temporary tables. This variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (“:”) on Unix and semicolon characters (“;”) on Windows, NetWare, and OS/2.

The multiple-directory feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set tmpdir to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or LOAD DATA INFILE operations. If files in the temporary file directory are lost when the server restarts, replication fails. You can set the slave's temporary directory using the slave_load_tmpdir variable. In that case, the slave will not use the general tmpdir value and you can set tmpdir to a nonpermanent location.

- transaction_alloc_block_size

Command-Line Format	--transaction_alloc_block_size=#	
System Variable	Name	transaction_alloc_block_size
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	8192
	Min Value	1024
	Max Value	4294967295
	Block Size	1024
Permitted Values (64-bit platforms)	Type	integer
	Default	8192
	Min Value	1024
	Max Value	18446744073709547520
	Block Size	1024

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size`.

- `transaction_prealloc_size`

Command-Line Format	`--transaction_prealloc_size=#`	
System Variable	**Name**	`transaction_prealloc_size`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`4096`
	Min Value	`1024`
	Max Value	`4294967295`
	Block Size	`1024`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`4096`
	Min Value	`1024`
	Max Value	`18446744073709547520`
	Block Size	`1024`

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size`. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` bytes.

By making `transaction_prealloc_size` sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `tx_isolation`

System Variable	**Name**	`tx_isolation`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`enumeration`
	Default	`REPEATABLE-READ`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Valid Values	READ-UNCOMMITTED
		READ-COMMITTED
		REPEATABLE-READ
		SERIALIZABLE

The default transaction isolation level. Defaults to REPEATABLE-READ.

This variable can be set directly, or indirectly using the SET TRANSACTION statement. See Section 13.3.6, “SET TRANSACTION Syntax”. If you set tx_isolation directly to an isolation level name that contains a space, the name should be enclosed within quotation marks, with the space replaced by a dash. For example:

```
SET tx_isolation = 'READ-COMMITTED';
```

Any unique prefix of a valid value may be used to set the value of this variable.

The default transaction isolation level can also be set at startup using the --transaction-isolation server option.

- unique_checks

System Variable	**Name**	unique_checks
	Variable Scope	Session
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	1

If set to 1 (the default), uniqueness checks for secondary indexes in InnoDB tables are performed. If set to 0, storage engines are permitted to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to InnoDB.

Setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still permitted to check for them and issue duplicate-key errors if it detects them.

- updatable_views_with_limit

Introduced	5.0.2	
Command-Line Format	--updatable_views_with_limit=#	
System Variable	**Name**	updatable_views_with_limit
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	1

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key defined in the underlying table, if the update statement contains a `LIMIT` clause. (Such updates often are generated by GUI tools.) An update is an `UPDATE` or `DELETE` statement. Primary key here means a `PRIMARY KEY`, or a `UNIQUE` index in which no column can contain `NULL`.

The variable can have two values:

- `1` or `YES`: Issue a warning only (not an error message). This is the default value.
- `0` or `NO`: Prohibit the update.

This variable was added in MySQL 5.0.2.

- `version`

System Variable	Name	`version`
	Variable Scope	Global
	Dynamic Variable	No

The version number for the server. The value might also include a suffix indicating server build or configuration information. `-log` indicates that one or more of the general log, slow query log, or binary log are enabled. `-debug` indicates that the server was built with debugging support enabled.

System Variable	Name	`version`
	Variable Scope	Global
	Dynamic Variable	No

Starting with MySQL 5.0.24, the version number will also indicate whether the server is a standard release (Community) or Enterprise release (for example, `5.0.28-enterprise-gpl-nt`).

- `version_bdb`

The `BDB` storage engine version.

- `version_comment`

System Variable	Name	`version_comment`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`string`

The `configure` script has a `--with-comment` option that permits a comment to be specified when building MySQL. This variable contains the value of that comment.

For precompiled binaries, this variable will hold the server version and license information. Starting with MySQL 5.0.24, `version_comment` will include the full server type and license. For community users

This documentation is for an older version. If you're
This documentation is for an older version. If you're

this will appear as `MySQL Community Edition - Standard (GPL)`. For Enterprise users, the version might be displayed as `MySQL Enterprise Server (GPL)`. The corresponding license for your MySQL binary is shown in parentheses. For server compiled from source, the default value will be the same as that for Community releases.

- `version_compile_machine`

System Variable	Name	version_compile_machine
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string

The type of machine or architecture on which MySQL was built.

- `version_compile_os`

System Variable	Name	version_compile_os
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	string

The type of operating system on which MySQL was built.

- `wait_timeout`

Command-Line Format		--wait_timeout=#
System Variable	Name	wait_timeout
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (Windows)	Type	integer
	Default	28800
	Min Value	1
	Max Value	2147483
Permitted Values (Other)	Type	integer
	Default	28800
	Min Value	1
	Max Value	31536000

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of seconds the server waits for activity on a noninteractive connection before closing it. This timeout applies only to TCP/IP and Unix socket file connections, not to connections made using named pipes, or shared memory.

On thread startup, the session `wait_timeout` value is initialized from the global `wait_timeout` value or from the global `interactive_timeout` value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout`.

- `warning_count`

  The number of errors, warnings, and notes that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.5.37, “SHOW WARNINGS Syntax”.

5.1.5 Using System Variables

The MySQL server maintains many system variables that indicate how it is configured. Section 5.1.4, “Server System Variables”, describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

The server maintains two kinds of system variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified on the command line or in an option file. (See Section 4.2.3, “Specifying Program Options”.)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, the client's SQL mode is controlled by the session `sql_mode` value, which is initialized when the client connects to the value of the global `sql_mode` value.

System variable values can be set globally at server startup by using options on the command line or in an option file. When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with a query cache size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; `16M` and `16m` are equivalent, as are `1G` and `1g`.

If you want to restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, you can specify this maximum by using an option of the form `--maximum-var_name=value`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

at server startup. For example, to prevent the value of `query_cache_size` from being increased to more than 32MB at runtime, use the option `--maximum-query_cache_size=32M`.

Many system variables are dynamic and can be changed while the server runs by using the `SET` statement. For a list, see Section 5.1.5.2, “Dynamic System Variables”. To change a system variable with `SET`, refer to it as *`var_name`*, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by `GLOBAL` or `@@global.`. The `SUPER` privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by `SESSION`, `@@session.`, or `@@`. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- `LOCAL` and `@@local.` are synonyms for `SESSION` and `@@session.`.
- If no modifier is present, `SET` changes the session variable.

A `SET` statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent `GLOBAL` or `SESSION` modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The *`@@var_name`* syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the `SET GLOBAL` statement).

To prevent incorrect usage, MySQL produces an error if you use `SET GLOBAL` with a variable that can only be used with `SET SESSION` or if you do not specify `GLOBAL` (or `@@global.`) when setting a global variable.

To set a `SESSION` variable to the `GLOBAL` value or a `GLOBAL` value to the compiled-in MySQL default value, use the `DEFAULT` keyword. For example, the following two statements are identical in setting the session value of `max_join_size` to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to `DEFAULT`. In such cases, use of `DEFAULT` results in an error.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can refer to the values of specific global or session system variables in expressions by using one of the `@@`-modifiers. For example, you can retrieve values in a `SELECT` statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as `@@var_name` (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)

Note

Some variables displayed by `SHOW VARIABLES` may not be available using `SELECT @@var_name` syntax; an `Unknown system variable` occurs. As a workaround in such cases, you can use `SHOW VARIABLES LIKE 'var_name'`.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Note

Some system variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` works but `--delay_key_write=ON` does not.

To display system variable names and values, use the `SHOW VARIABLES` statement:

```
mysql> SHOW VARIABLES;
+--------+-----------------------------------------------------------------+
| Variable_name                   | Value                                 |
+--------+-----------------------------------------------------------------+
| auto_increment_increment        | 1                                     |
| auto_increment_offset           | 1                                     |
| automatic_sp_privileges         | ON                                    |
| back_log                        | 50                                    |
| basedir                         | /                                     |
| bdb_cache_size                  | 8388600                               |
| bdb_home                        | /var/lib/mysql/                       |
| bdb_log_buffer_size             | 32768                                 |
| bdb_logdir                      |                                       |
| bdb_max_lock                    | 10000                                 |
| bdb_shared_data                 | OFF                                   |
| bdb_tmpdir                      | /tmp/                                 |
| binlog_cache_size               | 32768                                 |
| bulk_insert_buffer_size         | 8388608                               |
| character_set_client            | latin1                                |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| character_set_connection        | latin1                              |
| character_set_database          | latin1                              |
| character_set_results           | latin1                              |
| character_set_server            | latin1                              |
| character_set_system            | utf8                                |
| character_sets_dir              | /usr/share/mysql/charsets/          |
| collation_connection            | latin1_swedish_ci                   |
| collation_database              | latin1_swedish_ci                   |
| collation_server                | latin1_swedish_ci                   |
...
| innodb_additional_mem_pool_size | 1048576                             |
| innodb_autoextend_increment     | 8                                   |
| innodb_buffer_pool_awe_mem_mb   | 0                                   |
| innodb_buffer_pool_size         | 8388608                             |
| innodb_checksums                | ON                                  |
| innodb_commit_concurrency       | 0                                   |
| innodb_concurrency_tickets      | 500                                 |
| innodb_data_file_path           | ibdata1:10M:autoextend              |
| innodb_data_home_dir            |                                     |
...
| version                         | 5.0.19                              |
| version_comment                 | MySQL Community Edition - (GPL)     |
| version_compile_machine         | i686                                |
| version_compile_os              | pc-linux-gnu                        |
| wait_timeout                    | 28800                               |
+--------+-----------------------------------------------------------+
```

With a `LIKE` clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the "`%`" wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because "`_`" is a wildcard that matches any single character, you should escape it as "`\_`" to match it literally. In practice, this is rarely necessary.

For `SHOW VARIABLES`, if you specify neither `GLOBAL` nor `SESSION`, MySQL returns `SESSION` values.

The reason for requiring the `GLOBAL` keyword when setting `GLOBAL`-only variables but not when retrieving them is to prevent problems in the future. If we were to remove a `SESSION` variable that has the same name as a `GLOBAL` variable, a client with the `SUPER` privilege might accidentally change the `GLOBAL` variable rather than just the `SESSION` variable for its own connection. If we add a `SESSION` variable with the same name as a `GLOBAL` variable, a client that intends to change the `GLOBAL` variable might find only its own `SESSION` variable changed.

5.1.5.1 Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in Section 8.10.1, “The MyISAM Key Cache”.

To refer to a component of a structured variable instance, you can use a compound name in *`instance_name.component_name`* format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without any instance name, the `default` instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but `` `hot-cache` `` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@global.`*`var_name`* for referring to nonstructured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
[mysqld]
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.1.5.2 Dynamic System Variables

Many server system variables are dynamic and can be set at runtime using `SET GLOBAL` or `SET SESSION`. You can also obtain their values using `SELECT`. See Section 5.1.5, “Using System Variables”.

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether `GLOBAL` or `SESSION` (or both) apply. The table also lists session options that can be set with the `SET` statement. Section 5.1.4, “Server System Variables”, discusses these options.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, `ON` or `OFF`. (If you set them on the command line or in an option file, use the numeric values.) Variables that are marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration

This documentation is for an older version. If you're
This documentation is for an older version. If you're

value corresponds to 0. This differs from ENUM columns, for which the first enumeration value corresponds to 1.

Table 5.3 Dynamic Variable Summary

Variable Name	Variable Type	Variable Scope
auto_increment_increment	integer	GLOBAL | SESSION
auto_increment_offset	integer	GLOBAL | SESSION
autocommit	boolean	SESSION
automatic_sp_privileges	boolean	GLOBAL
big_tables	boolean	SESSION
binlog_cache_size	integer	GLOBAL
bulk_insert_buffer_size	integer	GLOBAL | SESSION
character_set_client	string	GLOBAL | SESSION
character_set_connection	string	GLOBAL | SESSION
character_set_database	string	GLOBAL | SESSION
character_set_filesystem	string	GLOBAL | SESSION
character_set_results	string	GLOBAL | SESSION
character_set_server	string	GLOBAL | SESSION
collation_connection	string	GLOBAL | SESSION
collation_database	string	GLOBAL | SESSION
collation_server	string	GLOBAL | SESSION
completion_type	integer	GLOBAL | SESSION
concurrent_insert	integer	GLOBAL
connect_timeout	integer	GLOBAL
debug	string	GLOBAL | SESSION
storage_engine	enumeration	GLOBAL | SESSION
default_week_format	integer	GLOBAL | SESSION
delay_key_write	enumeration	GLOBAL
delayed_insert_limit	integer	GLOBAL
delayed_insert_timeout	integer	GLOBAL
delayed_queue_size	integer	GLOBAL
div_precision_increment	integer	GLOBAL | SESSION
engine_condition_pushdown	boolean	GLOBAL | SESSION
expire_logs_days	integer	GLOBAL
flush	boolean	GLOBAL
flush_time	integer	GLOBAL
foreign_key_checks	boolean	SESSION
ft_boolean_syntax	string	GLOBAL
group_concat_max_len	integer	GLOBAL | SESSION
identity	integer	SESSION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
init_connect	string	GLOBAL
init_slave	string	GLOBAL
innodb_autoextend_increment	integer	GLOBAL
innodb_commit_concurrency	integer	GLOBAL
innodb_concurrency_tickets	integer	GLOBAL
innodb_fast_shutdown	integer	GLOBAL
innodb_flush_log_at_trx_commit	enumeration	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_purge_lag	integer	GLOBAL
innodb_support_xa	boolean	GLOBAL | SESSION
innodb_sync_spin_loops	integer	GLOBAL
innodb_table_locks	boolean	GLOBAL | SESSION
innodb_thread_concurrency	integer	GLOBAL
innodb_thread_sleep_delay	integer	GLOBAL
innodb_use_legacy_cardinality_algorithm	boolean	GLOBAL
insert_id	integer	SESSION
interactive_timeout	integer	GLOBAL | SESSION
join_buffer_size	integer	GLOBAL | SESSION
keep_files_on_create	boolean	GLOBAL | SESSION
key_buffer_size	integer	GLOBAL
key_cache_age_threshold	integer	GLOBAL
key_cache_block_size	integer	GLOBAL
key_cache_division_limit	integer	GLOBAL
last_insert_id	integer	SESSION
lc_time_names	string	GLOBAL | SESSION
local_infile	boolean	GLOBAL
log_bin_trust_function_creators	boolean	GLOBAL
log_bin_trust_routine_creators	boolean	GLOBAL
log_queries_not_using_indexes	boolean	GLOBAL
log_warnings	integer	GLOBAL | SESSION
long_query_time	numeric	GLOBAL | SESSION
low_priority_updates	boolean	GLOBAL | SESSION
max_allowed_packet	integer	GLOBAL | SESSION
max_binlog_cache_size	integer	GLOBAL
max_binlog_size	integer	GLOBAL
max_connect_errors	integer	GLOBAL
max_connections	integer	GLOBAL
max_delayed_threads	integer	GLOBAL | SESSION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
max_error_count	integer	GLOBAL | SESSION
max_heap_table_size	integer	GLOBAL | SESSION
max_insert_delayed_threads	integer	GLOBAL | SESSION
max_join_size	integer	GLOBAL | SESSION
max_length_for_sort_data	integer	GLOBAL | SESSION
max_prepared_stmt_count	integer	GLOBAL
max_relay_log_size	integer	GLOBAL
max_seeks_for_key	integer	GLOBAL | SESSION
max_sort_length	integer	GLOBAL | SESSION
max_sp_recursion_depth	integer	GLOBAL | SESSION
max_tmp_tables	integer	GLOBAL | SESSION
max_user_connections	integer	GLOBAL | SESSION
max_write_lock_count	integer	GLOBAL
multi_range_count	integer	GLOBAL | SESSION
myisam_data_pointer_size	integer	GLOBAL
myisam_max_sort_file_size	integer	GLOBAL
myisam_repair_threads	integer	GLOBAL | SESSION
myisam_sort_buffer_size	integer	GLOBAL | SESSION
myisam_stats_method	enumeration	GLOBAL | SESSION
ndb_autoincrement_prefetch_sz	integer	GLOBAL | SESSION
ndb_cache_check_time	integer	GLOBAL
ndb_force_send	boolean	GLOBAL | SESSION
ndb_index_stat_cache_entries	integer	GLOBAL | SESSION
ndb_index_stat_enable	boolean	GLOBAL | SESSION
ndb_index_stat_update_freq	integer	GLOBAL | SESSION
ndb_use_exact_count	boolean	GLOBAL | SESSION
ndb_use_transactions	boolean	GLOBAL | SESSION
net_buffer_length	integer	GLOBAL | SESSION
net_read_timeout	integer	GLOBAL | SESSION
net_retry_count	integer	GLOBAL | SESSION
net_write_timeout	integer	GLOBAL | SESSION
new	boolean	GLOBAL | SESSION
old_passwords	boolean	GLOBAL | SESSION
optimizer_prune_level	boolean	GLOBAL | SESSION
optimizer_search_depth	integer	GLOBAL | SESSION
preload_buffer_size	integer	GLOBAL | SESSION
profiling	boolean	SESSION
profiling_history_size	integer	GLOBAL | SESSION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
pseudo_thread_id	integer	SESSION
query_alloc_block_size	integer	GLOBAL | SESSION
query_cache_limit	integer	GLOBAL
query_cache_min_res_unit	integer	GLOBAL
query_cache_size	integer	GLOBAL
query_cache_type	enumeration	GLOBAL | SESSION
query_cache_wlock_invalidate	boolean	GLOBAL | SESSION
query_prealloc_size	integer	GLOBAL | SESSION
rand_seed1	integer	SESSION
rand_seed2	integer	SESSION
range_alloc_block_size	integer	GLOBAL | SESSION
read_buffer_size	integer	GLOBAL | SESSION
read_only	boolean	GLOBAL
read_rnd_buffer_size	integer	GLOBAL | SESSION
relay_log_purge	boolean	GLOBAL
rpl_recovery_rank	integer	GLOBAL
secure_auth	boolean	GLOBAL
server_id	integer	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	integer	GLOBAL
slave_transaction_retries	integer	GLOBAL
slow_launch_time	integer	GLOBAL
sort_buffer_size	integer	GLOBAL | SESSION
sql_auto_is_null	boolean	SESSION
sql_big_selects	boolean	SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION
sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL | SESSION
sql_max_join_size	integer	GLOBAL | SESSION
sql_mode	set	GLOBAL | SESSION
sql_notes	boolean	SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	integer	GLOBAL | SESSION
sql_slave_skip_counter	integer	GLOBAL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
sql_warnings	boolean	SESSION
storage_engine	enumeration	GLOBAL | SESSION
sync_binlog	integer	GLOBAL
sync_frm	boolean	GLOBAL
table_cache	integer	GLOBAL
table_lock_wait_timeout	integer	GLOBAL
table_type	enumeration	GLOBAL | SESSION
thread_cache_size	integer	GLOBAL
time_zone	string	GLOBAL | SESSION
timed_mutexes	boolean	GLOBAL
timestamp	numeric	SESSION
tmp_table_size	integer	GLOBAL | SESSION
transaction_alloc_block_size	integer	GLOBAL | SESSION
transaction_prealloc_size	integer	GLOBAL | SESSION
tx_isolation	enumeration	GLOBAL | SESSION
unique_checks	boolean	SESSION
updatable_views_with_limit	boolean	GLOBAL | SESSION
wait_timeout	integer	GLOBAL | SESSION

5.1.6 Server Status Variables

The MySQL server maintains many status variables that provide information about its operation. You can view these variables and their values by using the SHOW [GLOBAL | SESSION] STATUS statement (see Section 13.7.5.32, "SHOW STATUS Syntax"). The optional GLOBAL keyword aggregates the values over all connections, and SESSION shows the values for the current connection.

```
mysql> SHOW GLOBAL STATUS;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Bytes_received                    | 155372598  |
| Bytes_sent                        | 1176560426 |
...
| Connections                       | 30023      |
| Created_tmp_disk_tables           | 0          |
| Created_tmp_files                 | 3          |
| Created_tmp_tables                | 2          |
...
| Threads_created                   | 217        |
| Threads_running                   | 88         |
| Uptime                            | 1389872    |
+-----------------------------------+------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

Before MySQL 5.0.2, `SHOW STATUS` returned global status values. Because the default as of 5.0.2 is to return session values, this is incompatible with previous versions. To issue a `SHOW STATUS` statement that will retrieve global status values for all versions of MySQL, write it like this:

```
SHOW /*!50002 GLOBAL */ STATUS;
```

Several status variables provide statement counts. To determine the number of statements executed, use these relationships:

```
  SUM(Com_xxx) + Qcache_hits
= Questions + statements executed within stored programs
= Queries
```

Many status variables are reset to 0 by the `FLUSH STATUS` statement.

The following table lists all available server status variables:

Table 5.4 Status Variable Summary

Variable Name	Variable Type	Variable Scope
`Aborted_clients`	integer	`GLOBAL`
`Aborted_connects`	integer	`GLOBAL`
`Binlog_cache_disk_use`	integer	`GLOBAL`
`Binlog_cache_use`	integer	`GLOBAL`
`Bytes_received`	integer	`GLOBAL` | `SESSION`
`Bytes_sent`	integer	`GLOBAL` | `SESSION`
`Com_admin_commands`	integer	`GLOBAL` | `SESSION`
`Com_alter_db`	integer	`GLOBAL` | `SESSION`
`Com_alter_event`	integer	`GLOBAL` | `SESSION`
`Com_alter_table`	integer	`GLOBAL` | `SESSION`
`Com_analyze`	integer	`GLOBAL` | `SESSION`
`Com_backup_table`	integer	`GLOBAL` | `SESSION`
`Com_begin`	integer	`GLOBAL` | `SESSION`
`Com_call_procedure`	integer	`GLOBAL` | `SESSION`
`Com_change_db`	integer	`GLOBAL` | `SESSION`
`Com_change_master`	integer	`GLOBAL` | `SESSION`
`Com_check`	integer	`GLOBAL` | `SESSION`
`Com_checksum`	integer	`GLOBAL` | `SESSION`
`Com_commit`	integer	`GLOBAL` | `SESSION`
`Com_create_db`	integer	`GLOBAL` | `SESSION`
`Com_create_event`	integer	`GLOBAL` | `SESSION`
`Com_create_function`	integer	`GLOBAL` | `SESSION`
`Com_create_index`	integer	`GLOBAL` | `SESSION`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Com_create_table	integer	GLOBAL | SESSION
Com_create_user	integer	GLOBAL | SESSION
Com_dealloc_sql	integer	GLOBAL | SESSION
Com_delete	integer	GLOBAL | SESSION
Com_delete_multi	integer	GLOBAL | SESSION
Com_do	integer	GLOBAL | SESSION
Com_drop_db	integer	GLOBAL | SESSION
Com_drop_event	integer	GLOBAL | SESSION
Com_drop_function	integer	GLOBAL | SESSION
Com_drop_index	integer	GLOBAL | SESSION
Com_drop_table	integer	GLOBAL | SESSION
Com_drop_user	integer	GLOBAL | SESSION
Com_execute_sql	integer	GLOBAL | SESSION
Com_flush	integer	GLOBAL | SESSION
Com_grant	integer	GLOBAL | SESSION
Com_ha_close	integer	GLOBAL | SESSION
Com_ha_open	integer	GLOBAL | SESSION
Com_ha_read	integer	GLOBAL | SESSION
Com_help	integer	GLOBAL | SESSION
Com_insert	integer	GLOBAL | SESSION
Com_insert_select	integer	GLOBAL | SESSION
Com_kill	integer	GLOBAL | SESSION
Com_load	integer	GLOBAL | SESSION
Com_lock_tables	integer	GLOBAL | SESSION
Com_optimize	integer	GLOBAL | SESSION
Com_preload_keys	integer	GLOBAL | SESSION
Com_prepare_sql	integer	GLOBAL | SESSION
Com_purge	integer	GLOBAL | SESSION
Com_purge_before_date	integer	GLOBAL | SESSION
Com_rename_table	integer	GLOBAL | SESSION
Com_repair	integer	GLOBAL | SESSION
Com_replace	integer	GLOBAL | SESSION
Com_replace_select	integer	GLOBAL | SESSION
Com_reset	integer	GLOBAL | SESSION
Com_restore_table	integer	GLOBAL | SESSION
Com_revoke	integer	GLOBAL | SESSION
Com_revoke_all	integer	GLOBAL | SESSION
Com_rollback	integer	GLOBAL | SESSION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Com_savepoint	integer	GLOBAL | SESSION
Com_select	integer	GLOBAL | SESSION
Com_set_option	integer	GLOBAL | SESSION
Com_show_binlog_events	integer	GLOBAL | SESSION
Com_show_binlogs	integer	GLOBAL | SESSION
Com_show_charsets	integer	GLOBAL | SESSION
Com_show_collations	integer	GLOBAL | SESSION
Com_show_column_types	integer	GLOBAL | SESSION
Com_show_create_db	integer	GLOBAL | SESSION
Com_show_create_event	integer	GLOBAL | SESSION
Com_show_create_table	integer	GLOBAL | SESSION
Com_show_databases	integer	GLOBAL | SESSION
Com_show_engine_logs	integer	GLOBAL | SESSION
Com_show_engine_mutex	integer	GLOBAL | SESSION
Com_show_engine_status	integer	GLOBAL | SESSION
Com_show_errors	integer	GLOBAL | SESSION
Com_show_events	integer	GLOBAL | SESSION
Com_show_fields	integer	GLOBAL | SESSION
Com_show_grants	integer	GLOBAL | SESSION
Com_show_innodb_status	integer	GLOBAL | SESSION
Com_show_keys	integer	GLOBAL | SESSION
Com_show_logs	integer	GLOBAL | SESSION
Com_show_master_status	integer	GLOBAL | SESSION
Com_show_ndb_status	integer	GLOBAL | SESSION
Com_show_new_master	integer	GLOBAL | SESSION
Com_show_open_tables	integer	GLOBAL | SESSION
Com_show_plugins	integer	GLOBAL | SESSION
Com_show_privileges	integer	GLOBAL | SESSION
Com_show_processlist	integer	GLOBAL | SESSION
Com_show_slave_hosts	integer	GLOBAL | SESSION
Com_show_slave_status	integer	GLOBAL | SESSION
Com_show_status	integer	GLOBAL | SESSION
Com_show_storage_engines	integer	GLOBAL | SESSION
Com_show_tables	integer	GLOBAL | SESSION
Com_show_triggers	integer	GLOBAL | SESSION
Com_show_variables	integer	GLOBAL | SESSION
Com_show_warnings	integer	GLOBAL | SESSION
Com_slave_start	integer	GLOBAL | SESSION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Com_slave_stop	integer	GLOBAL | SESSION
Com_stmt_close	integer	GLOBAL | SESSION
Com_stmt_execute	integer	GLOBAL | SESSION
Com_stmt_fetch	integer	GLOBAL | SESSION
Com_stmt_prepare	integer	GLOBAL | SESSION
Com_stmt_reset	integer	GLOBAL | SESSION
Com_stmt_send_long_data	integer	GLOBAL | SESSION
Com_truncate	integer	GLOBAL | SESSION
Com_unlock_tables	integer	GLOBAL | SESSION
Com_update	integer	GLOBAL | SESSION
Com_update_multi	integer	GLOBAL | SESSION
Com_xa_commit	integer	GLOBAL | SESSION
Com_xa_end	integer	GLOBAL | SESSION
Com_xa_prepare	integer	GLOBAL | SESSION
Com_xa_recover	integer	GLOBAL | SESSION
Com_xa_rollback	integer	GLOBAL | SESSION
Com_xa_start	integer	GLOBAL | SESSION
Compression	integer	SESSION
Connections	integer	GLOBAL
Created_tmp_disk_tables	integer	GLOBAL | SESSION
Created_tmp_files	integer	GLOBAL
Created_tmp_tables	integer	GLOBAL | SESSION
Delayed_errors	integer	GLOBAL
Delayed_insert_threads	integer	GLOBAL
Delayed_writes	integer	GLOBAL
Flush_commands	integer	GLOBAL
Handler_commit	integer	GLOBAL | SESSION
Handler_delete	integer	GLOBAL | SESSION
Handler_discover	integer	GLOBAL | SESSION
Handler_prepare	integer	GLOBAL | SESSION
Handler_read_first	integer	GLOBAL | SESSION
Handler_read_key	integer	GLOBAL | SESSION
Handler_read_next	integer	GLOBAL | SESSION
Handler_read_prev	integer	GLOBAL | SESSION
Handler_read_rnd	integer	GLOBAL | SESSION
Handler_read_rnd_next	integer	GLOBAL | SESSION
Handler_rollback	integer	GLOBAL | SESSION
Handler_savepoint	integer	GLOBAL | SESSION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Handler_savepoint_rollback	integer	GLOBAL | SESSION
Handler_update	integer	GLOBAL | SESSION
Handler_write	integer	GLOBAL | SESSION
Innodb_buffer_pool_pages_data	integer	GLOBAL
Innodb_buffer_pool_pages_dirty	integer	GLOBAL
Innodb_buffer_pool_pages_flushed	integer	GLOBAL
Innodb_buffer_pool_pages_free	integer	GLOBAL
Innodb_buffer_pool_pages_latched	integer	GLOBAL
Innodb_buffer_pool_pages_misc	integer	GLOBAL
Innodb_buffer_pool_pages_total	integer	GLOBAL
Innodb_buffer_pool_read_ahead_rnd	integer	GLOBAL
Innodb_buffer_pool_read_ahead_seq	integer	GLOBAL
Innodb_buffer_pool_read_requests	integer	GLOBAL
Innodb_buffer_pool_reads	integer	GLOBAL
Innodb_buffer_pool_wait_free	integer	GLOBAL
Innodb_buffer_pool_write_requests	integer	GLOBAL
Innodb_data_fsyncs	integer	GLOBAL
Innodb_data_pending_fsyncs	integer	GLOBAL
Innodb_data_pending_reads	integer	GLOBAL
Innodb_data_pending_writes	integer	GLOBAL
Innodb_data_read	integer	GLOBAL
Innodb_data_reads	integer	GLOBAL
Innodb_data_writes	integer	GLOBAL
Innodb_data_written	integer	GLOBAL
Innodb_dblwr_pages_written	integer	GLOBAL
Innodb_dblwr_writes	integer	GLOBAL
Innodb_log_waits	integer	GLOBAL
Innodb_log_write_requests	integer	GLOBAL
Innodb_log_writes	integer	GLOBAL
Innodb_os_log_fsyncs	integer	GLOBAL
Innodb_os_log_pending_fsyncs	integer	GLOBAL
Innodb_os_log_pending_writes	integer	GLOBAL
Innodb_os_log_written	integer	GLOBAL
Innodb_page_size	integer	GLOBAL
Innodb_pages_created	integer	GLOBAL
Innodb_pages_read	integer	GLOBAL
Innodb_pages_written	integer	GLOBAL
Innodb_row_lock_current_waits	integer	GLOBAL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Innodb_row_lock_time	integer	GLOBAL
Innodb_row_lock_time_avg	integer	GLOBAL
Innodb_row_lock_time_max	integer	GLOBAL
Innodb_row_lock_waits	integer	GLOBAL
Innodb_rows_deleted	integer	GLOBAL
Innodb_rows_inserted	integer	GLOBAL
Innodb_rows_read	integer	GLOBAL
Innodb_rows_updated	integer	GLOBAL
Key_blocks_not_flushed	integer	GLOBAL
Key_blocks_unused	integer	GLOBAL
Key_blocks_used	integer	GLOBAL
Key_read_requests	integer	GLOBAL
Key_reads	integer	GLOBAL
Key_write_requests	integer	GLOBAL
Key_writes	integer	GLOBAL
Last_query_cost	numeric	SESSION
Max_used_connections	integer	GLOBAL
Ndb_cluster_node_id	integer	GLOBAL | SESSION
Ndb_config_from_host	integer	GLOBAL | SESSION
Ndb_config_from_port	integer	GLOBAL | SESSION
Ndb_cluster_node_id	integer	GLOBAL
Not_flushed_delayed_rows	integer	GLOBAL
Open_files	integer	GLOBAL
Open_streams	integer	GLOBAL
Open_tables	integer	GLOBAL | SESSION
Opened_tables	integer	GLOBAL | SESSION
Prepared_stmt_count	integer	GLOBAL
Qcache_free_blocks	integer	GLOBAL
Qcache_free_memory	integer	GLOBAL
Qcache_hits	integer	GLOBAL
Qcache_inserts	integer	GLOBAL
Qcache_lowmem_prunes	integer	GLOBAL
Qcache_not_cached	integer	GLOBAL
Qcache_queries_in_cache	integer	GLOBAL
Qcache_total_blocks	integer	GLOBAL
Queries	integer	GLOBAL | SESSION
Questions	integer	GLOBAL | SESSION
Rpl_status	string	GLOBAL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Select_full_join	integer	GLOBAL | SESSION
Select_full_range_join	integer	GLOBAL | SESSION
Select_range	integer	GLOBAL | SESSION
Select_range_check	integer	GLOBAL | SESSION
Select_scan	integer	GLOBAL | SESSION
Slave_open_temp_tables	integer	GLOBAL
Slave_retried_transactions	integer	GLOBAL
Slave_running	boolean	GLOBAL
Slow_launch_threads	integer	GLOBAL | SESSION
Slow_queries	integer	GLOBAL | SESSION
Sort_merge_passes	integer	GLOBAL | SESSION
Sort_range	integer	GLOBAL | SESSION
Sort_rows	integer	GLOBAL | SESSION
Sort_scan	integer	GLOBAL | SESSION
Ssl_accept_renegotiates	integer	GLOBAL
Ssl_accepts	integer	GLOBAL
Ssl_callback_cache_hits	integer	GLOBAL
Ssl_cipher	string	GLOBAL | SESSION
Ssl_cipher_list	string	GLOBAL | SESSION
Ssl_client_connects	integer	GLOBAL
Ssl_connect_renegotiates	integer	GLOBAL
Ssl_ctx_verify_depth	integer	GLOBAL
Ssl_ctx_verify_mode	integer	GLOBAL
Ssl_default_timeout	integer	GLOBAL | SESSION
Ssl_finished_accepts	integer	GLOBAL
Ssl_finished_connects	integer	GLOBAL
Ssl_session_cache_hits	integer	GLOBAL
Ssl_session_cache_misses	integer	GLOBAL
Ssl_session_cache_mode	string	GLOBAL
Ssl_session_cache_overflows	integer	GLOBAL
Ssl_session_cache_size	integer	GLOBAL
Ssl_session_cache_timeouts	integer	GLOBAL
Ssl_sessions_reused	integer	GLOBAL | SESSION
Ssl_used_session_cache_entries	integer	GLOBAL
Ssl_verify_depth	integer	GLOBAL | SESSION
Ssl_verify_mode	integer	GLOBAL | SESSION
Ssl_version	string	GLOBAL | SESSION
Table_locks_immediate	integer	GLOBAL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Variable Name	Variable Type	Variable Scope
Table_locks_waited	integer	GLOBAL
Tc_log_max_pages_used	integer	GLOBAL
Tc_log_page_size	integer	GLOBAL
Tc_log_page_waits	integer	GLOBAL
Threads_cached	integer	GLOBAL
Threads_connected	integer	GLOBAL
Threads_created	integer	GLOBAL
Threads_running	integer	GLOBAL
Uptime	integer	GLOBAL
Uptime_since_flush_status	integer	GLOBAL

The status variables have the following meanings. For meanings of status variables specific to MySQL Cluster, see MySQL Cluster Status Variables.

- Aborted_clients

  The number of connections that were aborted because the client died without closing the connection properly. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

- Aborted_connects

  The number of failed attempts to connect to the MySQL server. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

- Binlog_cache_disk_use

  The number of transactions that used the temporary binary log cache but that exceeded the value of binlog_cache_size and used a temporary file to store statements from the transaction.

- Binlog_cache_use

  The number of transactions that used the temporary binary log cache.

- Bytes_received

  The number of bytes received from all clients.

- Bytes_sent

  The number of bytes sent to all clients.

- Com_*xxx*

  The Com_*xxx* statement counter variables indicate the number of times each *xxx* statement has been executed. There is one status variable for each type of statement. For example, Com_delete and Com_update count DELETE and UPDATE statements, respectively. Com_delete_multi and Com_update_multi are similar but apply to DELETE and UPDATE statements that use multiple-table syntax.

  If a query result is returned from query cache, the server increments the Qcache_hits status variable, not Com_select. See Section 8.10.3.4, “Query Cache Status and Maintenance”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The discussion at the beginning of this section indicates how to relate these statement-counting status variables to other such variables.

All of the `Com_stmt_xxx` variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

The `Com_stmt_xxx` status variables were added in 5.0.8:

- `Com_stmt_prepare`
- `Com_stmt_execute`
- `Com_stmt_fetch`
- `Com_stmt_send_long_data`
- `Com_stmt_reset`
- `Com_stmt_close`

Those variables stand for prepared statement commands. Their names refer to the `COM_xxx` command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as `mysql_stmt_prepare()`, `mysql_stmt_execute()`, and so forth are executed. However, `Com_stmt_prepare`, `Com_stmt_execute` and `Com_stmt_close` also increase for `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE`, respectively. Additionally, the values of the older statement counter variables `Com_prepare_sql`, `Com_execute_sql`, and `Com_dealloc_sql` increase for the `PREPARE`, `EXECUTE`, and `DEALLOCATE PREPARE` statements. `Com_stmt_fetch` stands for the total number of network round-trips issued when fetching from cursors.

- `Compression`

  Whether the client connection uses compression in the client/server protocol. Added in MySQL 5.0.16.

- `Connections`

  The number of connection attempts (successful or not) to the MySQL server.

- `Created_tmp_disk_tables`

  The number of internal on-disk temporary tables created by the server while executing statements.

  If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` and `max_heap_table_size` values. If `Created_tmp_disk_tables` is large, you may want to increase the `tmp_table_size` or `max_heap_table_size` value to lessen the likelihood that internal temporary tables in memory will be converted to on-disk tables.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

  See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `Created_tmp_files`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

How many temporary files `mysqld` has created.

- `Created_tmp_tables`

  The number of internal temporary tables created by the server while executing statements.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

  See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `Delayed_errors`

  The number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).

- `Delayed_insert_threads`

  The number of `INSERT DELAYED` handler threads in use.

- `Delayed_writes`

  The number of `INSERT DELAYED` rows written.

- `Flush_commands`

  The number of times the server flushes tables, whether because a user executed a `FLUSH TABLES` statement or due to internal server operation. It is also incremented by receipt of a `COM_REFRESH` packet. This is in contrast to `Com_flush`, which indicates how many `FLUSH` statements have been executed, whether `FLUSH TABLES`, `FLUSH LOGS`, and so forth.

- `Handler_commit`

  The number of internal `COMMIT` statements.

- `Handler_delete`

  The number of times that rows have been deleted from tables.

- `Handler_prepare`

  A counter for the prepare phase of two-phase commit operations. Added in MySQL 5.0.3.

- `Handler_read_first`

  The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans; for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed.

- `Handler_read_key`

  The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_next`

  The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Handler_read_prev`

  The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

  The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- `Handler_read_rnd_next`

  The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

  The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`

  The number of requests for a storage engine to place a savepoint. Added in MySQL 5.0.3.

- `Handler_savepoint_rollback`

  The number of requests for a storage engine to roll back to a savepoint. Added in MySQL 5.0.3.

- `Handler_update`

  The number of requests to update a row in a table.

- `Handler_write`

  The number of requests to insert a row in a table.

- `Innodb_buffer_pool_pages_data`

  The number of pages containing data (dirty or clean). Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_dirty`

  The number of pages currently dirty. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_flushed`

  The number of buffer pool page-flush requests. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_free`

  The number of free pages. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_latched`

  The number of latched pages in `InnoDB` buffer pool. These are pages currently being read or written or that cannot be flushed or removed for some other reason. Added in MySQL 5.0.2. Calculation of this variable is expensive, so as of MySQL 5.0.68, it is available only when the `UNIV_DEBUG` system is defined at server build time.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Innodb_buffer_pool_pages_misc`

  The number of pages that are busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as `Innodb_buffer_pool_pages_total` − `Innodb_buffer_pool_pages_free` − `Innodb_buffer_pool_pages_data`. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_total`

  The total size of buffer pool, in pages. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_read_ahead_rnd`

  The number of “random” read-aheads initiated by `InnoDB`. This happens when a query scans a large portion of a table but in random order. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_read_ahead_seq`

  The number of sequential read-aheads initiated by `InnoDB`. This happens when `InnoDB` does a sequential full table scan. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_read_requests`

  The number of logical read requests. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_reads`

  The number of logical reads that `InnoDB` could not satisfy from the buffer pool, and had to read directly from the disk.

- `Innodb_buffer_pool_wait_free`

  Normally, writes to the `InnoDB` buffer pool happen in the background. However, if it is necessary to read or create a page and no clean pages are available, it is also necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size has been set properly, this value should be small. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_write_requests`

  The number writes done to the `InnoDB` buffer pool. Added in MySQL 5.0.2.

- `Innodb_data_fsyncs`

  The number of `fsync()` operations so far. Added in MySQL 5.0.2.

- `Innodb_data_pending_fsyncs`

  The current number of pending `fsync()` operations. Added in MySQL 5.0.2.

- `Innodb_data_pending_reads`

  The current number of pending reads. Added in MySQL 5.0.2.

- `Innodb_data_pending_writes`

  The current number of pending writes. Added in MySQL 5.0.2.

- `Innodb_data_read`

  The amount of data read since the server was started. Added in MySQL 5.0.2.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Innodb_data_reads`

  The total number of data reads. Added in MySQL 5.0.2.

- `Innodb_data_writes`

  The total number of data writes. Added in MySQL 5.0.2.

- `Innodb_data_written`

  The amount of data written so far, in bytes. Added in MySQL 5.0.2.

- `Innodb_dblwr_pages_written`

  The number of pages that have been written for doublewrite operations. Added in MySQL 5.0.2. See Section 14.2.11.1, “InnoDB Disk I/O”.

- `Innodb_dblwr_writes`

  The number of doublewrite operations that have been performed. Added in MySQL 5.0.2. See Section 14.2.11.1, “InnoDB Disk I/O”.

- `Innodb_log_waits`

  The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing. Added in MySQL 5.0.2.

- `Innodb_log_write_requests`

  The number of log write requests. Added in MySQL 5.0.2.

- `Innodb_log_writes`

  The number of physical writes to the log file. Added in MySQL 5.0.2.

- `Innodb_os_log_fsyncs`

  The number of `fsync()` writes done to the log file. Added in MySQL 5.0.2.

- `Innodb_os_log_pending_fsyncs`

  The number of pending log file `fsync()` operations. Added in MySQL 5.0.2.

- `Innodb_os_log_pending_writes`

  The number of pending log file writes. Added in MySQL 5.0.2.

- `Innodb_os_log_written`

  The number of bytes written to the log file. Added in MySQL 5.0.2.

- `Innodb_page_size`

  The compiled-in `InnoDB` page size (default 16KB). Many values are counted in pages; the page size permits them to be easily converted to bytes. Added in MySQL 5.0.2.

- `Innodb_pages_created`

  The number of pages created. Added in MySQL 5.0.2.

- `Innodb_pages_read`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of pages read. Added in MySQL 5.0.2.

- `Innodb_pages_written`

  The number of pages written. Added in MySQL 5.0.2.

- `Innodb_row_lock_current_waits`

  The number of row locks currently being waited for. Added in MySQL 5.0.3.

- `Innodb_row_lock_time`

  The total time spent in acquiring row locks, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_time_avg`

  The average time to acquire a row lock, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_time_max`

  The maximum time to acquire a row lock, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_waits`

  The number of times a row lock had to be waited for. Added in MySQL 5.0.3.

- `Innodb_rows_deleted`

  The number of rows deleted from `InnoDB` tables. Added in MySQL 5.0.2.

- `Innodb_rows_inserted`

  The number of rows inserted into `InnoDB` tables. Added in MySQL 5.0.2.

- `Innodb_rows_read`

  The number of rows read from `InnoDB` tables. Added in MySQL 5.0.2.

- `Innodb_rows_updated`

  The number of rows updated in `InnoDB` tables. Added in MySQL 5.0.2.

- `Key_blocks_not_flushed`

  The number of key blocks in the key cache that have changed but have not yet been flushed to disk.

- `Key_blocks_unused`

  The number of unused blocks in the key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in Section 5.1.4, “Server System Variables”.

- `Key_blocks_used`

  The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests`

  The number of requests to read a key block from the cache.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Key_reads`

  The number of physical reads of a key block from disk. If `Key_reads` is large, then your `key_buffer_size` value is probably too small. The cache miss rate can be calculated as `Key_reads`/`Key_read_requests`.

- `Key_write_requests`

  The number of requests to write a key block to the cache.

- `Key_writes`

  The number of physical writes of a key block to disk.

- `Last_query_cost`

  The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. This variable was added in MySQL 5.0.1, with a default value of -1. In MySQL 5.0.7, the default was changed to 0; also in version 5.0.7, the scope of `Last_query_cost` was changed to session rather than global.

  The `Last_query_cost` value can be computed accurately only for simple “flat” queries, not complex queries such as those with subqueries or `UNION`. For the latter, the value is set to 0.

  Prior to MySQL 5.0.16, this variable was not updated for queries served from the query cache.

- `Max_used_connections`

  The maximum number of connections that have been in use simultaneously since the server started.

- `Not_flushed_delayed_rows`

  The number of rows waiting to be written in `INSERT DELAYED` queues.

- `Open_files`

  The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- `Open_streams`

  The number of streams that are open (used mainly for logging).

- `Open_tables`

  The number of tables that are open.

- `Opened_tables`

  The number of tables that have been opened. If `Opened_tables` is big, your `table_cache` value is probably too small.

- `Prepared_stmt_count`

  The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` system variable.) This variable was added in MySQL 5.0.32.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Qcache_free_blocks`

  The number of free memory blocks in the query cache.

- `Qcache_free_memory`

  The amount of free memory for the query cache.

- `Qcache_hits`

  The number of query cache hits.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Qcache_inserts`

  The number of queries added to the query cache.

- `Qcache_lowmem_prunes`

  The number of queries that were deleted from the query cache because of low memory.

- `Qcache_not_cached`

  The number of noncached queries (not cacheable, or not cached due to the `query_cache_type` setting).

- `Qcache_queries_in_cache`

  The number of queries registered in the query cache.

- `Qcache_total_blocks`

  The total number of blocks in the query cache.

- `Queries`

  The number of statements executed by the server. This variable includes statements executed within stored programs, unlike the `Questions` variable. It does not count `COM_PING` or `COM_STATISTICS` commands. This variable was added in MySQL 5.0.76.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Questions`

  The number of statements executed by the server. As of MySQL 5.0.72, this includes only statements sent to the server by clients and no longer includes statements executed within stored programs, unlike the `Queries` variable. This variable does not count `COM_PING`, `COM_STATISTICS`, `COM_STMT_PREPARE`, `COM_STMT_CLOSE`, or `COM_STMT_RESET` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Rpl_status`

  The status of fail-safe replication (not implemented). This variable is unused and is removed in MySQL 5.6.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Select_full_join`

  The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- `Select_full_range_join`

  The number of joins that used a range search on a reference table.

- `Select_range`

  The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- `Select_range_check`

  The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- `Select_scan`

  The number of joins that did a full scan of the first table.

- `Slave_open_temp_tables`

  The number of temporary tables that the slave SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the slave; see Section 16.4.1.16, “Replication and Temporary Tables”.

- `Slave_retried_transactions`

  The total number of times since startup that the replication slave SQL thread has retried transactions. This variable was added in version 5.0.4.

- `Slave_running`

  This is `ON` if this server is a replication slave that is connected to a replication master, and both the I/O and SQL threads are running; otherwise, it is `OFF`.

- `Slow_launch_threads`

  The number of threads that have taken more than `slow_launch_time` seconds to create.

- `Slow_queries`

  The number of queries that have taken more than `long_query_time` seconds. This counter increments regardless of whether the slow query log is enabled. For information about that log, see Section 5.4.4, “The Slow Query Log”.

- `Sort_merge_passes`

  The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` system variable.

- `Sort_range`

  The number of sorts that were done using ranges.

- `Sort_rows`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of sorted rows.

- `Sort_scan`

  The number of sorts that were done by scanning the table.

- `Ssl_accept_renegotiates`

  The number of negotiates needed to establish the connection.

- `Ssl_accepts`

  The number of accepted SSL connections.

- `Ssl_callback_cache_hits`

  The number of callback cache hits.

- `Ssl_cipher`

  The current encryption cipher (empty for unencrypted connections).

- `Ssl_cipher_list`

  The list of possible SSL ciphers (empty for non-SSL connections).

- `Ssl_client_connects`

  The number of SSL connection attempts to an SSL-enabled master.

- `Ssl_connect_renegotiates`

  The number of negotiates needed to establish the connection to an SSL-enabled master.

- `Ssl_ctx_verify_depth`

  The SSL context verification depth (how many certificates in the chain are tested).

- `Ssl_ctx_verify_mode`

  The SSL context verification mode.

- `Ssl_default_timeout`

  The default SSL timeout.

- `Ssl_finished_accepts`

  The number of successful SSL connections to the server.

- `Ssl_finished_connects`

  The number of successful slave connections to an SSL-enabled master.

- `Ssl_session_cache_hits`

  The number of SSL session cache hits.

- `Ssl_session_cache_misses`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of SSL session cache misses.

- `Ssl_session_cache_mode`

  The SSL session cache mode.

- `Ssl_session_cache_overflows`

  The number of SSL session cache overflows.

- `Ssl_session_cache_size`

  The SSL session cache size.

- `Ssl_session_cache_timeouts`

  The number of SSL session cache timeouts.

- `Ssl_sessions_reused`

  How many SSL connections were reused from the cache.

- `Ssl_used_session_cache_entries`

  How many SSL session cache entries were used.

- `Ssl_verify_depth`

  The verification depth for replication SSL connections.

- `Ssl_verify_mode`

  The verification mode for replication SSL connections.

- `Ssl_version`

  The SSL protocol version of the connection; for example, TLSv1. If the connection is not encrypted, the value is empty.

- `Table_locks_immediate`

  The number of times that a request for a table lock could be granted immediately.

- `Table_locks_waited`

  The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- `Tc_log_max_pages_used`

  For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of `Tc_log_max_pages_used` and `Tc_log_page_size` is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size` option. This variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines that are capable of two-phase commit and that support XA transactions is greater than one. (`InnoDB` is the only applicable engine.) Added in MySQL 5.0.3.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Tc_log_page_size`

  The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `getpagesize()`. This variable is unused for the same reasons as described for `Tc_log_max_pages_used`. Added in MySQL 5.0.3.

- `Tc_log_page_waits`

  For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.) Added in MySQL 5.0.3.

- `Threads_cached`

  The number of threads in the thread cache.

- `Threads_connected`

  The number of currently open connections.

- `Threads_created`

  The number of threads created to handle connections. If `Threads_created` is big, you may want to increase the `thread_cache_size` value. The cache miss rate can be calculated as `Threads_created`/`Connections`.

- `Threads_running`

  The number of threads that are not sleeping.

- `Uptime`

  The number of seconds that the server has been up.

- `Uptime_since_flush_status`

  The number of seconds since the most recent `FLUSH STATUS` statement. This variable was added in 5.0.35. (MySQL Community only)

5.1.7 Server SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

- Setting the SQL Mode
- The Most Important SQL Modes
- Full List of SQL Modes
- Strict SQL Mode

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Combination SQL Modes

Setting the SQL Mode

The default SQL mode is empty (no modes set).

To set the SQL mode at server startup, use the `--sql-mode="modes"` option on the command line, or `sql-mode="modes"` in an option file such as `my.cnf` (Unix operating systems) or `my.ini` (Windows). *`modes`* is a list of different modes separated by commas. To clear the SQL mode explicitly, set it to an empty string using `--sql-mode=""` on the command line, or `sql-mode=""` in an option file.

Note

MySQL installation programs may configure the SQL mode during the installation process. If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

To change the SQL mode at runtime, set the global or session `sql_mode` system variable using a `SET` statement:

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

Setting the `GLOBAL` variable requires the `SUPER` privilege and affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Each client can change its session `sql_mode` value at any time.

To determine the current global or session `sql_mode` value, use the following statements:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

The Most Important SQL Modes

The most important `sql_mode` values are probably these:

- `ANSI`

  This mode changes syntax and behavior to conform more closely to standard SQL.

- `STRICT_TRANS_TABLES`

  If a value could not be inserted as given into a transactional table, abort the statement. For a nontransactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More details are given later in this section. (Implemented in MySQL 5.0.2)

- `TRADITIONAL`

  Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column.

Note

The `INSERT` or `UPDATE` aborts as soon as the error is noticed. This may not be what you want if you are using a nontransactional storage engine, because data changes made prior to the error may not be rolled back, resulting in a “partially done” update. (Added in MySQL 5.0.2)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When this manual refers to “strict mode,” it means a mode with either or both `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` enabled.

Full List of SQL Modes

The following list describes all supported SQL modes:

- `ALLOW_INVALID_DATES`

  Do not perform full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This is very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation). This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

  This mode is implemented in MySQL 5.0.2. Before 5.0.2, this was the default MySQL date-handling mode. As of 5.0.2, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`.

- `ANSI_QUOTES`

  Treat “`"`” as an identifier quote character (like the “`` ` ``” quote character) and not as a string quote character. You can still use “`` ` ``” to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotation marks to quote literal strings, because it is interpreted as an identifier.

- `ERROR_FOR_DIVISION_BY_ZERO`

  The `ERROR_FOR_DIVISION_BY_ZERO` mode affects handling of division by zero, which includes `MOD(N,0)`. For data-change operations (`INSERT`, `UPDATE`), its effect also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, division by zero inserts `NULL` and produces no warning.
  - If this mode is enabled, division by zero inserts `NULL` and produces a warning.
  - If this mode and strict mode are enabled, division by zero produces an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, division by zero inserts `NULL` and produces a warning.

  For `SELECT`, division by zero returns `NULL`. Enabling `ERROR_FOR_DIVISION_BY_ZERO` causes a warning to be produced as well, regardless of whether strict mode is enabled.

  This mode was implemented in MySQL 5.0.2.

- `HIGH_NOT_PRECEDENCE`

  From MySQL 5.0.2 on, the precedence of the `NOT` operator is such that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. Before MySQL 5.0.2, the expression is parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` SQL mode. (Added in MySQL 5.0.2)

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
       -> 1
```

- `IGNORE_SPACE`

  Permit spaces between a function name and the "`(`" character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 9.2, "Schema Object Names". For example, because there is a `COUNT()` function, the use of `count` as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

  The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

  The `IGNORE_SPACE` SQL mode applies to built-in functions, not to user-defined functions or stored functions. It is always permissible to have spaces after a UDF or stored function name, regardless of whether `IGNORE_SPACE` is enabled.

  For further discussion of `IGNORE_SPACE`, see Section 9.2.3, "Function Name Parsing and Resolution".

- `NO_AUTO_CREATE_USER`

  Prevent the `GRANT` statement from automatically creating new users if it would otherwise do so, unless a nonempty password also is specified. (Added in MySQL 5.0.2)

- `NO_AUTO_VALUE_ON_ZERO`

  `NO_AUTO_VALUE_ON_ZERO` affects handling of `AUTO_INCREMENT` columns. Normally, you generate the next sequence number for the column by inserting either `NULL` or `0` into it. `NO_AUTO_VALUE_ON_ZERO` suppresses this behavior for `0` so that only `NULL` generates the next sequence number.

  This mode can be useful if `0` has been stored in a table's `AUTO_INCREMENT` column. (Storing `0` is not a recommended practice, by the way.) For example, if you dump the table with `mysqldump` and then reload it, MySQL normally generates new sequence numbers when it encounters the `0` values, resulting in a table with contents different from the one that was dumped. Enabling `NO_AUTO_VALUE_ON_ZERO` before reloading the dump file solves this problem. `mysqldump` now automatically includes in its output a statement that enables `NO_AUTO_VALUE_ON_ZERO`, to avoid this problem.

- `NO_BACKSLASH_ESCAPES`

  Disable the use of the backslash character ("`\`") as an escape character within strings. With this mode enabled, backslash becomes an ordinary character like any other. (Implemented in MySQL 5.0.1)

- `NO_DIR_IN_CREATE`

  When creating a table, ignore all `INDEX DIRECTORY` and `DATA DIRECTORY` directives. This option is useful on slave replication servers.

- `NO_ENGINE_SUBSTITUTION`

  Control automatic substitution of the default storage engine when a statement such as `CREATE TABLE` or `ALTER TABLE` specifies a storage engine that is disabled or not compiled in. (Implemented in MySQL 5.0.8)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

With NO_ENGINE_SUBSTITUTION disabled, the default engine is used and a warning occurs if the desired engine is known but disabled or not compiled in. If the desired engine is invalid (not a known engine name), an error occurs and the table is not created or altered.

With NO_ENGINE_SUBSTITUTION enabled, an error occurs and the table is not created or altered if the desired engine is unavailable for any reason (whether disabled or invalid).

- NO_FIELD_OPTIONS

  Do not print MySQL-specific column options in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_KEY_OPTIONS

  Do not print MySQL-specific index options in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_TABLE_OPTIONS

  Do not print MySQL-specific table options (such as ENGINE) in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_UNSIGNED_SUBTRACTION

  By default, subtraction between integer operands produces an UNSIGNED result if any operand is UNSIGNED. When NO_UNSIGNED_SUBTRACTION is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column c2 in table t1 with that of column c2 in table t2:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c2    | bigint(21) unsigned | NO   |     | 0       |       |
+-------+---------------------+------+-----+---------+-------+

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| c2    | bigint(21) | NO   |     | 0       |       |
+-------+------------+------+-----+---------+-------+
```

This means that BIGINT UNSIGNED is not 100% usable in all contexts. See Section 12.10, “Cast Functions and Operators”.

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|    18446744073709551615 |
+-------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

- `NO_ZERO_DATE`

  The `NO_ZERO_DATE` mode affects whether the server permits `'0000-00-00'` as a valid date. Its effect also depends on whether strict SQL mode is enabled. This mode was added in MySQL 5.0.2.

  - If this mode is not enabled, `'0000-00-00'` is permitted and inserts produce no warning.
  - If this mode is enabled, `'0000-00-00'` is permitted and inserts produce a warning.
  - If this mode and strict mode are enabled, `'0000-00-00'` is not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, `'0000-00-00'` is permitted and inserts produce a warning.

- `NO_ZERO_IN_DATE`

  The `NO_ZERO_IN_DATE` mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0. (This mode affects dates such as `'2010-00-01'` or `'2010-01-00'`, but not `'0000-00-00'`. To control whether the server permits `'0000-00-00'`, use the `NO_ZERO_DATE` mode.) The effect of `NO_ZERO_IN_DATE` also depends on whether strict SQL mode is enabled. This mode was added in MySQL 5.0.2.

  - If this mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
  - If this mode is enabled, dates with zero parts are inserted as `'0000-00-00'` and produce a warning.
  - If this mode and strict mode are enabled, dates with zero parts are not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, dates with zero parts are inserted as `'0000-00-00'` and produce a warning.

- `ONLY_FULL_GROUP_BY`

  Reject queries for which the select list or (as of MySQL 5.0.23) `HAVING` list refer to nonaggregated columns that are not named in the `GROUP BY` clause.

  A MySQL extension to standard SQL permits references in the `HAVING` clause to aliased expressions in the select list. Enabling `ONLY_FULL_GROUP_BY` disables this extension, thus requiring the `HAVING` clause to be written using unaliased expressions.

  For additional discussion and examples, see Section 12.16.3, “MySQL Handling of GROUP BY”.

- `PIPES_AS_CONCAT`

  Treat `||` as a string concatenation operator (same as `CONCAT()`) rather than as a synonym for `OR`.

- `REAL_AS_FLOAT`

  Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`.

- `STRICT_ALL_TABLES`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Enable strict mode for all storage engines. Invalid data values are rejected. For details, see Strict SQL Mode. (Added in MySQL 5.0.2)

- `STRICT_TRANS_TABLES`

  Enable strict mode for transactional storage engines, and when possible for nontransactional storage engines. For details, see Strict SQL Mode. (Implemented in MySQL 5.0.2)

Combination SQL Modes

The following special modes are provided as shorthand for combinations of mode values from the preceding list. All are available beginning with version MySQL 5.0.0, except for `TRADITIONAL`, which was implemented in MySQL 5.0.2.

- `ANSI`

  Equivalent to `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`. Before MySQL 5.0.3, `ANSI` also includes `ONLY_FULL_GROUP_BY`.

  As of MySQL 5.0.40, `ANSI` mode also causes the server to return an error for queries where a set function *`S`* with an outer reference *`S(outer_ref)`* cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

  ```
  SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
  ```

  Here, `MAX(t1.b)` cannot aggregated in the outer query because it appears in the `WHERE` clause of that query. Standard SQL requires an error in this situation. If `ANSI` mode is not enabled, the server treats *`S(outer_ref)`* in such queries the same way that it would interpret *`S(const)`*, as was always done prior to 5.0.40.

  See Section 1.8, “MySQL Standards Compliance”.

- `DB2`

  Equivalent to `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`.

- `MAXDB`

  Equivalent to `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`, `NO_AUTO_CREATE_USER`.

- `MSSQL`

  Equivalent to `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`.

- `MYSQL323`

  Equivalent to `MYSQL323`, `HIGH_NOT_PRECEDENCE`. This means `HIGH_NOT_PRECEDENCE` plus some `SHOW CREATE TABLE` behaviors specific to `MYSQL323`:

  - `TIMESTAMP` column display does not include `DEFAULT` or `ON UPDATE` attributes that were introduced in MySQL 4.1.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - String column display does not include character set and collation attributes that were introduced in MySQL 4.1. For `CHAR` and `VARCHAR` columns, if the collation is binary, `BINARY` is appended to the column type.
  - The `ENGINE=engine_name` table option displays as `TYPE=engine+name`.
  - For `MEMORY` tables, the storage engine is displayed as `HEAP`.

- `MYSQL40`

  Equivalent to `MYSQL40`, `HIGH_NOT_PRECEDENCE`. This means `HIGH_NOT_PRECEDENCE` plus some behaviors specific to `MYSQL40`. These are the same as for `MYSQL323`, except that `SHOW CREATE TABLE` does not display `HEAP` as the storage engine for `MEMORY` tables.

- `ORACLE`

  Equivalent to `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`, `NO_AUTO_CREATE_USER`.

- `POSTGRESQL`

  Equivalent to `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`.

- `TRADITIONAL`

  Equivalent to `STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, `NO_AUTO_CREATE_USER`.

Strict SQL Mode

Strict mode controls how MySQL handles invalid or missing values in data-change statements such as `INSERT` or `UPDATE`. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-`NULL` column that has no explicit `DEFAULT` clause in its definition. (For a `NULL` column, `NULL` is inserted if the value is missing.)

If strict mode is not in effect, MySQL inserts adjusted values for invalid or missing values and produces warnings (see Section 13.7.5.37, “SHOW WARNINGS Syntax”). In strict mode, you can produce this behavior by using `INSERT IGNORE` or `UPDATE IGNORE`.

For statements such as `SELECT` that do not change data, invalid values generate a warning in strict mode, not an error.

Strict mode does not affect whether foreign key constraints are checked. `foreign_key_checks` can be used for that. (See Section 5.1.4, “Server System Variables”.)

Strict SQL mode is in effect if either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled, although the effects of these modes differ somewhat:

- For transactional tables, an error occurs for invalid or missing values in a data-change statement when either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled. The statement is aborted and rolled back.
- For nontransactional tables, the behavior is the same for either mode if the bad value occurs in the first row to be inserted or updated: The statement is aborted and the table remains unchanged. If the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict mode is enabled:

- For `STRICT_ALL_TABLES`, MySQL returns an error and ignores the rest of the rows. However, because the earlier rows have been inserted or updated, the result is a partial update. To avoid this, use single-row statements, which can be aborted without changing the table.
- For `STRICT_TRANS_TABLES`, MySQL converts an invalid value to the closest valid value for the column and inserts the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in Section 11.6, "Data Type Default Values".

Strict mode also affects handling of division by zero, zero dates, and zeros in dates, in conjunction with the `ERROR_FOR_DIVISION_BY_ZERO`, `NO_ZERO_DATE`, and `NO_ZERO_IN_DATE` modes. For details, see the descriptions of those modes.

5.1.8 Server-Side Help

MySQL Server supports a `HELP` statement that returns information from the MySQL Reference manual (see Section 13.8.3, "HELP Syntax"). Several tables in the `mysql` system database contain the information needed to support this statement (see Section 5.3, "The mysql System Database"). The proper operation of this statement requires that these help tables be initialized, which is done by processing the contents of the `fill_help_tables.sql` script.

If you install MySQL using a binary or source distribution on Unix, help table content initialization occurs when you initialize the data directory (see Section 2.18.1, "Initializing the Data Directory"). For an RPM distribution on Linux or binary distribution on Windows, content initialization occurs as part of the MySQL installation process.

If you upgrade MySQL using a binary distribution, help table content is not upgraded automatically, but you can upgrade it manually. Locate the `fill_help_tables.sql` file in the `share` or `share/mysql` directory. Change location into that directory and process the file with the `mysql` client as follows:

```
shell> mysql -u root mysql < fill_help_tables.sql
```

You can also obtain the latest `fill_help_tables.sql` at any time to upgrade your help tables. Download the proper file for your version of MySQL from http://dev.mysql.com/doc/index-other.html. After downloading and uncompressing the file, process it with `mysql` as described previously.

If you are working with Bazaar and a MySQL development source tree, you must use a downloaded copy of the `fill_help_tables.sql` file because the source tree contains only a "stub" version.

5.1.9 Server Response to Signals

On Unix, signals can be sent to processes. `mysqld` responds to signals sent to it as follows:

- `SIGTERM` causes the server to shut down.
- `SIGHUP` causes the server to reload the grant tables and to flush tables, logs, the thread cache, and the host cache. These actions are like various forms of the `FLUSH` statement. The server also writes a status report to the error log that has this format:

```
Status information:

Current dir: /var/mysql/data/
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Running threads: 0  Stack size: 196608
Current locks:

Key caches:
default
Buffer_size:       8388600
Block_size:           1024
Division_limit:        100
Age_limit:             300
blocks used:             0
not flushed:             0
w_requests:              0
writes:                  0
r_requests:              0
reads:                   0

handler status:
read_key:            0
read_next:           0
read_rnd             0
read_first:          1
write:               0
delete               0
update:              0

Table status:
Opened tables:          5
Open tables:            0
Open files:             7
Open streams:           0

Alarm status:
Active alarms:   1
Max used alarms: 2
Next alarm time: 67
```

5.1.10 The Server Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

   This can occur initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

   Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

   ```
   Error: Can't create thread to kill server
   ```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections by closing the handlers for the network interfaces to which it normally listens for

This documentation is for an older version. If you're
This documentation is for an older version. If you're

connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread associated with a client connection, the server breaks the connection to the client and marks the thread as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see Section 13.7.6.3, “KILL Syntax”, in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

   For threads that have an open transaction, the transaction is rolled back. If a thread is updating a nontransactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated because the operation can terminate before completion.

   If the server is a master replication server, it treats threads associated with currently connected slaves like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a slave replication server, it stops the I/O and SQL threads, if they are active, before marking client threads as killed. The SQL thread is permitted to finish its current statement (to avoid causing replication problems), and then stops. In MySQL 5.0.80 and earlier, if the SQL thread was in the middle of a transaction at this point, the transaction was rolled back; in MySQL 5.0.81 and later, the server waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. See also Section 13.4.2.8, “STOP SLAVE Syntax”.

   If the slave is updating a nontransactional table when it is forcibly killed, the slave's data may become inconsistent with the master.

5. The server shuts down or closes storage engines.

   At this stage, the server flushes the table cache and closes all open tables.

   Each storage engine performs any actions necessary for tables that it manages. For example, `MyISAM` flushes any pending index writes for a table. `InnoDB` flushes its buffer pool to disk (starting from 5.0.5: unless `innodb_fast_shutdown` is 2), writes the current LSN to the tablespace, and terminates its own internal threads.

6. The server exits.

5.2 The MySQL Data Directory

Information managed by the MySQL server is stored under a directory known as the data directory. The following list briefly describes the items typically found in the data directory, with cross references for additional information:

- Each data directory subdirectory corresponds to a database managed by the server:
  - The `mysql` directory corresponds to the `mysql` system database, which contains information required by the MySQL server as it runs. See Section 5.3, “The mysql System Database”.
  - Other subdirectories correspond to databases created by users or applications.
- Log files written by the server. See Section 5.4, “MySQL Server Logs”.
- `InnoDB` tablespace and log files. See Section 14.2, “The InnoDB Storage Engine”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The server process ID file (while the server is running).

Some items in the preceding list can be relocated elsewhere; for any given MySQL installation, check the server configuration to determine whether items have been moved. In addition, the location of the data directory itself can be discovered or configured using the `datadir` system variable.

5.3 The mysql System Database

The `mysql` database is the system database. It contains tables that store information required by the MySQL server as it runs.

Tables in the `mysql` database fall into these categories:

- Grant tables
- Object information tables
- Server-side help tables
- Time zone tables

Grant System Tables

These system tables contain grant information about user accounts and the privileges held by them:

- `user`: User accounts, global privileges, and other non-privilege columns.
- `db`: Database-level privileges.
- `host`: Obsolete.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.
- `procs_priv`: Stored procedure and function privileges.

For more information about the structure, contents, and purpose of the grant tables, see Section 6.2.2, “Grant Tables”.

Object Information System Tables

The `func` system table contains information about user-defined functions. See Section 21.2, “Adding New Functions to MySQL”.

Server-Side Help System Tables

These system tables contain server-side help information:

- `help_category`: Information about help categories.
- `help_keyword`: Keywords associated with help topics.
- `help_relation`: Mappings between help keywords and topics.
- `help_topic`: Help topic contents.

For more information, see Section 5.1.8, “Server-Side Help”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Time Zone System Tables

These system tables contain time zone information:

- `time_zone`: List of time zone IDs and whether they use leap seconds.
- `time_zone_leap_second`: When leap seconds occur.
- `time_zone_name`: Mappings between time zone IDs and names.
- `time_zone_transition`, `time_zone_transition_type`: Time zone descriptions.

For more information, see Section 10.6, “MySQL Server Time Zone Support”.

5.4 MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping `mysqld`
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication master server
Slow query log	Queries that took more than `long_query_time` seconds to execute

By default, no logs are enabled (except the error log on Windows). The following log-specific sections provide information about the server options that enable logging.

By default, the server writes files for all enabled logs in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement; execute `mysqladmin` with a `flush-logs` or `refresh` argument; or execute `mysqldump` with a `--flush-logs` or `--master-data` option. See Section 13.7.6.2, “FLUSH Syntax”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` system variable.

The relay log is used only on slave replication servers, to hold data changes from the master server that must also be made on the slave. For discussion of relay log contents and configuration, see Section 16.2.2.1, “The Slave Relay Log”.

For information about log maintenance operations such as expiration of old log files, see Section 5.4.5, “Server Log Maintenance”.

For information about keeping logs secure, see Section 6.1.2.3, “Passwords and Logging”.

5.4.1 The Error Log

The error log contains information indicating when `mysqld` was started and stopped and also any critical errors that occur while the server is running. If `mysqld` notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if `mysqld` exits abnormally. The trace can be used to determine where `mysqld` exited. See Section 21.3, “Debugging and Porting MySQL”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If `mysqld_safe` is used to start `mysqld` and `mysqld` exits abnormally, `mysqld_safe` notices this, restarts `mysqld`, and writes a `mysqld restarted` message to the error log.

In the following discussion, “console” means `stderr`, the standard error output; this is your terminal or console window unless the standard error output has been redirected.

On Windows, the `--log-error`, `--pid-file`, and `--console` options affect error logging:

- If no log file name is specified, the default log file is *`host_name`*`.err` in the data directory, unless the `--pid-file` option is specified. In that case, the default name is the PID file base name with a suffix of `.err` in the data directory.
- Without `--log-error`, `mysqld` writes error messages to the default log file.
- With `--log-error[=`*`file_name`*`]`, `mysqld` writes error messages to an error log file. `mysqld` writes to the named file if present, creating it in the data directory unless an absolute path name is given to specify a different directory. If no file is named, `mysqld` writes to the default log file.
- With `--console`, `mysqld` writes error messages to the console, unless `--log-error` is also given. If both options are present, the last one takes precedence.

In addition, on Windows, the server writes events and error messages to the Windows Event Log within the Application log. Entries marked as `Warning` and `Note` are written to the Event Log, but not informational messages such as information statements from individual storage engines. These log entries have a source of `MySQL`. You cannot disable writing information to the Windows Event Log.

On Unix and Unix-like systems, `mysqld` writes error log messages as follows:

- Without `--log-error`, `mysqld` writes error messages to the console.
- With `--log-error[=file_name]`, `mysqld` writes error messages to an error log file. The server uses the named file if present, creating it in the data directory unless an absolute path name is given to specify a different directory. If no file is named, the default name is *`host_name`*`.err` in the data directory.

At runtime, `log_error` system variable indicates the error log file name if error output is written to a file.

If you flush the logs using `FLUSH LOGS` or `mysqladmin flush-logs` and `mysqld` is writing the error log to a file (for example, if it was started with the `--log-error` option), it renames the current log file with the suffix `-old`, then creates a new empty log file. Be aware that a second log-flushing operation thus causes the original error log file to be lost unless you save it under a different name. For example, you can use the following commands to save the file:

```
shell> mysqladmin flush-logs
shell> mv host_name.err-old backup-directory
```

If the server is not writing to a named file, no error log renaming occurs when the logs are flushed.

If you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for `mysqld` to write error messages to a log file. If you specify a file name using `--log-error` to `mysqld_safe` or `mysqld`, that file name is used. Otherwise, `mysqld_safe` uses the default error log file.

The `--log-warnings` option or `log_warnings` system variable can be used to control warning logging to the error log. The default value is enabled (1). Warning logging can be disabled using a value of 0. If the value is greater than 1, aborted connections are written to the error log. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

5.4.2 The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order is in contrast with that of the binary log, for which statements are written after they are executed but before any locks are released. (Also, the query log contains all statements, whereas the binary log does not contain statements that only select data.)

To enable the general query log, start `mysqld` with the `--log[=file_name]` or `-l [file_name]` option.

If the general query log file is enabled but no name is specified, the default name is *`host_name`*`.log` and the server creates the file in the same directory where it creates the PID file. If a name is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). On Unix, to rename the file and create a new one, use the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

On Windows, you cannot rename a log file while the server has it open before MySQL 5.0.17. You must stop the server, rename the file, and then restart the server to create a new log file. As of 5.0.17, this applies only to the error log. However, a stop and restart can be avoided by using `FLUSH LOGS`, which causes the server to rename the error log with an `-old` suffix and open a new error log.

The general query log should be protected because logged statements might contain passwords. See Section 6.1.2.3, “Passwords and Logging”.

5.4.3 The Binary Log

The binary log contains “events” that describe database changes such as table creation operations or changes to table data. It also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows). The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log on a master replication server provides a record of the data changes to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 16.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Note

The binary log has replaced the old update log, which is no longer available as of MySQL 5.0. The binary log contains all information that is available in the update

This documentation is for an older version. If you're
This documentation is for an older version. If you're

log in a more efficient format and in a manner that is transaction-safe. If you are using transactions, you must use the MySQL binary log for backups instead of the old update log.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. To log all statements (for example, to identify a problem query), use the general query log. See Section 5.4.2, “The General Query Log”.

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

The binary log should be protected because logged statements might contain passwords. See Section 6.1.2.3, “Passwords and Logging”.

The following discussion describes some of the server options and variables that affect the operation of binary logging. For a complete list, see Section 16.1.2.4, “Binary Log Options and Variables”.

For detailed information about the format of the binary log, see MySQL Internals: The Binary Log.

To enable the binary log, start the server with the `--log-bin[=base_name]` option. If no *`base_name`* value is given, the default name is the value of the `pid-file` option (which by default is the name of host machine) followed by `-bin`. If the base name is given, the server writes the file in the data directory unless the base name is given with a leading absolute path name to specify a different directory. It is recommended that you specify a base name explicitly rather than using the default of the host name; see Section B.5.7, “Known Issues in MySQL”, for the reason.

Note

From MySQL 5.0.41 through 5.0.52, “mysql” was used when no *`base_name`* was specified. Also in these versions, a path given as part of the `--log-bin` options was treated as absolute rather than relative. The previous behaviors were restored in MySQL 5.0.54. (See Bug #28603 and Bug #28597.)

If you supply an extension in the log name (for example, `--log-bin=base_name.extension`), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log base name to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time it starts or flushes the logs. The server also creates a new binary log file automatically after the current log's size reaches `max_binlog_size`. A binary log file may become larger than `max_binlog_size` if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of all used binary log files. By default, this has the same base name as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

A client that has the `SUPER` privilege can disable binary logging of its own statements by using a `SET sql_log_bin=0` statement. See Section 5.1.4, “Server System Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server evaluates the `--binlog-do-db` and `--binlog-ignore-db` options in the same way as it does the `--replicate-do-db` and `--replicate-ignore-db` options. For information about how this is done, see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

A replication slave server by default does not write to its own binary log any data modifications that are received from the replication master. To log these modifications, start the slave with the `--log-slave-updates` option in addition to the `--log-bin` option (see Section 16.1.2.3, “Replication Slave Options and Variables”). This is done when a slave is also to act as a master to other slaves in chained replication.

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE BINARY LOGS`. See Section 13.7.6.5, “RESET Syntax”, and Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

If you are using replication, you should not delete old binary log files on the master until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute `mysqladmin flush-logs` on the master and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use `PURGE BINARY LOGS`, which also safely updates the binary log index file for you (and which can take a date argument). See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

You can display the contents of binary log files with the `mysqlbinlog` utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` also can be used to display replication slave relay log file contents because they are written using the same format as binary log files. For more information on the `mysqlbinlog` utility and how to use it, see Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”. For more information about the binary log and recovery operations, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Binary logging is done immediately after a statement or transaction completes but before any locks are released or any commit is done. This ensures that the log is logged in commit order.

Updates to nontransactional tables are stored in the binary log immediately after execution. In MySQL 5.0.53 and earlier versions of MySQL 5.0, an `UPDATE` statement using a stored function that modified a nontransactional table was not logged if it failed, and an `INSERT ... ON DUPLICATE KEY UPDATE` statement that encountered a duplicate key constraint—but did not actually change any data—was not logged. Beginning with MySQL 5.0.54, both of these statements are written to the binary log. (Bug #23333)

Within an uncommitted transaction, all updates (`UPDATE`, `DELETE`, or `INSERT`) that change transactional tables such as `BDB` or `InnoDB` tables are cached until a `COMMIT` statement is received by the server. At that point, `mysqld` writes the entire transaction to the binary log before the `COMMIT` is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that the modifications to those tables are replicated.

When a thread that handles the transaction starts, it allocates a buffer of `binlog_cache_size` to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

The `Binlog_cache_use` status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The `Binlog_cache_disk_use` status variable shows

This documentation is for an older version. If you're
This documentation is for an older version. If you're

how many of those transactions actually had to use a temporary file. These two variables can be used for tuning `binlog_cache_size` to a large enough value that avoids the use of temporary files.

The `max_binlog_cache_size` system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement-based logging, the original statement is written to the log.

The binary log format has some known limitations that can affect recovery from backups. See Section 16.4.1, “Replication Features and Issues”.

Binary logging for stored programs is done as described in Section 18.6, “Binary Logging of Stored Programs”.

Note that the binary log format differs in MySQL 5.0 from previous versions of MySQL, due to enhancements in replication. See Section 16.4.2, “Replication Compatibility Between MySQL Versions”.

Writes to the binary log file and binary log index file are handled in the same way as writes to `MyISAM` tables. See Section B.5.3.4, “How MySQL Handles a Full Disk”.

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every *N* writes to the binary log, with the `sync_binlog` system variable. See Section 5.1.4, “Server System Variables”. 1 is the safest value for `sync_binlog`, but also the slowest. Even with `sync_binlog` set to 1, there is still the chance of an inconsistency between the table content and binary log content in case of a crash. For example, if you are using `InnoDB` tables and the MySQL server processes a `COMMIT` statement, it writes the whole transaction to the binary log and then commits this transaction into `InnoDB`. If the server crashes between those two operations, the transaction is rolled back by `InnoDB` at restart but still exists in the binary log. This problem can be solved with the `--innodb-safe-binlog` option, which adds consistency between the content of `InnoDB` tables and the binary log. (Note: `--innodb-safe-binlog` is unneeded as of MySQL 5.0; it was made obsolete by the introduction of XA transaction support.)

For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the `InnoDB` logs to disk at every transaction. The `InnoDB` logs are synchronized by default, and `sync_binlog=1` can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back `InnoDB` transactions from the binary log. This ensures that the binary log reflects the exact data of `InnoDB` tables, and so, that the slave remains in synchrony with the master (not receiving a statement which has been rolled back).

Note that `--innodb-safe-binlog` can be used even if the MySQL server updates other storage engines than `InnoDB`. Only statements and transactions that affect `InnoDB` tables are subject to removal from the binary log at `InnoDB`'s crash recovery. If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed `InnoDB` transaction. This should not happen if `sync_binlog=1` and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message `The binary log file_name is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

For MySQL 5.0.46, the session values of the following system variables are written to the binary log and honored by the replication slave when parsing the binary log:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- sql_mode
- foreign_key_checks
- unique_checks
- character_set_client
- collation_connection
- collation_database
- collation_server
- sql_auto_is_null

5.4.4 The Slow Query Log

The slow query log consists of SQL statements that took more than long_query_time seconds to execute. The minimum and default values of long_query_time are 1 and 10, respectively.

By default, administrative statements are not logged, nor are queries that do not use indexes for lookups. This behavior can be changed using --log-slow-admin-statements and --log-queries-not-using-indexes, as described later.

The time to acquire the initial locks is not counted as execution time. mysqld writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

To enable the slow query log, start mysqld with the --log-slow-queries[=*file_name*] option.

If the slow query log file is enabled but no name is specified, the default name is *host_name*-slow.log and the server creates the file in the same directory where it creates the PID file. If a name is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To include slow administrative statements in the statements written to the slow query log, use the --log-slow-admin-statements server option. Administrative statements include ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE.

To include queries that do not use indexes for row lookups in the statements written to the slow query log, use the --log-queries-not-using-indexes server option. See Section 5.1.3, “Server Command Options”. When such queries are logged, the slow query log may grow quickly.

The server uses the controlling parameters in the following order to determine whether to write a query to the slow query log:

1. The query must either not be an administrative statement, or --log-slow-admin-statements must have been specified.
2. The query must have taken at least long_query_time seconds, or --log-queries-not-using-indexes must have been specified and the query used no indexes for row lookups.

The server does not write queries handled by the query cache to the slow query log, nor queries that would not benefit from the presence of an index because the table has zero rows or one row.

Replication slaves do not write replicated queries to the slow query log, even if the same queries were written to the slow query log on the master. This is a known issue. (Bug #23300)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The slow query log should be protected because logged statements might contain passwords. See Section 6.1.2.3, “Passwords and Logging”.

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can process a slow query log file using the `mysqldumpslow` command to summarize the queries that appear in the log. See Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”.

5.4.5 Server Log Maintenance

As described in Section 5.4, “MySQL Server Logs”, MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 7.2, “Database Backup Methods”.

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

For the binary log, you can set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days (see Section 5.1.4, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master. To remove binary logs on demand, use the `PURGE BINARY LOGS` statement (see Section 13.4.1.1, “PURGE BINARY LOGS Syntax”).

You can force MySQL to start using new log files by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement or execute a `mysqladmin flush-logs`, `mysqladmin refresh`, `mysqldump --flush-logs`, or `mysqldump --master-data` command. See Section 13.7.6.2, “FLUSH Syntax”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` system variable.

A log-flushing operation does the following:

- If general query logging (`--log`) or slow query logging (`--log-slow-queries`) to a log file is enabled, the server closes and reopens the general query log file or slow query log file.
- If binary logging (`--log-bin`) is used, the server closes the current log file and opens a new log file with the next sequence number.
- If the server was started with the `--log-error` option to cause the error log to be written to a file, it renames the current log file with the suffix `-old` and creates a new empty error log file.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current log files before flushing them. At flush time, the server opens new log files with the original names. For example, if the general and slow query log files are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.old` and `mysql-slow.old` and then remove them from disk.

For older versions of MySQL, you cannot rename certain log files on Windows while the server has them open. Before MySQL 5.0.17, this restriction applies to all log files. You must stop the server, rename the file, then restart the server to create a new log file. From 5.0.18 on, the restriction applies only to the error log file. To rename the error log file, a stop and restart can be avoided by flushing the logs to cause the server to rename the current log file with the suffix `-old` and create a new empty error log file.

To disable or enable general query logging for the current connection, set the session `sql_log_off` variable to `ON` or `OFF`.

5.5 Running Multiple MySQL Instances on One Machine

In some cases, you might want to run multiple instances of MySQL on a single machine. You might want to test a new MySQL release while leaving an existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

It is possible to use a different MySQL server binary per instance, or use the same binary for multiple instances, or any combination of the two approaches. For example, you might run a server from MySQL 4.1 and one from MySQL 5.0, to see how different versions handle a given workload. Or you might run multiple instances of the current production version, each managing a different set of databases.

Whether or not you use distinct server binaries, each instance that you run must be configured with unique values for several operating parameters. This eliminates the potential for conflict between instances. Parameters can be set on the command line, in option files, or by setting environment variables. See Section 4.2.3, “Specifying Program Options”. To see the values used by a given instance, connect to it and execute a `SHOW VARIABLES` statement.

The primary resource managed by a MySQL instance is the data directory. Each instance should use a different data directory, the location of which is specified using the `--datadir=dir_name` option. For methods of configuring each instance with its own data directory, and warnings about the dangers of failing to do so, see Section 5.5.1, “Setting Up Multiple Data Directories”.

In addition to using different data directories, several other options must have different values for each server instance:

- `--port=port_num`

  `--port` controls the port number for TCP/IP connections. Alternatively, if the host has multiple network addresses, you can use `--bind-address` to cause each server to listen to a different address.

- `--socket={file_name|pipe_name}`

  `--socket` controls the Unix socket file path on Unix or the named pipe name on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers configured to permit named-pipe connections.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--shared-memory-base-name=name`

  This option is used only on Windows. It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory. It is necessary to specify distinct shared-memory names only for those servers configured to permit shared-memory connections.

- `--pid-file=file_name`

  This option indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, their values must differ for each server:

- `--log[=file_name]`
- `--log-bin[=file_name]`
- `--log-error[=file_name]`
- `--bdb-logdir=file_name`

For further discussion of log file options, see Section 5.4, “MySQL Server Logs”.

To achieve better performance, you can specify the following options differently for each server, to spread the load between several physical disks:

- `--tmpdir=dir_name`
- `--bdb-tmpdir=dir_name`

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

If you have multiple MySQL installations in different locations, you can specify the base directory for each installation with the `--basedir=dir_name` option. This causes each instance to automatically use a different data directory, log files, and PID file because the default for each of those parameters is relative to the base directory. In that case, the only other options you need to specify are the `--socket` and `--port` options. Suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--basedir` option to pass to `mysqld`, and you need specify only the `--socket` and `--port` options to `mysqld_safe`.

As discussed in the following sections, it is possible to start additional servers by specifying appropriate command options or by setting environment variables. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` option is useful for this purpose.

5.5.1 Setting Up Multiple Data Directories

Each MySQL Instance on a machine should have its own data directory. The location is specified using the `--datadir=dir_name` option.

There are different methods of setting up a data directory for a new instance:

- Create a new data directory.
- Copy an existing data directory.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following discussion provides more detail about each method.

Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files.

Even when the preceding precautions are observed, this kind of setup works only with `MyISAM` and `MERGE` tables, and not with any of the other storage engines. Also, this warning against sharing a data directory among servers always applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*. The primary problem is that NFS is the speed bottleneck. It is not meant for such use. Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Create a New Data Directory

With this method, the data directory will be in the same state as when you first install MySQL. It will have the default set of MySQL accounts and no user data.

On Unix, initialize the data directory. See Section 2.18, “Postinstallation Setup and Testing”.

On Windows, the data directory is included in MySQL distributions. If you obtain a distribution in Windows Zip archive format, you can unpack it into a temporary location, then copy the `data` directory from this location to where you are setting up the new instance.

Copy an Existing Data Directory

With this method, any MySQL accounts or user data present in the data directory are carried over to the new data directory.

1. Stop the existing MySQL instance using the data directory. This must be a clean shutdown so that the instance flushes any pending changes to disk.
2. Copy the data directory to the location where the new data directory should be.
3. Copy the `my.cnf` or `my.ini` option file used by the existing instance. This serves as a basis for the new instance.
4. Modify the new option file so that any pathnames referring to the original data directory refer to the new data directory. Also, modify any other options that must be unique per instance, such as the TCP/IP port number and the log files. For a list of parameters that must be unique per instance, see Section 5.5, “Running Multiple MySQL Instances on One Machine”.
5. Start the new instance, telling it to use the new option file.

5.5.2 Running Multiple MySQL Instances on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters, or by installing several servers as Windows services and running them

This documentation is for an older version. If you're
This documentation is for an older version. If you're

that way. General instructions for running MySQL from the command line or as a service are given in Section 2.10, “Installing MySQL on Microsoft Windows”. The following sections describe how to start each server with different values for those options that must be unique per server, such as the data directory. These options are listed in Section 5.5, “Running Multiple MySQL Instances on One Machine”.

5.5.2.1 Starting Multiple MySQL Instances at the Windows Command Line

The procedure for starting a single MySQL server manually from the command line is described in Section 2.10.4.5, “Starting MySQL from the Windows Command Line”. To start multiple servers this way, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` option when you run it.

Suppose that you want to run `mysqld` on port 3307 with a data directory of `C:\mydata1`, and `mysqld-debug` on port 3308 with a data directory of `C:\mydata2`. Use this procedure:

1. Make sure that each data directory exists, including its own copy of the `mysql` database that contains the grant tables.

2. Create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

   ```
   [mysqld]
   datadir = C:/mydata1
   port = 3307
   ```

   Create a second file named `C:\my-opts2.cnf` that looks like this:

   ```
   [mysqld]
   datadir = C:/mydata2
   port = 3308
   ```

3. Use the `--defaults-file` option to start each server with its own option file:

   ```
   C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
   C:\> C:\mysql\bin\mysqld-debug --defaults-file=C:\my-opts2.cnf
   ```

   Each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 --host=127.0.0.1 --user=root --password shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 --host=127.0.0.1 --user=root --password shutdown
```

Servers configured as just described permit clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to permit named-pipe connections, use the `mysqld-nt` or `mysqld-debug` server and specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Modify `C:\my-opts2.cnf` similarly for use by the second server. Then start the servers as described previously.

A similar procedure applies for servers that you want to permit shared-memory connections. Enable such connections with the `--shared-memory` option and specify a unique shared-memory name for each server with the `--shared-memory-base-name` option.

5.5.2.2 Starting Multiple MySQL Instances as Windows Services

On Windows, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in Section 2.10.4.7, “Starting MySQL as a Windows Service”.

To set up multiple MySQL services, you must make sure that each instance uses a different service name in addition to the other parameters that must be unique per instance.

For the following instructions, suppose that you want to run the `mysqld-nt` server from two different versions of MySQL that are installed at `C:\mysql-4.1.24` and `C:\mysql-5.0.96`, respectively. (This might be the case if you're running 4.1.24 as your production server, but also want to conduct tests using 5.0.96.)

To install MySQL as a Windows service, use the `--install` or `--install-manual` option. For information about these options, see Section 2.10.4.7, “Starting MySQL as a Windows Service”.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, shut down and remove any existing MySQL services.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 4.1.24 `mysqld-nt` using the service name of `mysqld1` and the 5.0.96 `mysqld-nt` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 4.1.24 and the `[mysqld2]` group for 5.0.96. For example, you can set up `C:\my.cnf` like this:

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-4.1.24
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.0.96
port = 3308
enable-named-pipe
socket = mypipe2
```

  Install the services as follows, using the full server path names to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-4.1.24\bin\mysqld-nt --install mysqld1
C:\> C:\mysql-5.0.96\bin\mysqld-nt --install mysqld2
```

  To start the services, use the services manager, or use `NET START` with the appropriate service names:

```
C:\> NET START mysqld1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
C:\> NET START mysqld2
```

To stop the services, use the services manager, or use `NET STOP` with the appropriate service names:

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

  With this approach, to specify options for the 4.1.24 `mysqld-nt`, create a file `C:\my-opts1.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-4.1.24
  port = 3307
  enable-named-pipe
  socket = mypipe1
  ```

  For the 5.0.96 `mysqld-nt`, create a file `C:\my-opts2.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-5.0.96
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Install the services as follows (enter each command on a single line):

  ```
  C:\> C:\mysql-4.1.24\bin\mysqld-nt --install mysqld1
            --defaults-file=C:\my-opts1.cnf
  C:\> C:\mysql-5.0.96\bin\mysqld-nt --install mysqld2
            --defaults-file=C:\my-opts2.cnf
  ```

  When you install a MySQL server as a service and use a `--defaults-file` option, the service name must precede the option.

  After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `mysqld --remove` for each one, specifying a service name following the `--remove` option. If the service name is the default (`MySQL`), you can omit it.

5.5.3 Running Multiple MySQL Instances on Unix

One way is to run multiple MySQL instances on Unix is to compile different servers with different default TCP/IP ports and Unix socket files so that each one listens on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 4.1 server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new 5.0.96 server to have different operating parameters, use a `configure` command something like this:

```
shell> ./configure --with-tcp-port=port_number \
            --with-unix-socket-path=file_name \
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
              --prefix=/usr/local/mysql-5.0.96
```

Here, *`port_number`* and *`file_name`* must be different from the default TCP/IP port number and Unix socket file path name, and the `--prefix` value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket file name:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

If you specify `localhost` as the host name, `mysqladmin` defaults to using a Unix socket file connection rather than TCP/IP. To explicitly specify the connection protocol, use the `--protocol={TCP|SOCKET|PIPE|MEMORY}` option.

You need not compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a `--datadir=`*`dir_name`* option to `mysqld_safe` so that the server uses a different data directory.

Alternatively, put the options for each server in a different option file, then start each server using a `--defaults-file` option that specifies the path to the appropriate option file. For example, if the option files for two server instances are named `/usr/local/mysql/my.cnf` and `/usr/local/mysql/my.cnf2`, start the servers like this: command:

```
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf2
```

Another way to achieve a similar effect is to use environment variables to set the Unix socket file name and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 2.21, “Environment Variables”, includes a list of other environment variables you can use to affect MySQL programs.

On Unix, the `mysqld_multi` script provides another way to start multiple servers. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

5.5.4 Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect using TCP/IP to a remote server, with `--host=127.0.0.1 --port=port_number` to connect using TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server using a Unix socket file or a Windows named pipe.

- Start the client with `--protocol=TCP` to connect using TCP/IP, `--protocol=SOCKET` to connect using a Unix socket file, `--protocol=PIPE` to connect using a named pipe, or `--protocol=MEMORY` to connect using shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` option to specify the shared-memory name. Shared-memory connections are supported only on Windows.

- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 2.21, “Environment Variables”.

- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.2.6, “Using Option Files”.

- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See Section 20.6.7, “C API Function Descriptions”.

- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;"
       . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
$dbh = DBI->connect($dsn, $user, $password);
```

  See Section 20.8, “MySQL Perl API”.

  Other programming interfaces may provide similar capabilities for reading option files.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 6 Security

Table of Contents

When thinking about security within a MySQL installation, you should consider a wide range of possible topics and how they affect the security of your MySQL server and related applications:

- General factors that affect security. These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others. See Section 6.1, “General Security Issues”.
- Security of the installation itself. The data files, log files, and the all the application files of your installation should be protected to ensure that they are not readable or writable by unauthorized parties. For more information, see Section 2.18, “Postinstallation Setup and Testing”.
- Access control and security within the database system itself, including the users and databases granted with access to the databases, views and stored programs in use within the database. For more information, see Section 6.2, “The MySQL Access Privilege System”, and Section 6.3, “MySQL User Account Management”.
- Network security of MySQL and your system. The security is related to the grants for individual users, but you may also wish to restrict MySQL so that it is available only locally on the MySQL server host, or to a limited set of other hosts.
- Ensure that you have adequate and appropriate backups of your database files, configuration and log files. Also be sure that you have a recovery solution in place and test that you are able to successfully recover the information from your backups. See Chapter 7, *Backup and Recovery*.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

6.1 General Security Issues

This section describes general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Section 6.2, “The MySQL Access Privilege System”.

For answers to some questions that are often asked about MySQL Server security issues, see Section A.9, “MySQL 5.0 FAQ: Security”.

6.1.1 Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, it is necessary to consider fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` database!** This is critical.
- Learn how the MySQL access privilege system works (see Section 6.2, “The MySQL Access Privilege System”). Use the `GRANT` and `REVOKE` statements to control access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

  Checklist:

  - Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See Section 2.18.4, “Securing the Initial MySQL Accounts”.
  - Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.
- Do not store cleartext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `SHA1()`, `MD5()`, or some other one-way hashing function and store the hash value.

  To prevent password recovery using rainbow tables, do not use these functions on a plain password; instead, choose some string to be used as a salt, and use hash(hash(password)+salt) values.
- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Four score and seven years ago” results in a password of “Fsasya”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence. In this case, you can additionally substitute digits

This documentation is for an older version. If you're
This documentation is for an older version. If you're

for the number words to obtain the phrase “4 score and 7 years ago”, yielding the password “4sa7ya” which is even more difficult to guess.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

  Checklist:

  - Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. As a simple way to check whether your MySQL port is open, try the following command from some remote machine, where *`server_host`* is the host name or IP address of the host on which your MySQL server runs:

    ```
    shell> telnet server_host 3306
    ```

    If `telnet` hangs or the connection is refused, the port is blocked, which is how you want it to be. If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open.

- Applications that access MySQL should not trust any data entered by users, and should be written using proper defensive programming techniques. See Section 6.1.7, “Client Programming Security Guidelines”.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the `tcpdump` and `strings` utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

  ```
  shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
  ```

  This works under Linux and should work with small modifications under other systems.

Warning

If you do not see cleartext data, this does not always mean that the information actually is encrypted. If you need high security, consult with a security expert.

6.1.2 Keeping Passwords Secure

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable end users and administrators to keep these passwords secure and avoid exposing them. There is also a discussion of how MySQL uses password hashing internally.

6.1.2.1 End-User Guidelines for Password Security

MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Use a `-pyour_pass` or `--password=your_pass` option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

This is convenient *but insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If your operating environment is set up to display your current command in the title bar of your terminal window, the password remains visible as long as the command is running, even if the command has scrolled out of view in the window content area.

- Use the `-p` or `--password` option on the command line with no password value specified. In this case, the client program solicits the password interactively:

```
shell> mysql -u francis -p db_name
Enter password: ********
```

The "`*`" characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix, you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=your_pass
```

To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

```
shell> chmod 600 .my.cnf
```

To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` option, where `file_name` is the full path name to the file. For example:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

Section 4.2.6, "Using Option Files", discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. See Section 2.21, "Environment Variables".

This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of `ps` include an option to display the environment of running processes. On some systems, if you set `MYSQL_PWD`, your password is exposed to any other user who runs `ps`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Even on systems without such a version of `ps`, it is unwise to assume that there are no other methods by which users can examine process environments.

On Unix, the `mysql` client writes a record of executed statements to a history file (see Section 4.5.1.3, “mysql Logging”). By default, this file is named `.mysql_history` and is created in your home directory. Passwords can be written as plain text in SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, so if you use these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the `.my.cnf` file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved will contain MySQL passwords entered on the command line. For example, `bash` uses `~/.bash_history`. Any such file should have a restrictive access mode.

6.1.2.2 Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the `mysql.user` table. Access to this table should never be granted to any nonadministrative accounts.

A user who has access to modify the plugin directory (the value of the `plugin_dir` system variable) or the `my.cnf` file that specifies the location of the plugin directory can replace plugins and modify the capabilities provided by plugins.

Files such as log files to which passwords might be written should be protected. See Section 6.1.2.3, “Passwords and Logging”.

6.1.2.3 Passwords and Logging

Passwords can be written as plain text in SQL statements such as `CREATE USER`, `GRANT`, `SET PASSWORD`, and statements that invoke the `PASSWORD()` function. If such statements are logged by the MySQL server as written, passwords in them become visible to anyone with access to the logs. This applies to the general query log, the slow query log, and the binary log (see Section 5.4, “MySQL Server Logs”).

To guard log files against unwarranted exposure, locate them in a directory that restricts access to the server and the database administrator.

Replication slaves store the password for the replication master in the `master.info` file. Retrict this file to be accessible only to the database administrator.

Use a restricted access mode to protect database backups that include log files containing passwords.

6.1.2.4 Password Hashing in MySQL

MySQL lists user accounts in the `user` table of the `mysql` database. Each MySQL account can be assigned a password, although the `user` table does not store the cleartext version of the password, but a hash value computed from it.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account the client wants to use.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- After the client connects, it can (if it has sufficient privileges) set or change the password hash for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` function to generate a password hash, or by using a password-generating statement (`CREATE USER`, `GRANT`, or `SET PASSWORD`).

In other words, the server *checks* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` function or uses a password-generating statement to set or change a password.

Password hashing methods in MySQL have the history described following. These changes are illustrated by changes in the result from the `PASSWORD()` function that computes password hash values and in the structure of the `user` table where passwords are stored.

Note

This discussion contrasts 4.1 behavior with pre-4.1 behavior, but the 4.1 behavior described here actually begins with 4.1.1. MySQL 4.1.0 is an “odd” release because it has a slightly different method than that implemented in 4.1.1 and up. Differences between 4.1.0 and more recent versions are described further in MySQL 3.23, 4.0, 4.1 Reference Manual.

The Original (Pre-4.1) Hashing Method

The original hashing method produced a 16-byte string. Such hashes look like this:

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

To store account passwords, the `Password` column of the `user` table was at this point 16 bytes long.

The 4.1 Hashing Method

MySQL 4.1 introduced password hashing that provided better security and reduced the risk of passwords being intercepted. There were several aspects to this change:

- Different format of password values produced by the `PASSWORD()` function
- Widening of the `Password` column
- Control over the default hashing method
- Control over the permitted hashing methods for clients attempting to connect to the server

The changes in MySQL 4.1 took place in two stages:

- MySQL 4.1.0 used a preliminary version of the 4.1 hashing method. This method was short lived and the following discussion says nothing more about it.
- In MySQL 4.1.1, the hashing method was modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
```

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

To accommodate longer password hashes, the `Password` column in the `user` table was changed at this point to be 41 bytes, its current length.

A widened `Password` column can store password hashes in both the pre-4.1 and 4.1 formats. The format of any given hash value can be determined two ways:

- The length: 4.1 and pre-4.1 hashes are 41 and 16 bytes, respectively.
- Password hashes in the 4.1 format always begin with a "`*`" character, whereas passwords in the pre-4.1 format never do.

To permit explicit generation of pre-4.1 password hashes, two additional changes were made:

- The `OLD_PASSWORD()` function was added, which returns hash values in the 16-byte format.
- For compatibility purposes, the `old_passwords` system variable was added, to enable DBAs and applications control over the hashing method. The default `old_passwords` value of 0 causes hashing to use the 4.1 method (41-byte hash values), but setting `old_passwords=1` causes hashing to use the pre-4.1 method. In this case, `PASSWORD()` produces 16-byte values and is equivalent to `OLD_PASSWORD()`

To permit DBAs control over how clients are permitted to connect, the `secure_auth` system variable was added. Starting the server with this variable disabled or enabled permits or prohibits clients to connect using the older pre-4.1 password hashing method. Before MySQL 5.6.5, `secure_auth` is disabled by default. As of 5.6.5, `secure_auth` is enabled by default to promote a more secure default configuration. (DBAs can disable it at their discretion, but this is not recommended.)

In addition, the `mysql` client supports a `--secure-auth` option that is analogous to `secure_auth`, but from the client side. It can be used to prevent connections to less secure accounts that use pre-4.1 password hashing. This option is disabled by default before MySQL 5.6.7, enabled thereafter.

Compatibility Issues Related to Hashing Methods

The widening of the `Password` column in MySQL 4.1 from 16 bytes to 41 bytes affects installation or upgrade operations as follows:

- If you perform a new installation of MySQL, the `Password` column is made 41 bytes long automatically.
- Upgrades from MySQL 4.1 or later to current versions of MySQL should not give rise to any issues in regard to the `Password` column because both versions use the same column length and password hashing method.
- For upgrades from a pre-4.1 release to 4.1 or later, you must upgrade the system tables after upgrading. (See Section 4.4.9, "`mysql_upgrade` — Check Tables for MySQL Upgrade".)

The 4.1 hashing method is understood only by MySQL 4.1 (and newer) servers and clients, which can result in some compatibility problems. A 4.1 or newer client can connect to a pre-4.1 server, because the client understands both the pre-4.1 and 4.1 password hashing methods. However, a pre-4.1 client that attempts to connect to a 4.1 or newer server may run into difficulties. For example, a 4.0 `mysql` client may fail with the following error message:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

This phenomenon also occurs for attempts to use the older PHP `mysql` extension after upgrading to MySQL 4.1 or newer. (See Common Problems with MySQL and PHP.)

The following discussion describes the differences between the pre-4.1 and 4.1 hashing methods, and what you should do if you upgrade your server but need to maintain backward compatibility with pre-4.1 clients. (However, permitting connections by old clients is not recommended and should be avoided if possible.) Additional information can be found in Section B.5.2.4, “Client does not support authentication protocol”. This information is of particular importance to PHP programmers migrating MySQL databases from versions older than 4.1 to 4.1 or higher.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the `Password` column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:
  - Pre-4.1 clients can connect, but because they know only about the pre-4.1 hashing method, they can authenticate only using accounts that have short hashes.
  - 4.1 and later clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash
- 4.1 or later client authenticating with short password hash
- 4.1 or later client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `old_passwords` system variable. A 4.1 or later server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and `old_passwords` must not be set to 1.

Those conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using the `PASSWORD()` function or a password-generating statement. This is the behavior that occurs if you have upgraded from a version of MySQL older than 4.1 to 4.1 or later but have not yet run the `mysql_upgrade` program to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, the `PASSWORD()` function and password-generating statements generate long hashes unless the server was started with the `old_passwords` system variable set to 1 to force the server to generate short password hashes instead.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The purpose of the `old_passwords` system variable is to permit backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option does not affect authentication (4.1 and later clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that permitted to occur, the account could no longer be used by pre-4.1 clients. With `old_passwords` disabled, the following undesirable scenario is possible:

- An old pre-4.1 client connects to an account that has a short password hash.
- The client changes its own password. With `old_passwords` disabled, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the 4.1 hashing method during authentication. (Once an account has a long password hash in the user table, only 4.1 and later clients can authenticate for it because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is problematic to run a 4.1 or newer server without `old_passwords` set to 1. By running the server with `old_passwords=1`, password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of `old_passwords=1` is that any passwords created or changed use short hashes, even for 4.1 or later clients. Thus, you lose the additional security provided by long password hashes. To create an account that has a long hash (for example, for use by 4.1 clients) or to change an existing account to use a long password hash, an administrator can set the session value of `old_passwords` set to 0 while leaving the global value set to 1:

```
mysql> SET @@session.old_passwords = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@session.old_passwords, @@global.old_passwords;
+--------------------------+------------------------+
| @@session.old_passwords | @@global.old_passwords |
+--------------------------+------------------------+
|                        0 |                      1 |
+--------------------------+------------------------+
1 row in set (0.00 sec)

mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'newpass';
Query OK, 0 rows affected (0.03 sec)

mysql> SET PASSWORD FOR 'existinguser'@'localhost' = PASSWORD('existingpass');
Query OK, 0 rows affected (0.00 sec)
```

The following scenarios are possible in MySQL 4.1 or later. The factors are whether the `Password` column is short or long, and, if long, whether the server is started with `old_passwords` enabled or disabled.

Scenario 1: Short `Password` column in user table:

- Only short hashes can be stored in the `Password` column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The value of `old_passwords` is irrelevant because with a short `Password` column, the server generates only short password hashes anyway.

This scenario occurs when a pre-4.1 MySQL installation has been upgraded to 4.1 or later but `mysql_upgrade` has not been run to upgrade the system tables in the `mysql` database. (This is not a recommended configuration because it does not permit use of more secure 4.1 password hashing.)

Scenario 2: Long `Password` column; server started with `old_passwords=1`:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate for accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, newly created accounts have short password hashes because `old_passwords=1` prevents generation of long hashes. Also, if you create an account with a long hash before setting `old_passwords` to 1, changing the account's password while `old_passwords=1` results in the account being given a short password, causing it to lose the security benefits of a longer hash.

To create a new account that has a long password hash, or to change the password of any existing account to use a long hash, first set the session value of `old_passwords` set to 0 while leaving the global value set to 1, as described previously.

In this scenario, the server has an up to date `Password` column, but is running with the default password hashing method set to generate pre-4.1 hash values. This is not a recommended configuration but may be useful during a transitional period in which pre-4.1 clients and passwords are upgraded to 4.1 or later. When that has been done, it is preferable to run the server with `old_passwords=0` and `secure_auth=1`.

Scenario 3: Long `Password` column; server started with `old_passwords=0`:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.
- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made using the `PASSWORD()` function or a password-generating statement results in the account being given a long password hash. From that point on, no pre-4.1 client can connect to the server using that account. The client must upgrade to 4.1 or later.

If this is a problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` is useful for situations in which you explicitly want to generate a short hash.

The disadvantages for each of the preceding scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, `old_passwords=1` prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes unless you take care to change the session value of `old_passwords` to 0 first.

In scenario 3, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()`.

The best way to avoid compatibility problems related to short password hashes is to not use them:

- Upgrade all client programs to MySQL 4.1 or later.
- Run the server with `old_passwords=0`.
- Reset the password for any account with a short password hash to use a long password hash.
- For additional security, run the server with `secure_auth=1`.

6.1.2.5 Implications of Password Hashing Changes in MySQL 4.1 for Application Programs

An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use `PASSWORD()` to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` for their own purposes anyway.

If you upgrade to 4.1 or later from a pre-4.1 version of MySQL and run the server under conditions where it generates long password hashes, an application using `PASSWORD()` for its own passwords breaks. The recommended course of action in such cases is to modify the application to use another function, such as `SHA1()` or `MD5()`, to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` function, which is provided for generate short hashes in the old format. However, you should note that `OLD_PASSWORD()` may one day no longer be supported.

If the server is running with `old_passwords=1`, it generates short hashes and `OLD_PASSWORD()` is equivalent to `PASSWORD()`.

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see MySQL and PHP.

6.1.3 Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See Section 6.1.2.4, “Password Hashing in MySQL”, for a discussion of the different password handling methods.)

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See Section 6.3.6, “Using Secure Connections”. Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at http://www.openssh.org/, and a commercial SSH client at http://www.ssh.com/.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if `other_user` has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

  For a discussion of methods for setting passwords, see Section 6.3.5, “Assigning Account Passwords”.

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` option.

  `mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

  ```
  [mysqld]
  user=mysql
  ```

  This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see Section 6.1.5, “How to Run MySQL as a Normal User”.

  Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` user name in the `user` table. *User names for MySQL accounts have nothing to do with user names for Unix accounts*.

- Do not grant the `FILE` privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon. This includes the server's data directory containing the files that implement the privilege tables. To make `FILE`-privilege operations a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

  The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could

This documentation is for an older version. If you're
This documentation is for an older version. If you're

be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

To limit the location in which files can be read and written, set the `secure_file_priv` system to a specific directory. See Section 5.1.4, “Server System Variables”.

- Do not grant the `PROCESS` or `SUPER` privilege to nonadministrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users such as `UPDATE user SET password=PASSWORD('not_secure')`.

  `mysqld` reserves an extra connection for users who have the `SUPER` privilege, so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

  The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run `mysqld` as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”.

- Stored programs and views should be written using the security guidelines discussed in Section 18.5, “Access Control for Stored Programs and Views”.

- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.

- If you want to restrict the number of connections permitted to a single account, you can do so by setting the `max_user_connections` variable in `mysqld`. The `GRANT` statement also supports resource control options for limiting the extent of server use permitted to an account. See Section 13.7.1.3, “GRANT Syntax”.

- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

6.1.4 Security-Related mysqld Options and Variables

The following table shows `mysqld` options and system variables that affect security. For descriptions of each of these, see Section 5.1.3, “Server Command Options”, and Section 5.1.4, “Server System Variables”.

Table 6.1 Security Option/Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
allow-suspicious-udfs	Yes	Yes				
automatic_sp_privileges			Yes		Global	Yes
chroot	Yes	Yes				
des-key-file	Yes	Yes				
local_infile			Yes		Global	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
old_passwords			Yes		Both	Yes
safe-show-database	Yes	Yes				
safe-user-create	Yes	Yes				
secure-auth	Yes	Yes			Global	Yes
- *Variable*: secure_auth			Yes		Global	Yes
secure-file-priv	Yes	Yes			Global	No
- *Variable*: secure_file_priv			Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-name-resolve	Yes	Yes				
skip-networking	Yes	Yes			Global	No
- *Variable*: skip_networking			Yes		Global	No
skip-show-database	Yes	Yes			Global	No
- *Variable*: skip_show_database			Yes		Global	No

6.1.5 How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Unix, the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user *`user_name`*, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).
2. Change the database directories and files so that *`user_name`* has privileges to read and write files in them (you might need to do this as the Unix `root` user):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   If you do not do this, the server will not be able to access databases or tables when it runs as *`user_name`*.

   If directories or files within the MySQL data directory are symbolic links, `chown -R` might not follow symbolic links for you. If it does not, you will also need to follow those links and change the directories and files they point to.
3. Start the server as user *`user_name`*. Another alternative is to start `mysqld` as the Unix `root` user and use the `--user=`*`user_name`* option. `mysqld` starts up, then switches to run as the Unix user *`user_name`* before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the user name by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
[mysqld]
user=user_name
```

If your Unix machine itself is not secured, you should assign passwords to the MySQL `root` accounts in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 2.18.4, “Securing the Initial MySQL Accounts”.

6.1.6 Security Issues with LOAD DATA LOCAL

The `LOAD DATA` statement can load a file that is located on the server host, or it can load a file that is located on the client host when the `LOCAL` keyword is specified.

There are two potential security issues with supporting the `LOCAL` version of `LOAD DATA` statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD DATA` statement. Such a server could access any file on the client host to which the client user has read access.
- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

To deal with these problems, `LOAD DATA LOCAL` works like this:

- By default, all MySQL clients and libraries in binary distributions are compiled with the `--enable-local-infile` option.
- If you build MySQL from source but do not invoke `configure` with the `--enable-local-infile` option, `LOAD DATA LOCAL` cannot be used by any client unless it is written explicitly to invoke `mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0)`. See Section 20.6.7.49, “mysql_options()”.
- You can disable all `LOAD DATA LOCAL` statements from the server side by starting `mysqld` with the `--local-infile=0` option.
- For the `mysql` command-line client, enable `LOAD DATA LOCAL` by specifying the `--local-infile[=1]` option, or disable it with the `--local-infile=0` option. For `mysqlimport`, local data file loading is off by default; enable it with the `--local` or `-L` option. In any case, successful use of a local load operation requires that the server permits it.
- If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add the `local-infile=1` option to that group. However, to keep this from causing problems for programs that do not understand `local-infile`, specify it using the `loose-` prefix:

```
[client]
loose-local-infile=1
```

- If `LOAD DATA LOCAL` is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6.1.7 Client Programming Security Guidelines

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like "`; DROP DATABASE mysql;`". This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value `234`, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts. See Section 5.1.7, "Server SQL Modes".
- Try to enter single and double quotation marks ("`'`" and "`"`") in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding `%22` ("`"`"), `%23` ("`#`"), and `%27` ("`'`") to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:

- MySQL C API: Use the `mysql_real_escape_string()` API call.
- MySQL++: Use the `escape` and `quote` modifiers for query streams.
- PHP: Use either the `mysqli` or `pdo_mysql` extensions, and not the older `ext/mysql` extension. The preferred API's support the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders. See also Choosing an API.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the older `ext/mysql` extension must be used, then for escaping use the `mysql_real_escape_string()` function and not `mysql_escape_string()` or `addslashes()` because only `mysql_real_escape_string()` is character set-aware; the other functions can be "bypassed" when using (invalid) multibyte character sets.

- Perl DBI: Use placeholders or the `quote()` method.
- Ruby DBI: Use placeholders or the `quote()` method.
- Java JDBC: Use a `PreparedStatement` object and placeholders.

Other programming interfaces might have similar capabilities.

6.2 The MySQL Access Privilege System

The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

The user interface to the MySQL privilege system consists of SQL statements such as `CREATE USER`, `GRANT`, and `REVOKE`. See Section 13.7.1, "Account Management Statements".

Internally, the server stores privilege information in the grant tables of the `mysql` database (that is, in the database named `mysql`). The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the user name you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`. To see what privileges a given account has, use the `SHOW GRANTS` statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';
SHOW GRANTS FOR 'joe'@'home.example.com';
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL access control involves two stages when you run a client program that connects to the server:

Stage 1: The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT` privilege for the table or the `DROP` privilege for the database.

For a more detailed description of what happens during each stage, see Section 6.2.4, “Access Control, Stage 1: Connection Verification”, and Section 6.2.5, “Access Control, Stage 2: Request Verification”.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see Section 6.2.6, “When Privilege Changes Take Effect”.

For general security-related advice, see Section 6.1, “General Security Issues”. For help in diagnosing privilege-related problems, see Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”.

6.2.1 Privileges Provided by MySQL

MySQL provides privileges that apply in different contexts and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.
- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases).

Information about account privileges is stored in the `user`, `db`, `host`, `tables_priv`, `columns_priv`, and `procs_priv` tables in the `mysql` database (see Section 6.2.2, “Grant Tables”). The MySQL server reads the contents of these tables into memory when it starts and reloads them under the circumstances indicated in Section 6.2.6, “When Privilege Changes Take Effect”. Access-control decisions are based on the in-memory copies of the grant tables.

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to have the current structure whenever you update to a new version of MySQL. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

The following table shows the privilege names used at the SQL level in the `GRANT` and `REVOKE` statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

Table 6.2 Permissible Privileges for GRANT and REVOKE

Privilege	Column	Context
`CREATE`	`Create_priv`	databases, tables, or indexes
`DROP`	`Drop_priv`	databases, tables, or views

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Privilege	Column	Context
GRANT OPTION	Grant_priv	databases, tables, or stored routines
LOCK TABLES	Lock_tables_priv	databases
REFERENCES	References_priv	databases or tables
ALTER	Alter_priv	tables
DELETE	Delete_priv	tables
INDEX	Index_priv	tables
INSERT	Insert_priv	tables or columns
SELECT	Select_priv	tables or columns
UPDATE	Update_priv	tables or columns
CREATE TEMPORARY TABLES	Create_tmp_table_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
FILE	File_priv	file access on server host
CREATE USER	Create_user_priv	server administration
PROCESS	Process_priv	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration
ALL [PRIVILEGES]		server administration
USAGE		server administration

The following list provides a general description of each privilege available in MySQL. Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- The ALL or ALL PRIVILEGES privilege specifier is shorthand. It stands for “all privileges available at a given privilege level” (except GRANT OPTION). For example, granting ALL at the global or table level grants all global privileges or all table-level privileges.
- The ALTER privilege enables use of ALTER TABLE to change the structure of tables. ALTER TABLE also requires the CREATE and INSERT privileges. Renaming a table requires ALTER and DROP on the old table, CREATE, and INSERT on the new table.
- The ALTER ROUTINE privilege is needed to alter or drop stored routines (procedures and functions). This privilege was added in MySQL 5.0.3.
- The CREATE privilege enables creation of new databases and tables.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `CREATE ROUTINE` privilege is needed to create stored routines (procedures and functions). This privilege was added in MySQL 5.0.3.

- The `CREATE TEMPORARY TABLES` privilege enables the creation of temporary tables using the `CREATE TEMPORARY TABLE` statement.

  However, other operations on a temporary table, such as `INSERT`, `UPDATE`, or `SELECT`, require additional privileges for those operations for the database containing the temporary table, or for the nontemporary table of the same name.

  To keep privileges for temporary and nontemporary tables separate, a common workaround for this situation is to create a database dedicated to the use of temporary tables. Then for that database, a user can be granted the `CREATE TEMPORARY TABLES` privilege, along with any other privileges required for temporary table operations done by that user.

- The `CREATE USER` privilege enables use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`. This privilege was added in MySQL 5.0.3.

- The `CREATE VIEW` privilege enables use of `CREATE VIEW`. This privilege was added in MySQL 5.0.1.

- The `DELETE` privilege enables rows to be deleted from tables in a database.

- The `DROP` privilege enables you to drop (remove) existing databases and tables. *If you grant the `DROP` privilege for the `mysql` database to a user, that user can drop the database in which the MySQL access privileges are stored.*

- The `EXECUTE` privilege is required to execute stored routines (procedures and functions). This privilege was added in MySQL 5.0.0 but did not become operational until MySQL 5.0.3.

- The `FILE` privilege gives you permission to read and write files on the server host using the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. A user who has the `FILE` privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The `FILE` privilege also enables the user to create new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables. As a security measure, the server will not overwrite existing files.

  To limit the location in which files can be read and written, set the `secure_file_priv` system to a specific directory. See Section 5.1.4, “Server System Variables”.

- The `GRANT OPTION` privilege enables you to give to other users or remove from other users those privileges that you yourself possess.

- The `INDEX` privilege enables you to create or drop (remove) indexes. `INDEX` applies to existing tables. If you have the `CREATE` privilege for a table, you can include index definitions in the `CREATE TABLE` statement.

- The `INSERT` privilege enables rows to be inserted into tables in a database. `INSERT` is also required for the `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` table-maintenance statements.

- The `LOCK TABLES` privilege enables the use of explicit `LOCK TABLES` statements to lock tables for which you have the `SELECT` privilege. This includes the use of write locks, which prevents other sessions from reading the locked table.

- The `PROCESS` privilege pertains to display of information about the threads executing within the server (that is, information about the statements being executed by sessions). The privilege enables use of

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`SHOW PROCESSLIST` or `mysqladmin processlist` to see threads belonging to other accounts; you can always see your own threads.

- The `REFERENCES` privilege is unused.
- The `RELOAD` privilege enables use of the `FLUSH` statement. It also enables `mysqladmin` commands that are equivalent to `FLUSH` operations: `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh`, and `reload`.

  The `reload` command tells the server to reload the grant tables into memory. `flush-privileges` is a synonym for `reload`. The `refresh` command closes and reopens the log files and flushes all tables. The other `flush-`*`xxx`* commands perform functions similar to `refresh`, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, `flush-logs` is a better choice than `refresh`.
- The `REPLICATION CLIENT` privilege enables the use of `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`.
- The `REPLICATION SLAVE` privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server.
- The `SELECT` privilege enables you to select rows from tables in a database. `SELECT` statements require the `SELECT` privilege only if they actually retrieve rows from a table. Some `SELECT` statements do not access tables and can be executed without permission for any database. For example, you can use `SELECT` as a simple calculator to evaluate expressions that make no reference to tables:

  ```
  SELECT 1+1;
  SELECT PI()*2;
  ```

  The `SELECT` privilege is also needed for other statements that read column values. For example, `SELECT` is needed for columns referenced on the right hand side of *`col_name`*=*`expr`* assignment in `UPDATE` statements or for columns named in the `WHERE` clause of `DELETE` or `UPDATE` statements.
- The `SHOW DATABASES` privilege enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` option. Note that *any* global privilege is a privilege for the database.
- The `SHOW VIEW` privilege enables use of `SHOW CREATE VIEW`. This privilege was added in MySQL 5.0.1.
- The `SHUTDOWN` privilege enables use of the `mysqladmin shutdown` command and the `mysql_shutdown()` C API function. There is no corresponding SQL statement.
- The `SUPER` privilege enables an account to use `CHANGE MASTER TO`, `KILL` or `mysqladmin kill` to kill threads belonging to other accounts (you can always kill your own threads), `PURGE BINARY LOGS`, configuration changes using `SET GLOBAL` to modify global system variables, the `mysqladmin debug` command, enabling or disabling logging, performing updates even if the `read_only` system variable is enabled, starting and stopping replication on slave servers, specification of any account in the `DEFINER` attribute of stored programs and views, and enables you to connect (once) even if the connection limit controlled by the `max_connections` system variable is reached.

  To create or alter stored routines if binary logging is enabled, you may also need the `SUPER` privilege, as described in Section 18.6, “Binary Logging of Stored Programs”.
- The `UPDATE` privilege enables rows to be updated in tables in a database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `USAGE` privilege specifier stands for “no privileges.” It is used at the global level with `GRANT` to modify account attributes such as resource limits or SSL characteristics without affecting existing account privileges.

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the `FILE` and administrative privileges:

- The `FILE` privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using `SELECT` to transfer its contents to the client host.
- The `GRANT OPTION` privilege enables users to give their privileges to other users. Two users that have different privileges and with the `GRANT OPTION` privilege are able to combine privileges.
- The `ALTER` privilege may be used to subvert the privilege system by renaming tables.
- The `SHUTDOWN` privilege can be abused to deny service to other users entirely by terminating the server.
- The `PROCESS` privilege can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- The `SUPER` privilege can be used to terminate other sessions or change how the server operates.
- Privileges granted for the `mysql` database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the `user` table `Password` column can change an account's password, and then connect to the MySQL server using that account.

6.2.2 Grant Tables

The `mysql` system database includes several grant tables that contain information about user accounts and the privileges held by them. This section describes those tables. For information about other tables in the system database, see Section 5.3, “The mysql System Database”.

Normally, to manipulate the contents of grant tables, you modify them indirectly by using account-management statements such as `CREATE USER`, `GRANT`, and `REVOKE` to set up accounts and control the privileges available to each one. See Section 13.7.1, “Account Management Statements”. The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

Note

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.

These `mysql` database tables contain grant information:

- `user`: User accounts, global privileges, and other non-privilege columns.
- `db`: Database-level privileges.
- `host`: Obsolete.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `procs_priv`: Stored procedure and function privileges.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` applies to authenticating connections made to the server from the host `thomas.loc.gov` by a client that specifies a user name of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'thomas.loc.gov'`, `'bob'` and `'reports'` applies when `bob` connects from the host `thomas.loc.gov` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.

- Privilege columns indicate which privileges a table row grants; that is, which operations it permits to be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. Section 6.2.5, “Access Control, Stage 2: Request Verification”, describes the rules for this.

The server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the `user` table indicate the user's global privileges. Any privileges granted in this table apply to *all* databases on the server.

Caution

Because any global privilege is considered a privilege for all databases, any global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine the permitted operations. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.

- The `host` table is used in conjunction with the `db` table when you want a given `db` table row to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the `Host` value empty in the user's `db` table row, then populate the `host` table with a row for each of those hosts. This mechanism is described more detail in Section 6.2.5, “Access Control, Stage 2: Request Verification”.

Note

The `host` table must be modified directly with statements such as `INSERT`, `UPDATE`, and `DELETE`. It is not affected by statements such as `GRANT` and `REVOKE` that modify the grant tables indirectly. Most MySQL installations need not use this table at all.

- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.

- The `procs_priv` table applies to stored routines (procedures and functions). A privilege granted at the routine level applies only to a single procedure or function.

The server uses the `user`, `db`, and `host` tables in the `mysql` database at both the first and second stages of access control (see Section 6.2, “The MySQL Access Privilege System”). The columns in the `user` and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`db` tables are shown here. The `host` table is similar to the `db` table but has a specialized use as described in Section 6.2.5, “Access Control, Stage 2: Request Verification”.

Table 6.3 user and db Table Columns

Table Name	`user`	`db`
Scope columns	`Host`	`Host`
	`User`	`Db`
	`Password`	`User`
Privilege columns	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`
	`Drop_priv`	`Drop_priv`
	`Grant_priv`	`Grant_priv`
	`Create_view_priv`	`Create_view_priv`
	`Show_view_priv`	`Show_view_priv`
	`Create_routine_priv`	`Create_routine_priv`
	`Alter_routine_priv`	`Alter_routine_priv`
	`Execute_priv`	`Execute_priv`
	`Create_tmp_table_priv`	`Create_tmp_table_priv`
	`Lock_tables_priv`	`Lock_tables_priv`
	`References_priv`	`References_priv`
	`Reload_priv`	
	`Shutdown_priv`	
	`Process_priv`	
	`File_priv`	
	`Show_db_priv`	
	`Super_priv`	
	`Repl_slave_priv`	
	`Repl_client_priv`	
	`Create_user_priv`	
Security columns	`ssl_type`	
	`ssl_cipher`	
	`x509_issuer`	
	`x509_subject`	
Resource control columns	`max_questions`	
	`max_updates`	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table Name	user	db
	max_connections	
	max_user_connections	

Execute_priv was present in MySQL 5.0.0, but did not become operational until MySQL 5.0.3.

The Create_view_priv and Show_view_priv columns were added in MySQL 5.0.1.

The Create_routine_priv, Alter_routine_priv, and max_user_connections columns were added in MySQL 5.0.3.

During the second stage of access control, the server performs request verification to ensure that each client has sufficient privileges for each request that it issues. In addition to the user, db, and host grant tables, the server may also consult the tables_priv and columns_priv tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

Table 6.4 tables_priv and columns_priv Table Columns

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The Timestamp and Grantor columns are unused.

For verification of requests that involve stored routines, the server may consult the procs_priv table, which has the columns shown in the following table.

Table 6.5 procs_priv Table Columns

Table Name	procs_priv
Scope columns	Host
	Db
	User
	Routine_name
	Routine_type
Privilege columns	Proc_priv
Other columns	Timestamp
	Grantor

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `procs_priv` table exists as of MySQL 5.0.3. The `Routine_type` column was added in MySQL 5.0.6. It is an `ENUM` column with values of `'FUNCTION'` or `'PROCEDURE'` to indicate the type of routine the row refers to. This column enables privileges to be granted separately for a function and a procedure with the same name.

The `Timestamp` and `Grantor` columns are set to the current timestamp and the `CURRENT_USER` value, respectively, but are otherwise unused.

Scope columns in the grant tables contain strings. The default value for each is the empty string. The following table shows the number of characters permitted in each column.

Table 6.6 Grant Table Scope Column Lengths

Column Name	Maximum Permitted Characters
`Host`	60
`User`	16
`Password`	41
`Db`	64
`Table_name`	64
`Column_name`	64
`Routine_name`	64

For access-checking purposes, comparisons of `User`, `Password`, `Db`, and `Table_name` values are case sensitive. Comparisons of `Host`, `Column_name`, and `Routine_name` values are not case sensitive.

The `user`, `db`, and `host` tables list each privilege in a separate column that is declared as `ENUM('N','Y') DEFAULT 'N'`. In other words, each privilege can be disabled or enabled, with the default being disabled.

The `tables_priv`, `columns_priv`, and `procs_priv` tables declare the privilege columns as `SET` columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 6.7 Set-Type Privilege Column Values

Table Name	Column Name	Possible Set Elements
`tables_priv`	`Table_priv`	`'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view'`
`tables_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`columns_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`procs_priv`	`Proc_priv`	`'Execute', 'Alter Routine', 'Grant'`

Only the `user` table specifies administrative privileges, such as `RELOAD` and `SHUTDOWN`. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list

This documentation is for an older version. If you're
This documentation is for an older version. If you're

these privileges in the other grant tables. Consequently, the server need consult only the `user` table to determine whether a user can perform an administrative operation.

The `FILE` privilege also is specified only in the `user` table. It is not an administrative privilege as such, but a user's ability to read or write files on the server host is independent of the database being accessed.

The server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a `FLUSH PRIVILEGES` statement or executing a `mysqladmin flush-privileges` or `mysqladmin reload` command. Changes to the grant tables take effect as indicated in Section 6.2.6, "When Privilege Changes Take Effect".

When you modify an account, it is a good idea to verify that your changes have the intended effect. To check the privileges for a given account, use the `SHOW GRANTS` statement. For example, to determine the privileges that are granted to an account with user name and host name values of `bob` and `pc84.example.com`, use this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

6.2.3 Specifying Account Names

MySQL account names consist of a user name and a host name. This enables creation of accounts for users with the same name who can connect from different hosts. This section describes how to write account names, including special values and wildcard rules.

In SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, write account names using the following rules:

- Syntax for account names is *`'user_name'@'host_name'`*.
- An account name consisting only of a user name is equivalent to *`'user_name'`*`@'%'`. For example, `'me'` is equivalent to `'me'@'%'`.
- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a *`user_name`* string containing special characters (such as "`-`"), or a *`host_name`* string containing special characters or wildcard characters (such as "`%`"); for example, `'test-user'@'%.com'`.
- Quote user names and host names as identifiers or as strings, using either backticks ("`` ` ``"), single quotation marks ("`'`"), or double quotation marks ("`"`").
- The user name and host name parts, if quoted, must be quoted separately. That is, write `'me'@'localhost'`, not `'me@localhost'`; the latter is interpreted as `'me@localhost'@'%'`.

MySQL stores account names in grant tables in the `mysql` database using separate columns for the user name and host name parts:

- The `user` table contains one row for each account. The `User` and `Host` columns store the user name and host name. This table also indicates which global privileges the account has.
- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have `User` and `Host` columns to store the account name. Each row in these tables associates with the account in the `user` table that has the same `User` and `Host` values.
- A reference to the `CURRENT_USER` or `CURRENT_USER()` function is equivalent to specifying the current client's user name and host name literally.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For additional detail about grant table structure, see Section 6.2.2, “Grant Tables”.

User names and host names have certain special values or wildcard conventions, as described following.

A user name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as `''@'localhost'`.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address. The name `'localhost'` indicates the local host. The IP address `'127.0.0.1'` indicates the loopback interface.
- You can use the wildcard characters “`%`” and “`_`” in host name or IP address values. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, a host value of `'%'` matches any host name, whereas a value of `'%.mysql.com'` matches any host in the `mysql.com` domain. `'192.168.1.%'` matches any host in the 192.168.1 class C network.

  Because you can use IP wildcard values in host values (for example, `'192.168.1.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host `192.168.1.somewhere.com`. To foil such attempts, MySQL disallows matching on host names that start with digits and a dot. Thus, if you have a host named something like `1.2.example.com`, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.
- For a host value specified as an IP address, you can specify a netmask indicating how many address bits to use for the network number. The syntax is *`host_ip/netmask`*. For example:

```
CREATE USER 'david'@'192.58.197.0/255.255.255.0';
```

  This enables `david` to connect from any client host having an IP address *`client_ip`* for which the following condition is true:

```
client_ip & netmask = host_ip
```

  That is, for the `CREATE USER` statement just shown:

```
client_ip & 255.255.255.0 = 192.58.197.0
```

  IP addresses that satisfy this condition and can connect to the MySQL server are those in the range from `192.58.197.0` to `192.58.197.255`.

  A netmask typically begins with bits set to 1, followed by bits set to 0. Examples:

  - `192.0.0.0/255.0.0.0`: Any host on the 192 class A network
  - `192.168.0.0/255.255.0.0`: Any host on the 192.168 class B network
  - `192.168.1.0/255.255.255.0`: Any host on the 192.168.1 class C network
  - `192.168.1.1`: Only the host with this specific IP address

  The following netmask will not work because it masks 28 bits, and 28 is not a multiple of 8:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
192.168.0.1/255.255.255.240
```

The server performs matching of host values in account names against the client host using the value returned by the system DNS resolver for the client host name or IP address. Except in the case that the account host value is specified using netmask notation, this comparison is performed as a string match, even for an account host value given as an IP address. This means that you should specify account host values in the same format used by DNS. Here are examples of problems to watch out for:

- Suppose that a host on the local network has a fully qualified name of `host1.example.com`. If DNS returns name lookups for this host as `host1.example.com`, use that name in account host values. But if DNS returns just `host1`, use `host1` instead.
- If DNS returns the IP address for a given host as `192.168.1.2`, that will match an account host value of `192.168.1.2` but not `192.168.01.2`. Similarly, it will match an account host pattern like `192.168.1.%` but not `192.168.01.%`.

To avoid problems like this, it is advisable to check the format in which your DNS returns host names and addresses, and use values in the same format in MySQL account names.

6.2.4 Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Credential checking is performed using the three `user` table scope columns (`Host`, `User`, and `Password`). The server accepts the connection only if the `Host` and `User` columns in some `user` table row match the client host name and user name and the client supplies the password specified in that row. The rules for permissible `Host` and `User` values are given in Section 6.2.3, “Specifying Account Names”.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL user name

If the `User` column value is nonblank, the user name in an incoming connection must match exactly. If the `User` value is blank, it matches any user name. If the `user` table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `Password` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password.

Nonblank `Password` values in the `user` table represent encrypted passwords. MySQL does not store passwords in cleartext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` function). The encrypted password then is used during the connection process when checking whether the password is correct. This is done without the encrypted password ever traveling over the connection. See Section 6.3.1, “User Names and Passwords”.

From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give nonadministrative users read access to tables in the `mysql` database*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following table shows how various combinations of `User` and `Host` values in the `user` table apply to incoming connections.

`User` Value	`Host` Value	Permissible Connections
`'fred'`	`'thomas.loc.gov'`	`fred`, connecting from `thomas.loc.gov`
`''`	`'thomas.loc.gov'`	Any user, connecting from `thomas.loc.gov`
`'fred'`	`'%'`	`fred`, connecting from any host
`''`	`'%'`	Any user, connecting from any host
`'fred'`	`'%.loc.gov'`	`fred`, connecting from any host in the `loc.gov` domain
`'fred'`	`'x.y.%'`	`fred`, connecting from `x.y.net`, `x.y.com`, `x.y.edu`, and so on; this is probably not useful
`'fred'`	`'192.168.10.177'`	`fred`, connecting from the host with IP address `192.168.10.177`
`'fred'`	`'192.168.10.%'`	`fred`, connecting from any host in the `192.168.10` class C subnet
`'fred'`	`'192.168.10.0/255.255.255.0'`	Same as previous example

It is possible for the client host name and user name of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `thomas.loc.gov` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

The server uses sorting rules that order rows with the most-specific `Host` values first. Literal host names and IP addresses are the most specific. (The specificity of a literal IP address is not affected by whether it has a netmask, so `192.168.1.13` and `192.168.1.0/255.255.255.0` are considered equally specific.) The pattern `'%'` means “any host” and is least specific. The empty string `''` also means “any host” but sorts after `'%'`. Rows with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means “any user” and is least specific). For rows with equally-specific `Host` and `User` values, the order is indeterminate.

To see how this works, suppose that the `user` table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads the table into memory, it sorts the rows using the rules just described. The result after sorting looks like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

A connection by `jeffrey` from `thomas.loc.gov` is matched by the first row, whereas a connection by `jeffrey` from any host is matched by the second.

Note

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is not true. The preceding example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. (See Section 12.13, “Information Functions”.) It returns a value in *`user_name@host_name`* format that indicates the `User` and `Host` values from the matching `user` table row. Suppose that `jeffrey` connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+---------------+
```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

Another way to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

6.2.5 Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue through that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `db`, `host`, `tables_priv`, `columns_priv`, or `procs_priv` tables. (You may find it helpful to refer to Section 6.2.2, “Grant Tables”, which lists the columns present in each of the grant tables.)

The `user` table grants privileges that are assigned to you on a global basis and that apply no matter what the default database is. For example, if the `user` table grants you the `DELETE` privilege, you can delete rows from any table in any database on the server host! It is wise to grant privileges in the `user` table only to people who need them, such as database administrators. For other users, you should leave all privileges in the `user` table set to `'N'` and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, columns, or routines.

The `db` and `host` tables grant database-specific privileges. Values in the scope columns of these tables can take the following forms:

- A blank `User` value in the `db` table matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.
- The wildcard characters “`%`” and “`_`” can be used in the `Host` and `Db` columns of either table. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character (“`_`”) as part of a database name, specify it as “`\_`” in the `GRANT` statement.
- A `'%'` `Host` value in the `db` table means “any host.” A blank `Host` value in the `db` table means “consult the `host` table for further information” (a process that is described later in this section).
- A `'%'` or blank `Host` value in the `host` table means “any host.”
- A `'%'` or blank `Db` value in either table means “any database.”

The server reads the `db` and `host` tables into memory and sorts them at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns, and sorts the `host` table based on the `Host` and `Db` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching rows, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters “`%`” and “`_`” can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A `'%'` or blank `Host` value means “any host.”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `Db`, `Table_name`, `Column_name`, and `Routine_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the `user` table row because that is the only table that specifies administrative privileges. The server grants access if the row permits the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row does not grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` or `host` tables. (They contain no `Shutdown_priv` column, so there is no need to do so.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global privileges by looking in the `user` table row. If the row permits the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's host name and MySQL user name. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.

2. If there is a matching `db` table row and its `Host` column is not blank, that row defines the user's database-specific privileges.

3. If the matching `db` table row's `Host` column is blank, it signifies that the `host` table enumerates which hosts should be permitted access to the database. In this case, a further lookup is done in the `host` table to find a match on the `Host` and `Db` columns. If no `host` table row matches, access is denied. If there is a match, the user's database-specific privileges are computed as the intersection (*not* the union!) of the privileges in the `db` and `host` table rows; that is, the privileges that are `'Y'` in both rows. (This way you can grant general privileges in the `db` table row and then selectively restrict them on a host-by-host basis using the `host` table rows.)

After determining the database-specific privileges granted by the `db` and `host` table rows, the server adds them to the global privileges granted by the `user` table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and permits or denies access based on the result. For stored-routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global `user` row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges. Your

This documentation is for an older version. If you're
This documentation is for an older version. If you're

privileges might be such that the `user` table row grants one privilege and the `db` table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the rows in both tables must be combined.

The `host` table is not affected by the `GRANT` or `REVOKE` statements, so it is unused in most MySQL installations. If you modify it directly, you can use it for some specialized purposes, such as to maintain a list of secure servers on the local network that are granted all privileges.

You can also use the `host` table to indicate hosts that are *not* secure. Suppose that you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can enable access to all hosts on your network except that machine by using `host` table rows like this:

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

6.2.6 When Privilege Changes Take Effect

When `mysqld` starts, it reads all grant table contents into memory. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using account-management statements such as `GRANT`, `REVOKE`, or `SET PASSWORD`, the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE`, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables. If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes do not seem to make any difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

When the server reloads the grant tables, privileges for each existing client connection are affected as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.

Note

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database or flushing the privileges.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only for subsequent connections.

If the server is started with the `--skip-grant-tables` option, it does not read the grant tables or implement any access control. Anyone can connect and do anything. To cause a server thus started to read the tables and enable access checking, flush the privileges.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6.2.7 Troubleshooting Problems Connecting to MySQL

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port number, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking`, it will not accept TCP/IP connections at all. If the server was started with `--bind-address=127.0.0.1`, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the MySQL data directory, including the `mysql` database containing the grant tables. For distributions that do not do this, you must initialize the data directory manually. For details, see Section 2.18, “Postinstallation Setup and Testing”.

  To determine whether you need to initialize the grant tables, look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If not, initialize the data directory. After doing so and starting the server, test the initial privileges by executing this command:

```
shell> mysql -u root
```

  The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

```
shell> mysql -u root mysql
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see Section 2.18.4, “Securing the Initial MySQL Accounts”.

- If you have updated an existing MySQL installation to a newer version, did you run the `mysql_upgrade` script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

  For information on how to deal with this, see Section 6.1.2.4, “Password Hashing in MySQL”, and Section B.5.2.4, “Client does not support authentication protocol”.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check any applicable option files and your environment. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

  You can suppress the use of option files by a client program by invoking it with the `--no-defaults` option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

  The option files that clients use are listed in Section 4.2.6, “Using Option Files”. Environment variables are listed in Section 2.21, “Environment Variables”.

- If you get the following error, it means that you are using an incorrect `root` password:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

  If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the `--no-defaults` option as described in the previous item.

  For information on changing passwords, see Section 6.3.5, “Assigning Account Passwords”.

  If you have lost or forgotten the `root` password, see Section B.5.3.2, “How to Reset the Root Password”.

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` function. If you do not use `PASSWORD()` for these statements, the password will not work. For example, the following statement assigns a password, but fails to encrypt it, so the user is not able to connect afterward:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` function is unnecessary when you specify a password using the `GRANT` or (beginning with MySQL 5.0.2) `CREATE USER` statements, or the `mysqladmin password` command. Each of those automatically uses `PASSWORD()` to encrypt the password. See Section 6.3.5, “Assigning Account Passwords”, and Section 13.7.1.1, “CREATE USER Syntax”.

- `localhost` is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly.

  You can use a `--host=127.0.0.1` option to name the server host explicitly. This will make a TCP/IP connection to the local `mysqld` server. You can also use TCP/IP by specifying a `--host` option that uses the actual host name of the local host. In this case, the host name must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the `user` table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.

- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

  ```
  SELECT * FROM user;
  ```

  The result should include a row with the `Host` and `User` columns matching your client's host name and your MySQL user name.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

  ```
  Host ... is not allowed to connect to this MySQL server
  ```

  You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

  If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with `'%'` as the `Host` column value in the `user` table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. Then change the `'%'` in the `user` table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

  On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

  If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of `'root'` and that `mysqld` cannot resolve the host name for your client:

```
Access denied for user ''@'unknown'
```

  These errors indicate a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS host cache. See Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”.

  Some permanent solutions are:

  - Determine what is wrong with your DNS server and fix it.
  - Specify IP addresses rather than host names in the MySQL grant tables.
  - Put an entry for the client machine name in `/etc/hosts` on Unix or `\windows\hosts` on Windows.
  - Start `mysqld` with the `--skip-name-resolve` option.
  - Start `mysqld` with the `--skip-host-cache` option.
  - On Unix, if you are running the server and the client on the same machine, connect to `localhost`. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file, unless there are connection parameters specified to ensure that the client makes a TCP/IP connection. For more information, see Section 4.2.2, “Connecting to the MySQL Server”.
  - On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name `.` (period). Connections to `.` use a named pipe rather than TCP/IP.

- If `mysql -u root` works but `mysql -h your_hostname -u root` results in `Access denied` (where *`your_hostname`* is the actual host name of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have a row with host `'pluto'` in the `user` table, but your DNS tells MySQL that your host name is `'pluto.example.com'`, the row does not work. Try adding a row to the `user` table that contains the IP address of your host as the `Host` column value. (Alternatively, you could add a row to the `user` table with a `Host` value that contains a wildcard; for example, `'pluto.%'`. However, use of `Host` values ending with “`%`” is *insecure* and is *not* recommended!)

- If `mysql -u user_name` works but `mysql -u user_name some_db` does not, you have not granted access to the given user for the database named *`some_db`*.

- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If you cannot figure out why you get `Access denied`, remove from the `user` table all rows that have `Host` values containing wildcards (rows that contain `'%'` or `'_'` characters). A very common error is to insert a new row with `Host='%'` and `User='some_user'`, thinking that this enables you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include a row with `Host='localhost'` and `User=''`. Because that row has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new row when connecting from `localhost`! The correct procedure is to insert a second row with `Host='localhost'` and `User='some_user'`, or to delete the row with `Host='localhost'` and `User=''`. After deleting the row, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables. See also Section 6.2.4, “Access Control, Stage 1: Connection Verification”.

- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE` statement, your row in the `user` table does not have the `FILE` privilege enabled.

- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` statement, you will not need to specify the new password until after you flush the privileges, because the server will not know you've changed the password yet!

- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 6.2.6, “When Privilege Changes Take Effect”.

- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -pyour_pass db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass` syntax to specify the password. If you use the `-p` or `--password` option with no password value, MySQL prompts you for the password.)

- For testing purposes, start the `mysqld` server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the `mysqlaccess` script to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.

- If you get the following error, you may have a problem with the `db` or `host` table:

  ```
  Access to database denied
  ```

  If the row selected from the `db` table has an empty value in the `Host` column, make sure that there are one or more corresponding rows in the `host` table specifying which hosts the `db` table row applies to. This problem occurs infrequently because the `host` table is rarely used.

- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See Section 21.3.3, “The DBUG Package”.

- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at Section 1.7, “How to Report

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Bugs or Problems”. In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

6.3 MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system
- How to set up new accounts and remove existing accounts
- How to change passwords
- Guidelines for using passwords securely
- How to use secure connections

See also Section 13.7.1, “Account Management Statements”, which describes the syntax and use for all user-management SQL statements.

6.3.1 User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` system database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. The account may also have a password. For information about account representation in the `user` table, see Section 6.2.2, “Grant Tables”.

There are several distinctions between the way user names and passwords are used by MySQL and your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. This means that anyone can attempt to connect to the server using any user name, so you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password is able to connect successfully to the server.
- MySQL user names can be up to 16 characters long. Operating system user names may be of a different maximum length. For example, Unix user names typically are limited to eight characters.

Warning

The limit on MySQL user name length is hard-coded in MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

You should never alter the structure of tables in the `mysql` database in any manner whatsoever except by means of the procedure that is described in Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”. Attempting to redefine MySQL's system tables in any other fashion results in undefined (and unsupported!) behavior. The server is free to ignore rows that become malformed as a result of such modifications.

- To authenticate client connections that use MySQL built-in authentication, the server uses MySQL passwords stored in the `user` table. These passwords are distinct from passwords for logging in to your

This documentation is for an older version. If you're
This documentation is for an older version. If you're

operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

- MySQL encrypts passwords stored in the `user` table using its own algorithm. This encryption is the same as that implemented by the `PASSWORD()` SQL function but differs from that used during the Unix login process. Unix password encryption is the same as that implemented by the `ENCRYPT()` SQL function. See the descriptions of the `PASSWORD()` and `ENCRYPT()` functions in Section 12.12, “Encryption and Compression Functions”.

  From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the `user` table, knowledge of the encrypted password value could be used to connect to the MySQL server.) Section 6.1.2.4, “Password Hashing in MySQL”, discusses password encryption further.

- If the user name and password contain only ASCII characters, it is possible to connect to the server regardless of character set settings. To connect when the user name or password contain non-ASCII characters, the client should call the `mysql_options()` C API function with the `MYSQL_SET_CHARSET_NAME` option and appropriate character set name as arguments. This causes authentication to take place using the specified character set. Otherwise, authentication will fail unless the server default character set is the same as the encoding in the authentication defaults.

  Standard MySQL client programs support a `--default-character-set` option that causes `mysql_options()` to be called as just described. For programs that use a connector that is not based on the C API, the connector may provide an equivalent to `mysql_options()` that can be used instead. Check the connector documentation.

  The preceding notes do not apply for `ucs2`, which is not permitted as a client character set.

The MySQL installation process populates the grant tables with an initial account or accounts. The names and access privileges for these accounts are described in Section 2.18.4, “Securing the Initial MySQL Accounts”, which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as `CREATE USER`, `DROP USER`, `GRANT`, and `REVOKE`. See Section 13.7.1, “Account Management Statements”.

To connect to a MySQL server with a command-line client, specify user name and password options as necessary for the account that you want to use:

```
shell> mysql --user=monty --password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -p db_name
```

If you omit the password value following the `--password` or `-p` option on the command line (as just shown), the client prompts for one. Alternatively, the password can be specified on the command line:

```
shell> mysql --user=monty --password=password db_name
shell> mysql -u monty -ppassword db_name
```

If you use the `-p` option, there must be *no space* between `-p` and the following password value.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line. See Section 4.2.6, “Using Option Files”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For additional information about specifying user names, passwords, and other connection parameters, see Section 4.2.2, “Connecting to the MySQL Server”.

6.3.2 Adding User Accounts

You can create MySQL accounts two ways:

- By using account-management statements intended for creating accounts and establishing their privileges, such as `CREATE USER` and `GRANT`. These statements cause the server to make appropriate modifications to the underlying grant tables.
- By manipulating the MySQL grant tables directly with statements such as `INSERT`, `UPDATE`, or `DELETE`.

The preferred method is to use account-management statements because they are more concise and less error-prone than manipulating the grant tables directly. All such statements are described in Section 13.7.1, “Account Management Statements”. Direct grant table manipulation is discouraged, and the server is free to ignore rows that become malformed as a result of such modifications.

Another option for creating accounts is to use the GUI tool MySQL Workbench. Also, several third-party programs offer capabilities for MySQL account administration. `phpMyAdmin` is one such program.

The following examples show how to use the `mysql` client program to set up new accounts. These examples assume that privileges have been set up according to the defaults described in Section 2.18.4, “Securing the Initial MySQL Accounts”. This means that to make changes, you must connect to the MySQL server as the MySQL `root` user, and the `root` account, which has the `CREATE USER` privilege, the `INSERT` privilege for the `mysql` database and the `RELOAD` administrative privilege.

As noted in the examples where appropriate, some of the statements will fail if the server's SQL mode has been set to enable certain restrictions. In particular, strict mode (`STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`) and `NO_AUTO_CREATE_USER` will prevent the server from accepting some of the statements. Workarounds are indicated for these cases. For more information about SQL modes and their effect on grant table manipulation, see Section 5.1.7, “Server SQL Modes”, and Section 13.7.1.3, “GRANT Syntax”.

First, use the `mysql` program to connect to the server as the MySQL `root` user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the `root` account, you must also supply a `--password` or `-p` option, both for this `mysql` command and for those later in this section.

After connecting to the server as `root`, you can add new accounts. The following example uses `CREATE USER` and `GRANT` statements to set up four accounts:

```
mysql> CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
    ->     WITH GRANT OPTION;
mysql> CREATE USER 'monty'@'%' IDENTIFIED BY 'some_pass';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
    ->     WITH GRANT OPTION;
mysql> CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin_pass';
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> CREATE USER 'dummy'@'localhost';
```

The accounts created by those statements have the following properties:

- Two accounts have a user name of `monty` and a password of `some_pass`. Both are superuser accounts with full privileges to do anything. The `'monty'@'localhost'` account can be used only when

This documentation is for an older version. If you're
This documentation is for an older version. If you're

connecting from the local host. The `'monty'@'%'` account uses the `'%'` wildcard for the host part, so it can be used to connect from any host.

The `'monty'@'localhost'` account is necessary if there is an anonymous-user account for `localhost`. Without the `'monty'@'localhost'` account, that anonymous-user account takes precedence when `monty` connects from the local host and `monty` is treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific `Host` column value than the `'monty'@'%'` account and thus comes earlier in the `user` table sort order. (`user` table sorting is discussed in Section 6.2.4, “Access Control, Stage 1: Connection Verification”.)

- The `'admin'@'localhost'` account has a password of `admin_pass`. This account can be used only by `admin` to connect from the local host. It is granted the `RELOAD` and `PROCESS` administrative privileges. These privileges enable the `admin` user to execute the `mysqladmin reload`, `mysqladmin refresh`, and `mysqladmin flush-xxx` commands, as well as `mysqladmin processlist`. No privileges are granted for accessing any databases. You could add such privileges using `GRANT` statements.

- The `'dummy'@'localhost'` account has no password (which is insecure and not recommended). This account can be used only to connect from the local host. No privileges are granted. It is assumed that you will grant specific privileges to the account using `GRANT` statements.

The statements that create accounts with no password will fail if the `NO_AUTO_CREATE_USER` SQL mode is enabled. To deal with this, use an `IDENTIFIED BY` clause that specifies a nonempty password.

To see the privileges for an account, use `SHOW GRANTS`:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------------------+
| Grants for admin@localhost                          |
+-----------------------------------------------------+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

As an alternative to `CREATE USER` and `GRANT`, you can create the same accounts directly by issuing `INSERT` statements and then telling the server to reload the grant tables using `FLUSH PRIVILEGES`:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
    ->     VALUES('localhost','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
    ->     VALUES('%','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y',
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y',
    ->     '','','','',0,0,0,0);
mysql> INSERT INTO user SET Host='localhost',User='admin',
    ->     Password=PASSWORD('admin_pass'),
    ->     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

When you create accounts with `INSERT`, it is necessary to use `FLUSH PRIVILEGES` to tell the server to reload the grant tables. Otherwise, the changes go unnoticed until you restart the server. With `CREATE USER`, `FLUSH PRIVILEGES` is unnecessary.

The reason for using the `PASSWORD()` function with `INSERT` is to encrypt the password. The `CREATE USER` statement encrypts the password for you, so `PASSWORD()` is unnecessary.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The 'Y' values enable privileges for the accounts. Depending on your MySQL version, you may have to use a different number of 'Y' values in the first two INSERT statements. The INSERT statement for the admin account employs the more readable extended INSERT syntax using SET.

In the INSERT statement for the dummy account, only the Host, User, and Password columns in the user table row are assigned values. None of the privilege columns are set explicitly, so MySQL assigns them all the default value of 'N'. This is equivalent to what CREATE USER does.

If strict SQL mode is enabled, all columns that have no default value must have a value specified. In this case, INSERT statements must explicitly specify values for the ssl_cipher, x509_issuer, and x509_subject columns.

To set up a superuser account, it is necessary only to insert a user table row with all privilege columns set to 'Y'. The user table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and grant them access to specific databases. Each of them has a user name of custom and password of obscure.

To create the accounts with CREATE USER and GRANT, use the following statements:

```
shell> mysql --user=root mysql
mysql> CREATE USER 'custom'@'localhost' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON bankaccount.*
    ->     TO 'custom'@'localhost';
mysql> CREATE USER 'custom'@'host47.example.com' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON expenses.*
    ->     TO 'custom'@'host47.example.com';
mysql> CREATE USER 'custom'@'%.example.com' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON customer.*
    ->     TO 'custom'@'%.example.com';
```

The three accounts can be used as follows:

- The first account can access the bankaccount database, but only from the local host.
- The second account can access the expenses database, but only from the host host47.example.com.
- The third account can access the customer database, from any host in the example.com domain. This account has access from all machines in the domain due to use of the "%" wildcard character in the host part of the account name.

To set up the custom accounts without GRANT, use INSERT statements as follows to modify the grant tables directly:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('host47.example.com','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('%.example.com','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('localhost','bankaccount','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('host47.example.com','expenses','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('%.example.com','customer','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three `INSERT` statements add `user` table entries that permit the user `custom` to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of `'N'`). The next three `INSERT` statements add `db` table entries that grant privileges to `custom` for the `bankaccount`, `expenses`, and `customer` databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you must tell the server to reload them with `FLUSH PRIVILEGES` so that the privilege changes take effect.

6.3.3 Removing User Accounts

To remove an account, use the `DROP USER` statement, which is described in Section 13.7.1.2, “DROP USER Syntax”. For example:

```
mysql> DROP USER 'jeffrey'@'localhost';
```

6.3.4 Setting Account Resource Limits

One means of restricting client use of MySQL server resources is to set the global `max_user_connections` system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, setting `max_user_connections` does not enable management of individual accounts. Both types of control are of interest to MySQL administrators.

To address such concerns, MySQL permits limits for individual accounts on use of these server resources:

- The number of queries an account can issue per hour
- The number of updates an account can issue per hour
- The number of times an account can connect to the server per hour
- The number of simultaneous connections to the server by an account

Any statement that a client can issue counts against the query limit, unless its results are served from the query cache. Only statements that modify databases or tables count against the update limit.

An “account” in this context corresponds to a row in the `mysql.user` table. That is, a connection is assessed against the `User` and `Host` values in the `user` table row that applies to the connection. For example, an account `'usera'@'%.example.com'` corresponds to a row in the `user` table that has `User` and `Host` values of `usera` and `%.example.com`, to permit `usera` to connect from any host in the `example.com` domain. In this case, the server applies resource limits in this row collectively to all connections by `usera` from any host in the `example.com` domain because all such connections use the same account.

Before MySQL 5.0.3, an “account” was assessed against the actual host from which a user connects. This older method of accounting may be selected by starting the server with the `--old-style-user-limits` option. In this case, if `usera` connects simultaneously from `host1.example.com` and `host2.example.com`, the server applies the account resource limits separately to each connection.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If `usera` connects again from `host1.example.com`, the server applies the limits for that connection together with the existing connection from that host.

To establish resource limits for an account, use the `GRANT` statement (see Section 13.7.1.3, "GRANT Syntax"). Provide a `WITH` clause that names each resource to be limited. The default value for each limit is zero (no limit). For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue these statements:

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank';
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. For `MAX_USER_CONNECTIONS`, the limit is an integer representing the maximum number of simultaneous connections by the account. If this limit is set to zero, the global `max_user_connections` system variable value determines the number of simultaneous connections. If `max_user_connections` is also zero, there is no limit for the account.

To modify limits for an existing account, use a `GRANT USAGE` statement at the global level (`ON *.*`). The following statement changes the query limit for `francis` to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

As mentioned previously, the simultaneous-connection limit for an account is determined from the `MAX_USER_CONNECTIONS` limit and the `max_user_connections` system variable. Suppose that the global `max_user_connections` value is 10 and three accounts have individual resource limits specified as follows:

```
GRANT ... TO 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;
GRANT ... TO 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;
GRANT ... TO 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

`user1` has a connection limit of 10 (the global `max_user_connections` value) because it has a `MAX_USER_CONNECTIONS` limit of zero. `user2` and `user3` have connection limits of 5 and 20, respectively, because they have nonzero `MAX_USER_CONNECTIONS` limits.

The server stores resource limits for an account in the `user` table row corresponding to the account. The `max_questions`, `max_updates`, and `max_connections` columns store the per-hour limits, and the `max_user_connections` column stores the `MAX_USER_CONNECTIONS` limit. (See Section 6.2.2, "Grant Tables".) If your `user` table does not have these columns, it must be upgraded; see Section 4.4.9, "`mysql_upgrade` — Check Tables for MySQL Upgrade".

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, the server rejects further connections for the account until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, the server rejects further queries or updates until the hour is up. In all such cases, the server issues appropriate error messages.

Resource counting occurs per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be reset to zero by setting any of its limits again. Specify a limit value equal to the value currently assigned to the account.

Per-hour counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts. Counts do not carry over through server restarts.

For the `MAX_USER_CONNECTIONS` limit, an edge case can occur if the account currently has open the maximum number of connections permitted to it: A disconnect followed quickly by a connect can result in an error (`ER_TOO_MANY_USER_CONNECTIONS` or `ER_USER_LIMIT_REACHED`) if the server has not fully processed the disconnect by the time the connect occurs. When the server finishes disconnect processing, another connection will once more be permitted.

6.3.5 Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts.

MySQL stores passwords in the `user` table in the `mysql` system database. Operations that assign or modify passwords are permitted only to users with the `CREATE USER` privilege, or, alternatively, privileges for the `mysql` database (`INSERT` privilege to create new accounts, `UPDATE` privilege to modify existing accounts).

The discussion here summarizes syntax only for the most common password-assignment statements. For complete details on other possibilities, see Section 13.7.1.1, “CREATE USER Syntax”, Section 13.7.1.3, “GRANT Syntax”, and Section 13.7.1.6, “SET PASSWORD Syntax”.

MySQL hashes passwords stored in the `mysql.user` table to obfuscate them. For most statements described here, MySQL automatically hashes the password specified. An exception is `SET PASSWORD ... = PASSWORD('auth_string')`, for which you use the `PASSWORD()` function explicitly to hash the password. There are also syntaxes for `CREATE USER`, `GRANT`, and `SET PASSWORD` that permit hashed values to be specified literally; for details, see the descriptions of those statements.

To assign a password when you create a new account, use `CREATE USER` and include an `IDENTIFIED BY` clause:

```
mysql> CREATE USER 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

For this `CREATE USER` syntax, MySQL automatically hashes the password before storing it in the `mysql.user` table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To assign or change a password for an existing account, use one of the following methods:

- Use `SET PASSWORD` with the `PASSWORD()` function:

```
mysql> SET PASSWORD FOR
    -> 'jeffrey'@'localhost' = PASSWORD('mypass');
```

If you are not connected as an anonymous user, you can change your own password by omitting the `FOR` clause:

```
mysql> SET PASSWORD = PASSWORD('mypass');
```

The `PASSWORD()` function hashes the password using the hashing method determined by the value of the `old_passwords` system variable value. If `SET PASSWORD` rejects the hashed password value returned by `PASSWORD()` as not being in the correct format, it may be necessary to change `old_passwords` to change the hashing method. For descriptions of the permitted values, see Section 5.1.4, “Server System Variables”.

- You can also use a `GRANT USAGE` statement at the global level (`ON *.*`) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

For this `GRANT` syntax, MySQL automatically hashes the password before storing it in the `mysql.user` table.

- To change an account password from the command line, use the `mysqladmin` command:

```
shell> mysqladmin -u user_name -h host_name password "new_password"
```

The account for which this command sets the password is the one with a `mysql.user` table row that matches *`user_name`* in the `User` column and the client host *from which you connect* in the `Host` column.

For password changes made using `mysqladmin`, MySQL automatically hashes the password before storing it in the `mysql.user` table.

6.3.6 Using Secure Connections

With an unencrypted connection between the MySQL client and the server, someone with access to the network could watch all your traffic and inspect the data being sent or received between client and server.

When you must move information over a network in a secure fashion, an unencrypted connection is unacceptable. To make any kind of data unreadable, use encryption. Encryption algorithms must include security elements to resist many kinds of known attacks such as changing the order of encrypted messages or replaying data twice.

MySQL supports secure (encrypted) connections between clients and the server using the TLS (Transport Layer Security) protocol. TLS is sometimes referred to as SSL (Secure Sockets Layer).

TLS uses encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect data change, loss, or replay. TLS also incorporates algorithms that provide identity verification using the X509 standard.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

X509 makes it possible to identify someone on the Internet. In basic terms, there should be some entity called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can present the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted using this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

For more information about TLS, SSL, X509, encryption, or public-key cryptography, perform an Internet search for the keywords in which you are interested.

MySQL can be compiled for secure-connection support using OpenSSL or yaSSL. For a comparison of the two packages, see Section 6.3.6.1, “OpenSSL Versus yaSSL” For information about the encryption protocols and ciphers each package supports, see Section 6.3.6.3, “Secure Connection Protocols and Ciphers”.

MySQL performs encryption on a per-connection basis, and use of encryption can be optional or mandatory. This enables you to choose an encrypted or unencrypted connection according to the requirements of individual applications. For information on how to require users to use encrypted connections, see the discussion of the `REQUIRE` clause of the `GRANT` statement in Section 13.7.1.3, “GRANT Syntax”.

Encrypted connections are not used by default. For applications that require the security provided by encrypted connections, the extra computation to encrypt the data is worthwhile.

Secure connections are available through the MySQL C API using the `mysql_ssl_set()` and `mysql_options()` functions. See Section 20.6.7.67, “mysql_ssl_set()”, and Section 20.6.7.49, “mysql_options()”.

Replication uses the C API, so secure connections can be used between master and slave servers. See Section 16.3.7, “Setting Up Replication to Use Secure Connections”.

It is also possible to connect securely from within an SSH connection to the MySQL server host. For an example, see Section 6.3.8, “Connecting to MySQL Remotely from Windows with SSH”.

6.3.6.1 OpenSSL Versus yaSSL

MySQL can be compiled using OpenSSL or yaSSL, both of which enable secure conections based on the OpenSSL API.

OpenSSL and yaSSL offer the same basic functionality, but additional features are available in MySQL distributions compiled using OpenSSL: OpenSSL supports a wider range of encryption ciphers from which to choose for the `--ssl-cipher` option, and supports the `--ssl-capath` option. See Section 6.3.6.5, “Command Options for Secure Connections”.

6.3.6.2 Building MySQL with Support for Secure Connections

To use SSL connections between the MySQL server and client programs, your system must support either OpenSSL or yaSSL, and your version of MySQL must be built with SSL support. To make it easier to use secure connections, as of version 5.0.10, MySQL is bundled with yaSSL, which uses the same licensing model as MySQL. (OpenSSL uses an Apache-style license.) yaSSL support is available on all platforms supported by Oracle Corporation.

To get secure connections to work with MySQL and SSL, you must do the following:

1. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, and you are going to use OpenSSL rather than the bundled yaSSL library, install OpenSSL if it has not already been installed. We have tested MySQL with OpenSSL 0.9.6. To obtain OpenSSL, visit http://www.openssl.org.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Building MySQL using OpenSSL requires a shared OpenSSL library, otherwise linker errors occur. Alternatively, build MySQL using yaSSL.

2. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, configure a MySQL source distribution to use SSL. When you configure MySQL, invoke the `configure` script with the appropriate option to select the SSL library that you want to use.

   For yaSSL:

   ```
   shell> ./configure --with-yassl
   ```

   For OpenSSL:

   ```
   shell> ./configure --with-openssl
   ```

   Before MySQL 5.0, it was also neccessary to use `--with-vio`, but that option is no longer required.

   Then compile and install the distribution.

   On Unix platforms, yaSSL retrieves true random numbers from either `/dev/urandom` or `/dev/random`. Bug#13164 lists workarounds for some very old platforms which do not support these devices.

3. To check whether a server binary is compiled with SSL support, invoke it with the `--ssl` option. An error will occur if the server does not support SSL:

   ```
   shell> mysqld --ssl --help
   060525 14:18:52 [ERROR] mysqld: unknown option '--ssl'
   ```

   To check whether a running `mysqld` server supports secure connections, examine the value of the `have_ssl` system variable (if you have no `have_ssl` variable, check for `have_openssl`):

   ```
   mysql> SHOW VARIABLES LIKE 'have_ssl';
   +---------------+-------+
   | Variable_name | Value |
   +---------------+-------+
   | have_ssl      | YES   |
   +---------------+-------+
   ```

   If the value is `YES`, the server supports secure connections. If the value is `DISABLED`, the server is capable of supporting secure connections but was not started with the appropriate `--ssl-xxx` options to enable secure connections to be used; see Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”.

6.3.6.3 Secure Connection Protocols and Ciphers

To determine which encryption protocol and cipher are in use for an encrypted connection, use the following statements to check the values of the `Ssl_version` and `Ssl_cipher` status variables:

```
mysql> SHOW SESSION STATUS LIKE 'Ssl_version';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Ssl_version   | TLSv1 |
+---------------+-------+
mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

If the connection is not encrypted, both variables have an empty value.

MySQL supports encrypted connections using the TLSv1 protocol.

To determine which ciphers a given server supports, use the following statement to check the value of the `Ssl_cipher_list` status variable:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

The set of available ciphers depends on your MySQL version and whether MySQL was compiled using OpenSSL or yaSSL, and (for OpenSSL) the library version used to compile MySQL.

For example, for OpenSSL, the list may include these ciphers:

```
AES256-GCM-SHA384
AES256-SHA
AES256-SHA256
CAMELLIA256-SHA
DES-CBC3-SHA
DHE-DSS-AES256-GCM-SHA384
DHE-DSS-AES256-SHA
DHE-DSS-AES256-SHA256
DHE-DSS-CAMELLIA256-SHA
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
DHE-RSA-CAMELLIA256-SHA
ECDH-ECDSA-AES256-GCM-SHA384
ECDH-ECDSA-AES256-SHA
ECDH-ECDSA-AES256-SHA384
ECDH-ECDSA-DES-CBC3-SHA
ECDH-RSA-AES256-GCM-SHA384
ECDH-RSA-AES256-SHA
ECDH-RSA-AES256-SHA384
ECDH-RSA-DES-CBC3-SHA
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
ECDHE-ECDSA-DES-CBC3-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA
ECDHE-RSA-AES256-SHA384
ECDHE-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
PSK-3DES-EDE-CBC-SHA
PSK-AES256-CBC-SHA
SRP-DSS-3DES-EDE-CBC-SHA
SRP-DSS-AES-128-CBC-SHA
SRP-DSS-AES-256-CBC-SHA
SRP-RSA-3DES-EDE-CBC-SHA
SRP-RSA-AES-128-CBC-S
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SRP-RSA-AES-256-CBC-SHA
```

yaSSL supports these ciphers:

```
AES128-RMD
AES128-SHA
AES256-RMD
AES256-SHA
DES-CBC-SHA
DES-CBC3-RMD
DES-CBC3-SHA
DHE-RSA-AES128-RMD
DHE-RSA-AES128-SHA
DHE-RSA-AES256-RMD
DHE-RSA-AES256-SHA
DHE-RSA-DES-CBC3-RMD
EDH-RSA-DES-CBC-SHA
EDH-RSA-DES-CBC3-SHA
RC4-MD5
RC4-SHA
```

6.3.6.4 Configuring MySQL to Use Secure Connections

To enable secure connections, your MySQL distribution must be built with SSL support, as described in Section 6.3.6.2, “Building MySQL with Support for Secure Connections”. In addition, the proper options must be used to specify the appropriate certificate and key files. For a complete list of options related to establishment of secure connections, see Section 6.3.6.5, “Command Options for Secure Connections”.

If you need to create the required SSL files, see Section 6.3.7, “Creating SSL Certificates and Keys Using openssl”.

Server-Side Configuration for Secure Connections

To start the MySQL server so that it permits clients to connect securely, use options that identify the certificate and key files the server uses when establishing a secure connection:

- `--ssl-ca` identifies the Certificate Authority (CA) certificate.
- `--ssl-cert` identifies the server public key certificate. This can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key` identifies the server private key.

For example, start the server with these lines in the `my.cnf` file, changing the file names as necessary:

```
[mysqld]
ssl-ca=ca.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

Each option names a file in PEM format. If you have a MySQL source distribution, you can test your setup using the demonstration certificate and key files in its `mysql-test/std_data` directory.

Client-Side Configuration for Secure Connections

For client programs, options for secure connections are similar to those used on the server side, but `--ssl-cert` and `--ssl-key` identify the client public and private key:

- `--ssl-ca` identifies the Certificate Authority (CA) certificate. This option, if used, must specify the same certificate used by the server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--ssl-cert` identifies the client public key certificate.
- `--ssl-key` identifies the client private key.

To connect securely to a MySQL server that supports secure connections, the options that a client must specify depend on the encryption requirements of the MySQL account used by the client. (See the discussion of the `REQUIRE` clause in Section 13.7.1.3, “GRANT Syntax”.)

Suppose that you want to connect using an account that has no special encryption requirements or was created using a `GRANT` statement that includes the `REQUIRE SSL` option. As a recommended set of secure-connection options, start the server with at least `--ssl-cert` and `--ssl-key`, and invoke the client with `--ssl-ca`. A client can connect securely like this:

```
shell> mysql --ssl-ca=ca.pem
```

To require that a client certificate also be specified, create the account using the `REQUIRE X509` option. Then the client must also specify the proper client key and certificate files or the server will reject the connection:

```
shell> mysql --ssl-ca=ca.pem \
       --ssl-cert=client-cert.pem \
       --ssl-key=client-key.pem
```

To prevent use of encryption and override other `--ssl-xxx` options, invoke the client program with `--ssl=0` or a synonym (`--skip-ssl`, `--disable-ssl`):

```
shell> mysql --ssl=0
```

A client can determine whether the current connection with the server uses encryption by checking the value of the `Ssl_cipher` status variable. If the value is empty, the connection is encrypted. Otherwise, the connection is encrypted and the value indicates the encryption cipher. For example:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

For the `mysql` client, an alternative is to use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s
...
SSL: Cipher in use is DHE-RSA-AES256-SHA
...
```

Or:

```
mysql> \s
...
SSL: Not in use
...
```

C API Configuration for Secure Connections

The C API enables application programs to use secure connections:

This documentation is for an older version. If you're

- To establish a secure connection, use the `mysql_ssl_set()` C API function to set the appropriate certificate options before calling `mysql_real_connect()`. See Section 20.6.7.67, “mysql_ssl_set()”.
- To determine whether encryption is in use after the connection is established, use `mysql_get_ssl_cipher()`. A non-`NULL` return value indicates an encrypted connection and names the cipher used for encryption. A `NULL` return value indicates that encryption is not being used. See Section 20.6.7.33, “mysql_get_ssl_cipher()”.

Replication uses the C API, so secure connections can be used between master and slave servers. See Section 16.3.7, “Setting Up Replication to Use Secure Connections”.

6.3.6.5 Command Options for Secure Connections

This section describes options that specify whether to use secure connections and the names of certificate and key files. These options can be given on the command line or in an option file. They are not available unless MySQL has been built with SSL support. See Section 6.3.6.2, “Building MySQL with Support for Secure Connections”. For examples of suggested use and how to check whether a connection is secure, see Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”. (There are also `--master-ssl*` options that can be used for setting up a secure connection from a slave replication server to a master server; see Section 16.1.2, “Replication and Binary Logging Options and Variables”.)

Table 6.8 Secure-Connection Option Summary

Format	Description	Introduced
--skip-ssl	Do not use secure connection	
--ssl	Enable secure connection	
--ssl-ca	Path of file that contains list of trusted SSL CAs	5.0.23
--ssl-capath	Path of directory that contains trusted SSL CA certificates in PEM format	5.0.23
--ssl-cert	Path of file that contains X509 certificate in PEM format	5.0.23
--ssl-cipher	List of permitted ciphers to use for connection encryption	5.0.23
--ssl-key	Path of file that contains X509 key in PEM format	5.0.23
--ssl-verify-server-cert	Verify server certificate Common Name value against host name used when connecting to server	5.0.23

- `--ssl`

  For the MySQL server, this option specifies that the server permits but does not require secure connections.

  For MySQL client programs, this option permits but does not require the client to connect to the server using encryption. Therefore, this option is not sufficient in itself to cause a secure connection to be used. For example, if you specify this option for a client program but the server has not been configured to support secure connections, the client falls back to an unencrypted connection.

  As a recommended set of options to enable secure connections, use at least `--ssl-cert` and `--ssl-key` on the server side and `--ssl-ca` on the client side. See Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”.

  `--ssl` may be implied by other `--ssl-xxx` options, as indicated in the descriptions for those options.

  The `--ssl` option in negated form overrides other `--ssl-xxx` options and indicates that encryption should *not* be used. To do this, specify the option as `--ssl=0` or a synonym (`--skip-ssl`, `--disable-ssl`). For example, you might have options specified in the `[client]` group of your

This documentation is for an older version. If you're
This documentation is for an older version. If you're

option file to use secure connections by default when you invoke MySQL client programs. To use an unencrypted connection instead, invoke the client program with `--ssl=0` on the command line to override the options in the option file.

To require use of secure connections for a MySQL account, use a `GRANT` statement for the account that includes at least a `REQUIRE SSL` clause. Connections for the account will be rejected unless MySQL supports secure connections and the server and client have been started with the proper secure-connection options.

The `REQUIRE` clause permits other encryption-related options, which can be used to enforce stricter requirements than `REQUIRE SSL`. For additional details about which command options may or must be specified by clients that connect using accounts configured using the various `REQUIRE` options, see the description of `REQUIRE` in Section 13.7.1.3, “GRANT Syntax”.

- `--ssl-ca=file_name`

  The path to a file in PEM format that contains a list of trusted SSL certificate authorities. This option implies `--ssl`.

  As of MySQL 5.0.40, if you use encryption when establishing a client connection, to tell the client not to authenticate the server certificate, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established for the client account, and it still uses any `--ssl-ca` or `--ssl-capath` option values specified at server startup.

- `--ssl-capath=dir_name`

  The path to a directory that contains trusted SSL certificate authority certificates in PEM format. This option implies `--ssl`.

  As of MySQL 5.0.40, if you use encryption when establishing a client connection, to tell the client not to authenticate the server certificate, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established for the client account, and it still uses any `--ssl-ca` or `--ssl-capath` option values specified at server startup.

  MySQL distributions compiled using OpenSSL support the `--ssl-capath` option (see Section 6.3.6.1, “OpenSSL Versus yaSSL”). Distributions compiled using yaSSL do not because yaSSL does not look in any directory and does not follow a chained certificate tree. yaSSL requires that all components of the CA certificate tree be contained within a single CA certificate tree and that each certificate in the file has a unique SubjectName value. To work around this yaSSL limitation, concatenate the individual certificate files comprising the certificate tree into a new file and specify that file as the value of the `--ssl-ca` option.

- `--ssl-cert=file_name`

  The name of the SSL certificate file in PEM format to use for establishing a secure connection. This option implies `--ssl`.

- `--ssl-cipher=cipher_list`

  A list of permissible ciphers to use for connection encryption. If no cipher in the list is supported, encrypted connections will not work. This option implies `--ssl`.

  For greatest portability, *`cipher_list`* should be a list of one or more cipher names, separated by colons. This format is understood both by OpenSSL and yaSSL. Examples:

```
--ssl-cipher=AES128-SHA
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
--ssl-cipher=DHE-RSA-AES256-SHA:AES128-SHA
```

OpenSSL supports a more flexible syntax for specifying ciphers, as described in the OpenSSL documentation at http://www.openssl.org/docs/apps/ciphers.html. However, yaSSL does not, so attempts to use that extended syntax fail for a MySQL distribution compiled using yaSSL.

For information about which encryption ciphers MySQL supports, see Section 6.3.6.3, “Secure Connection Protocols and Ciphers”.

- `--ssl-key=file_name`

  The name of the SSL key file in PEM format to use for establishing a secure connection. This option implies `--ssl`.

  If the MySQL distribution was compiled using OpenSSL and the key file is protected by a passphrase, the program prompts the user for the passphrase. The password must be given interactively; it cannot be stored in a file. If the passphrase is incorrect, the program continues as if it could not read the key. If the MySQL distribution was built using yaSSL and the key file is protected by a passphrase, an error occurs.

- `--ssl-verify-server-cert`

  This option is available only for client programs, not the server. It causes the client to check the server's Common Name value in the certificate that the server sends to the client. The client verifies that name against the host name the client uses for connecting to the server, and the connection fails if there is a mismatch. For encrypted connections, this option helps prevent man-in-the-middle attacks. Verification is disabled by default. This option was added in MySQL 5.0.23.

6.3.7 Creating SSL Certificates and Keys Using openssl

This section describes how to use the `openssl` command to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the `openssl` command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

Important

Whatever method you use to generate the certificate and key files, the Common Name value used for the server and client certificates/keys must each differ from the Common Name value used for the CA certificate. Otherwise, the certificate and key files will not work for servers compiled using OpenSSL. A typical error in this case is:

```
ERROR 2026 (HY000): SSL connection error:
error:00000001:lib(0):func(0):reason(1)
```

Example 1: Creating SSL Files from the Command Line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the `openssl` commands. To generate test files, you can press Enter to all prompts. To generate files for production use, you should provide nonempty responses.

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts
```


This documentation is for an older version. If you're

```
# Create CA certificate
shell> openssl genrsa 2048 > ca-key.pem
shell> openssl req -new -x509 -nodes -days 3600 \
         -key ca-key.pem -out ca.pem

# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
shell> openssl req -newkey rsa:2048 -days 3600 \
         -nodes -keyout server-key.pem -out server-req.pem
shell> openssl rsa -in server-key.pem -out server-key.pem
shell> openssl x509 -req -in server-req.pem -days 3600 \
         -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem

# Create client certificate, remove passphrase, and sign it
# client-cert.pem = public key, client-key.pem = private key
shell> openssl req -newkey rsa:2048 -days 3600 \
         -nodes -keyout client-key.pem -out client-req.pem
shell> openssl rsa -in client-key.pem -out client-key.pem
shell> openssl x509 -req -in client-req.pem -days 3600 \
         -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

After generating the certificates, verify them:

```
shell> openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
server-cert.pem: OK
client-cert.pem: OK
```

Now you have a set of files that can be used as follows:

- `ca.pem`: Use this as the argument to `--ssl-ca` on the server and client sides. (The CA certificate, if used, must be the same on both sides.)
- `server-cert.pem`, `server-key.pem`: Use these as the arguments to `--ssl-cert` and `--ssl-key` on the server side.
- `client-cert.pem`, `client-key.pem`: Use these as the arguments to `--ssl-cert` and `--ssl-key` on the client side.

To use the files for SSL connections, see Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”.

Example 2: Creating SSL Files Using a Script on Unix

Here is an example script that shows how to set up SSL certificate and key files for MySQL. After executing the script, use the files for SSL connections as described in Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”.

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca.pem \
    -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# ..........++++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -cert $DIR/ca.pem -policy policy_anything \
    -out $DIR/server-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ....................................++++++
# ..........................................++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca -cert $DIR/ca.pem -policy policy_anything \
    -out $DIR/client-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/server-cert.pem
ssl-key=$DIR/server-key.pem
EOF
```

Example 3: Creating SSL Files on Windows

Download OpenSSL for Windows if it is not installed on your system. An overview of available packages can be seen here:

```
http://www.slproweb.com/products/Win32OpenSSL.html
```

Choose the Win32 OpenSSL Light or Win64 OpenSSL Light package, depending on your architecture (32-bit or 64-bit). The default installation location will be `C:\OpenSSL-Win32` or `C:\OpenSSL-Win64`, depending on which package you downloaded. The following instructions assume a default location of `C:\OpenSSL-Win32`. Modify this as necessary if you are using the 64-bit package.

If a message occurs during setup indicating `'...critical component is missing: Microsoft Visual C++ 2008 Redistributables'`, cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

- Visual C++ 2008 Redistributables (x86), available at:

  ```
  http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Visual C++ 2008 Redistributables (x64), available at:

```
http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6
```

After installing the additional package, restart the OpenSSL setup procedure.

During installation, leave the default `C:\OpenSSL-Win32` as the install path, and also leave the default option `'Copy OpenSSL DLL files to the Windows system directory'` selected.

When the installation has finished, add `C:\OpenSSL-Win32\bin` to the Windows System Path variable of your server:

1. On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
2. Select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
3. Under **System Variables**, select **Path**, then click the **Edit** button. The **Edit System Variable** dialogue should appear.
4. Add `';C:\OpenSSL-Win32\bin'` to the end (notice the semicolon).
5. Press OK 3 times.
6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (`Start>Run>cmd.exe`) and verifying that OpenSSL is available:

```
Microsoft Windows [Version ...]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd \

C:\>openssl
OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.

C:\>
```

Depending on your version of Windows, the preceding path-setting instructions might differ slightly.

After OpenSSL has been installed, use instructions similar to those from from Example 1 (shown earlier in this section), with the following changes:

- Change the following Unix commands:

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts
```

  On Windows, use these commands instead:

```
# Create clean environment
C:\> md c:\newcerts
C:\> cd c:\newcerts
```

- When a `'\'` character is shown at the end of a command line, this `'\'` character must be removed and the command lines entered all on a single line.

After generating the certificate and key files, to use them for SSL connections, see Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6.3.8 Connecting to MySQL Remotely from Windows with SSH

This section describes how to get a secure connection to a remote MySQL server with SSH. The information was provided by David Carlson <dcarlson@mplcomm.com>.

1. Install an SSH client on your Windows machine. As a user, the best nonfree one I have found is from SecureCRT from http://www.vandyke.com/. Another option is f-secure from http://www.f-secure.com/. You can also find some free ones on Google at http://directory.google.com/Top/Computers/Internet/Protocols/SSH/Clients/Windows/.

2. Start your Windows SSH client. Set Host_Name = *yourmysqlserver_URL_or_IP*. Set userid=*your_userid* to log in to your server. This userid value might not be the same as the user name of your MySQL account.

3. Set up port forwarding. Either do a remote forward (Set local_port: 3306, remote_host: *yourmysqlservername_or_ip*, remote_port: 3306) or a local forward (Set port: 3306, host: localhost, remote port: 3306).

4. Save everything, otherwise you will have to redo it the next time.

5. Log in to your server with the SSH session you just created.

6. On your Windows machine, start some ODBC application (such as Access).

7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in localhost for the MySQL host server, not *yourmysqlservername*.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

6.3.9 SQL-Based MySQL Account Activity Auditing

Applications can use the following guidelines to perform SQL-based auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the mysql.user table. When a client connects successfully, the server authenticates the client to a particular row in this table. The User and Host column values in this row uniquely identify the account and correspond to the *'user_name'@'host_name'* format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the CURRENT_USER() function can be invoked to determine which account this is for the client user. Its value is constructed from the User and Host columns of the user table row for the account.

However, there are circumstances under which the CURRENT_USER() value corresponds not to the client user but to a different account. This occurs in contexts when privilege checking is not based the client's account:

- Stored routines (procedures and functions) defined with the SQL SECURITY DEFINER characteristic.
- Views defined with the SQL SECURITY DEFINER characteristic (as of MySQL 5.0.24).
- Triggers (as of MySQL 5.0.17).

In those contexts, privilege checking is done against the DEFINER account and CURRENT_USER() refers to that account, not to the account for the client who invoked the stored routine or view or who caused the trigger to activate. To determine the invoking user, you can call the USER() function, which returns a

This documentation is for an older version. If you're

value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the `user` table, because the `USER()` value never contains wildcards, whereas account values (as returned by `CURRENT_USER()`) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of `''@'localhost'` enables clients to connect as an anonymous user from the local host with any user name. In this case, if a client connects as `user1` from the local host, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----------------+----------------+
| USER()          | CURRENT_USER() |
+-----------------+----------------+
| user1@localhost | @localhost     |
+-----------------+----------------+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER()` value will not indicate which one. For example, the account `'user2'@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+--------------------------+---------------------+
| USER()                   | CURRENT_USER()      |
+--------------------------+---------------------+
| user2@remote.example.com | user2@%.example.com |
+--------------------------+---------------------+
```

If an application must invoke `USER()` for user auditing (for example, if it does auditing from within triggers) but must also be able to associate the `USER()` value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `''@'localhost'` and `'user2'@'%.example.com'` accounts should be changed not to use wildcards:

```
RENAME USER ''@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';
```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` or `USER()` value, use the `SUBSTRING_INDEX()` function:

```
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',1);
+---------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',1) |
+---------------------------------------+
| user1                                 |
+---------------------------------------+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',-1);
+----------------------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| SUBSTRING_INDEX(CURRENT_USER(),'@',-1) |
+----------------------------------------+
| localhost                              |
+----------------------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 7 Backup and Recovery

Table of Contents

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replication slave servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

- Types of backups: Logical versus physical, full versus incremental, and so forth
- Methods for creating backups
- Recovery methods, including point-in-time recovery
- Backup scheduling, compression, and encryption
- Table maintenance, to enable recovery of corrupt tables

Additional Resources

Resources related to backup or to maintaining data availability include the following:

- A forum dedicated to backup issues is available at http://forums.mysql.com/list.php?28.
- Details for `mysqldump`, `mysqlhotcopy`, and other MySQL backup programs can be found in Chapter 4, *MySQL Programs*.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- The syntax of the SQL statements described here is given in Chapter 13, *SQL Statement Syntax*.
- For additional information about `InnoDB` backup procedures, see Section 14.2.6, “Backing Up and Recovering an InnoDB Database”.
- Replication enables you to maintain identical data on multiple servers. This has several benefits, such as enabling client query load to be distributed over servers, availability of data even if a given server is taken offline or fails, and the ability to make backups with no impact on the master by using a slave server. See Chapter 16, *Replication*.
- MySQL Cluster provides a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. See Chapter 17, *MySQL Cluster*. For information specifically about MySQL Cluster backup, see Section 17.5.3, “Online Backup of MySQL Cluster”.

7.1 Backup and Recovery Types

This section describes the characteristics of different types of backups.

Logical Versus Physical (Raw) Backups

Logical backups save information represented as logical database structure (`CREATE DATABASE`, `CREATE TABLE` statements) and content (`INSERT` statements or delimited-text files). Physical backups consist of raw copies of the directories and files that store database contents.

Logical backup methods have these characteristics:

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.
- Logical backup tools include the `mysqldump` program and the `SELECT ... INTO OUTFILE` statement. These work for any storage engine, even `MEMORY`.
- To restore logical backups, SQL-format dump files can be processed using the `mysql` client. To load delimited-text files, use the `LOAD DATA INFILE` statement or the `mysqlimport` client.

Physical backup methods have these characteristics:

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory. Data from `MEMORY` tables cannot be backed up this way because their contents are not stored on disk.
- Physical backup methods are faster than logical because they involve only file copying without conversion.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Output is more compact than for logical backup.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. (Each `MyISAM` table corresponds uniquely to a set of files, but an `InnoDB` table shares file storage with other `InnoDB` tables.)
- In addition to databases, the backup can include any related files such as log or configuration files.
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup.
- Physical backup tools include file system-level commands (such as `cp`, `scp`, `tar`, `rsync`), `mysqlhotcopy` for `MyISAM` tables, `ibbackup` for `InnoDB` tables, or `START BACKUP` for `NDB` tables.
- For restore, files copied at the file system level or with `mysqlhotcopy` can be copied back to their original locations with file system commands; `ibbackup` restores `InnoDB` tables, and `ndb_restore` restores `NDB` tables.

Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

- The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
- Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity.

Offline backup methods have these characteristics:

- Clients can be affected adversely because the server is unavailable during backup.
- The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients will be affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server. host.

- `mysqldump` can connect to local or remote servers. For SQL output (`CREATE` and `INSERT` statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the `--tab` option), data files are created on the server host.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `mysqlhotcopy` performs only local backups: It connects to the server to lock it against data modifications and then copies local table files.
- `SELECT ... INTO OUTFILE` can be initiated from a local or remote client host, but the output file is created on the server host.
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section. Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

Table Maintenance

Data integrity can be compromised if tables become corrupt. MySQL provides programs for checking `MyISAM` tables and repairing them should problems be found. See Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. `ibbackup` can compress `InnoDB` backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

7.2 Database Backup Methods

This section summarizes some general methods for making backups.

Making Backups with `mysqldump` or `mysqlhotcopy`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `mysqldump` program and the `mysqlhotcopy` script can make backups. `mysqldump` is more general because it can back up all kinds of tables. `mysqlhotcopy` works only with some storage engines. (See Section 7.4, “Using mysqldump for Backups”, and Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”.)

For `InnoDB` tables, it is possible to perform an online backup that takes no locks on tables using the `--single-transaction` option to `mysqldump`. See Section 7.3.1, “Establishing a Backup Policy”.

Making Backups by Copying Table Files

For storage engines that represent each table using its own files, tables can be backed up by copying those files. For example, `MyISAM` tables are stored as files, so it is easy to do a backup by copying files (`*.frm`, `*.MYD`, and `*.MYI` files). To get a consistent backup, stop the server or lock and flush the relevant tables:

```
LOCK TABLES tbl_list READ;
FLUSH TABLES tbl_list;
```

You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The `FLUSH TABLES` statement is needed to ensure that the all active index pages are written to disk before you start the backup. See Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”, and Section 13.7.6.2, “FLUSH Syntax”.

You can also create a binary backup simply by copying all table files, as long as the server isn't updating anything. The `mysqlhotcopy` script uses this method. (But note that table file copying methods do not work if your database contains `InnoDB` tables. `mysqlhotcopy` does not work for `InnoDB` tables because `InnoDB` does not necessarily store table contents in database directories. Also, even if the server is not actively updating data, `InnoDB` may still have modified data cached in memory and not flushed to disk.)

Making Delimited-Text File Backups

To create a text file containing a table's data, you can use `SELECT * INTO OUTFILE 'file_name' FROM tbl_name`. The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See Section 13.2.8, “SELECT Syntax”. This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing `CREATE TABLE` statements for the backed up tables) is to use `mysqldump` with the `--tab` option. See Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”.

To reload a delimited-text data file, use `LOAD DATA INFILE` or `mysqlimport`.

Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups: You must start the server with the `--log-bin` option to enable binary logging; see Section 5.4.3, “The Binary Log”. The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”. The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS`, `mysqldump --flush-logs`, or `mysqlhotcopy --flushlog`. See Section 4.5.4,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

"`mysqldump` — A Database Backup Program", and Section 4.6.9, "`mysqlhotcopy` — A Database Backup Program".

Making Backups Using Replication Slaves

If you have performance problems with your master server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See Section 16.3.1, "Using Replication for Backups".

If you are backing up a slave replication server, you should back up its `master.info` and `relay-log.info` files when you back up the slave's databases, regardless of the backup method you choose. These information files are always needed to resume replication after you restore the slave's data. If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` option. If the server was not started with that option, the directory location is the value of the `tmpdir` system variable.

Recovering Corrupt Tables

If you have to restore `MyISAM` tables that have become corrupt, try to recover them using `REPAIR TABLE` or `myisamchk -r` first. That should work in 99.9% of all cases. If `myisamchk` fails, see Section 7.6, "MyISAM Table Maintenance and Crash Recovery".

Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

7.3 Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

- Operating system crash
- Power failure
- File system crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as `--user` and `--password` for the `mysqldump` and `mysql` client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Assume that data is stored in the InnoDB storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The InnoDB data files might not contain consistent data due to the crash, but InnoDB reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. InnoDB automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

7.3.1 Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, InnoDB Hot Backup provides online nonblocking physical backup of the InnoDB data files, and mysqldump provides online logical backup. This discussion uses mysqldump.

Assume that we make a full backup of all our InnoDB tables in all databases using the following command on Sunday at 1 p.m., when load is low:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

The resulting .sql file produced by mysqldump contains a set of SQL INSERT statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using FLUSH TABLES WITH READ LOCK). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are `InnoDB` tables, so `--single-transaction` uses a consistent read and guarantees that data seen by `mysqldump` does not change. (Changes made by other clients to `InnoDB` tables are not seen by the `mysqldump` process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the `MyISAM` tables in the `mysql` database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the `--log-bin` option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the `--log-bin` option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`, because the `--flush-logs` option causes the server to flush its logs. The `--master-data` option causes `mysqldump` to write binary log information to its output, so the resulting `.sql` dump file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The dump file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.
- All data changes logged after the backup are not present in the dump file, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another `mysqladmin flush-logs` command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note

Deleting the MySQL binary logs with `mysqldump --delete-master-logs` can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the `PURGE BINARY LOGS` statement explains what should be verified before deleting the MySQL binary logs. See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

7.3.2 Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin` option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the `gbichot2-bin.000009` file (and any subsequent files) at hand, and we could apply them using `mysqlbinlog` and `mysql` to restore the most recent data changes with no loss up to the moment of the crash:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

For more information about using `mysqlbinlog` to process binary log files, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

7.3.3 Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` option, or even `--log-bin=`*`log_name`*, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the `mysqldump` command shown earlier in Section 7.3.1, “Establishing a Backup Policy”, that makes an online, nonblocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or `mysqladmin flush-logs`.

7.4 Using mysqldump for Backups

This section describes how to use `mysqldump` to produce dump files, and how to reload dump files. A dump file can be used in several ways:

- As a backup to enable data recovery in case of data loss.
- As a source of data for setting up replication slaves.
- As a source of data for experimentation:
  - To make a copy of a database that you can use without changing the original data.
  - To test potential upgrade incompatibilities.

`mysqldump` produces two types of output, depending on whether the `--tab` option is given:

- Without `--tab`, `mysqldump` writes SQL statements to the standard output. This output consists of `CREATE` statements to create dumped objects (databases, tables, stored routines, and so forth), and `INSERT` statements to load data into tables. The output can be saved in a file and reloaded later using `mysql` to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.
- With `--tab`, `mysqldump` produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named *`tbl_name`*`.txt` in the output directory. The server also sends a `CREATE TABLE` statement for the table to `mysqldump`, which writes it as a file named *`tbl_name`*`.sql` in the output directory.

7.4.1 Dumping Data in SQL Format with mysqldump

This section describes how to use `mysqldump` to create SQL-format dump files. For information about reloading such dump files, see Section 7.4.2, “Reloading SQL-Format Backups”.

By default, `mysqldump` writes information as SQL statements to the standard output. You can save the output in a file:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqldump [arguments] > file_name
```

To dump all databases, invoke `mysqldump` with the `--all-databases` option:

```
shell> mysqldump --all-databases > dump.sql
```

To dump only specific databases, name them on the command line and use the `--databases` option:

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

The `--databases` option causes all names on the command line to be treated as database names. Without this option, `mysqldump` treats the first name as a database name and those following as table names.

With `--all-databases` or `--databases`, `mysqldump` writes `CREATE DATABASE` and `USE` statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the `--add-drop-database` option as well. In this case, `mysqldump` writes a `DROP DATABASE` statement preceding each `CREATE DATABASE` statement.

To dump a single database, name it on the command line:

```
shell> mysqldump --databases test > dump.sql
```

In the single-database case, it is permissible to omit the `--databases` option:

```
shell> mysqldump test > dump.sql
```

The difference between the two preceding commands is that without `--databases`, the dump output contains no `CREATE DATABASE` or `USE` statements. This has several implications:

- When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
- For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
- If the database to be reloaded does not exist, you must create it first.
- Because the output will contain no `CREATE DATABASE` statement, the `--add-drop-database` option has no effect. If you use it, it produces no `DROP DATABASE` statement.

To dump only specific tables from a database, name them on the command line following the database name:

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

7.4.2 Reloading SQL-Format Backups

To reload a dump file written by `mysqldump` that consists of SQL statements, use it as input to the `mysql` client. If the dump file was created by `mysqldump` with the `--all-databases` or `--databases` option, it contains `CREATE DATABASE` and `USE` statements and it is not necessary to specify a default database into which to load the data:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql < dump.sql
```

Alternatively, from within `mysql`, use a `source` command:

```
mysql> source dump.sql
```

If the file is a single-database dump not containing `CREATE DATABASE` and `USE` statements, create the database first (if necessary):

```
shell> mysqladmin create db1
```

Then specify the database name when you load the dump file:

```
shell> mysql db1 < dump.sql
```

Alternatively, from within `mysql`, create the database, select it as the default database, and load the dump file:

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

7.4.3 Dumping Data in Delimited-Text Format with mysqldump

This section describes how to use `mysqldump` to create delimited-text dump files. For information about reloading such dump files, see Section 7.4.4, “Reloading Delimited-Text Format Backups”.

If you invoke `mysqldump` with the `--tab=dir_name` option, it uses *`dir_name`* as the output directory and dumps tables individually in that directory using two files for each table. The table name is the base name for these files. For a table named `t1`, the files are named `t1.sql` and `t1.txt`. The `.sql` file contains a `CREATE TABLE` statement for the table. The `.txt` file contains the table data, one line per table row.

The following command dumps the contents of the `db1` database to files in the `/tmp` database:

```
shell> mysqldump --tab=/tmp db1
```

The `.txt` files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses `SELECT ... INTO OUTFILE` to write the files, so you must have the `FILE` privilege to perform this operation, and an error occurs if a given `.txt` file already exists.

The server sends the `CREATE` definitions for dumped tables to `mysqldump`, which writes them to `.sql` files. These files therefore are owned by the user who executes `mysqldump`.

It is best that `--tab` be used only for dumping a local server. If you use it with a remote server, the `--tab` directory must exist on both the local and remote hosts, and the `.txt` files will be written by the server in the remote directory (on the server host), whereas the `.sql` files will be written by `mysqldump` in the local directory (on the client host).

For `mysqldump --tab`, the server by default writes table data to `.txt` files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for `SELECT ... INTO OUTFILE`.)

To enable data files to be written using a different format, `mysqldump` supports these options:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--fields-terminated-by=str`

  The string for separating column values (default: tab).

- `--fields-enclosed-by=char`

  The character within which to enclose column values (default: no character).

- `--fields-optionally-enclosed-by=char`

  The character within which to enclose non-numeric column values (default: no character).

- `--fields-escaped-by=char`

  The character for escaping special characters (default: no escaping).

- `--lines-terminated-by=str`

  The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want `mysqldump` to quote column values within double quotation marks. To do so, specify double quote as the value for the `--fields-enclosed-by` option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

```
--fields-enclosed-by='"'
```

On any platform, you can specify the value in hex:

```
--fields-enclosed-by=0x22
```

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (`\r\n`), use this command (enter it on a single line):

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1
```

Should you use any of the data-formatting options to dump table data, you will need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

7.4.4 Reloading Delimited-Text Format Backups

For backups produced with `mysqldump --tab`, each table is represented in the output directory by an `.sql` file containing the `CREATE TABLE` statement for the table, and a `.txt` file containing the table data. To reload a table, first change location into the output directory. Then process the `.sql` file with `mysql` to create an empty table and process the `.txt` file to load the data into the table:

```
shell> mysql db1 < t1.sql
shell> mysqlimport db1 t1.txt
```

An alternative to using `mysqlimport` to load the data file is to use the `LOAD DATA INFILE` statement from within the `mysql` client:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

If you used any data-formatting options with `mysqldump` when you initially dumped the table, you must use the same options with `mysqlimport` or `LOAD DATA INFILE` to ensure proper interpretation of the data file contents:

```
shell> mysqlimport --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt
```

Or:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1
    -> FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

7.4.5 mysqldump Tips

This section surveys techniques that enable you to use `mysqldump` to solve specific problems:

- How to make a copy a database
- How to copy a database from one server to another
- How to dump stored programs (stored procedures and functions and triggers)
- How to dump definitions and data separately

7.4.5.1 Making a Copy of a Database

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

Do not use `--databases` on the `mysqldump` command line because that causes `USE db1` to be included in the dump file, which overrides the effect of naming `db2` on the `mysql` command line.

7.4.5.2 Copy a Database from one Server to Another

On Server 1:

```
shell> mysqldump --databases db1 > dump.sql
```

Copy the dump file from Server 1 to Server 2.

On Server 2:

```
shell> mysql < dump.sql
```

Use of `--databases` with the `mysqldump` command line causes the dump file to include `CREATE DATABASE` and `USE` statements that create the database if it does exist and make it the default database for the reloaded data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Alternatively, you can omit `--databases` from the `mysqldump` command. Then you will need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

```
shell> mysqldump db1 > dump.sql
```

On Server 2:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

You can specify a different database name in this case, so omitting `--databases` from the `mysqldump` command enables you to dump data from one database and load it into another.

7.4.5.3 Dumping Stored Programs

Several options control how `mysqldump` handles stored programs (stored procedures and functions and triggers):

- `--routines`: Dump stored procedures and functions
- `--triggers`: Dump triggers for tables

The `--triggers` option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: `--skip-routines` or `--skip-triggers`.

7.4.5.4 Dumping Table Definitions and Content Separately

The `--no-data` option tells `mysqldump` not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the `--no-create-info` option tells `mysqldump` to suppress `CREATE` statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the `test` database, use these commands:

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

For a definition-only dump, add the `--routines` option to also include stored routine definitions:

```
shell> mysqldump --no-data --routines test > dump-defs.sql
```

7.4.5.5 Using mysqldump to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqldump --all-databases --no-data --routines > dump-defs.sql
```

On the upgraded server:

```
shell> mysql < dump-defs.sql
```

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

On the upgraded server:

```
shell> mysql < dump-data.sql
```

Now check the table contents and run some test queries.

7.5 Point-in-Time (Incremental) Recovery Using the Binary Log

Point-in-time recovery refers to recovery of data changes made since a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in Section 7.2, “Database Backup Methods”.) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

Point-in-time recovery is based on these principles:

- The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the `--log-bin` option to enable binary logging (see Section 5.4.3, “The Binary Log”).

  To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the `--log-bin` option to place the files in a different location. Section 5.4.3, “The Binary Log”.

  To see a listing of all binary log files, use this statement:

  ```
  mysql> SHOW BINARY LOGS;
  ```

  To determine the name of the current binary log file, issue the following statement:

  ```
  mysql> SHOW MASTER STATUS;
  ```

- The `mysqlbinlog` utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. `mysqlbinlog` has options for selecting sections of the binary log based on event times or position of events within the log. See Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process `mysqlbinlog` output using the `mysql` client:

```
shell> mysqlbinlog binlog_files | mysql -u root -p
```

- Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send `mysqlbinlog` output into a paging program:

```
shell> mysqlbinlog binlog_files | more
```

  Alternatively, save the output in a file and view the file in a text editor:

```
shell> mysqlbinlog binlog_files > tmpfile
shell> ... edit tmpfile ...
```

- Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental `DROP DATABASE`. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

```
shell> mysql -u root -p < tmpfile
```

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

7.5.1 Point-in-Time Recovery Using Event Times

To indicate the start and end times for recovery, specify the `--start-datetime` and `--stop-datetime` options for `mysqlbinlog`, in `DATETIME` format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-datetime` option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 10:01:00" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the `/tmp/mysql_restore.sql` file with a text editor to examine it.

Excluding specific changes by specifying times for `mysqlbinlog` does not work well if multiple statements executed at the same time as the one to be excluded.

7.5.2 Point-in-Time Recovery Using Event Positions

Instead of specifying dates and times, the `--start-position` and `--stop-position` options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 9:55:00" \
        --stop-datetime="2005-04-20 10:05:00" \
        /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \
        | mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \
        | mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

7.6 MyISAM Table Maintenance and Crash Recovery

This section discusses how to use `myisamchk` to check or repair `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). For general `myisamchk` background, see Section 4.6.3, "`myisamchk` — MyISAM Table-Maintenance Utility". Other table-repair information can be found at Section 2.19.4, "Rebuilding or Repairing Tables or Indexes".

You can use `myisamchk` to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using `myisamchk` to get information about your tables, see Section 4.6.3.5, "Obtaining Table Information with myisamchk".

Even though table repair with `myisamchk` is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

`myisamchk` operations that affect indexes can cause `FULLTEXT` indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in Section 4.6.3.1, "myisamchk General Options".

`MyISAM` table maintenance can also be done using the SQL statements that perform operations similar to what `myisamchk` can do:

- To check `MyISAM` tables, use `CHECK TABLE`.
- To repair `MyISAM` tables, use `REPAIR TABLE`.
- To optimize `MyISAM` tables, use `OPTIMIZE TABLE`.
- To analyze `MyISAM` tables, use `ANALYZE TABLE`.

For additional information about these statements, see Section 13.7.2, "Table Maintenance Statements".

These statements can be used directly or by means of the `mysqlcheck` client program. One advantage of these statements over `myisamchk` is that the server does all the work. With `myisamchk`, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between `myisamchk` and the server.

7.6.1 Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See Section B.5.3.3, "What to Do If MySQL Keeps Crashing".

For an explanation of how `MyISAM` tables can become corrupted, see Section 14.1.4, "MyISAM Table Problems".

If you run `mysqld` with external locking disabled (which is the default), you cannot reliably use `myisamchk` to check a table when `mysqld` is using the same table. If you can be certain that no one will access the tables through `mysqld` while you run `myisamchk`, you only have to execute `mysqladmin flush-tables` before you start checking the tables. If you cannot guarantee this, you must stop `mysqld` while you check the tables. If you run `myisamchk` to check tables that `mysqld` is updating at the same time, you may get a warning that a table is corrupt even when it is not.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the server is run with external locking enabled, you can use `myisamchk` to check tables at any time. In this case, if the server tries to update a table that `myisamchk` is using, the server will wait for `myisamchk` to finish before it continues.

If you use `myisamchk` to repair or optimize tables, you *must* always ensure that the `mysqld` server is not using the table (this also applies if external locking is disabled). If you do not stop `mysqld`, you should at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

When performing crash recovery, it is important to understand that each `MyISAM` table *`tbl_name`* in a database corresponds to the three files in the database directory shown in the following table.

File	Purpose
`tbl_name``.frm`	Definition (format) file
`tbl_name``.MYD`	Data file
`tbl_name``.MYI`	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

`myisamchk` works by creating a copy of the `.MYD` data file row by row. It ends the repair stage by removing the old `.MYD` file and renaming the new file to the original file name. If you use `--quick`, `myisamchk` does not create a temporary `.MYD` file, but instead assumes that the `.MYD` file is correct and generates only a new index file without touching the `.MYD` file. This is safe, because `myisamchk` automatically detects whether the `.MYD` file is corrupt and aborts the repair if it is. You can also specify the `--quick` option twice to `myisamchk`. In this case, `myisamchk` does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the `.MYD` file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running `myisamchk`.

7.6.2 How to Check MyISAM Tables for Errors

To check a `MyISAM` table, use the following commands:

- `myisamchk` *`tbl_name`*

  This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run `myisamchk` without options or with the `-s` (silent) option.

- `myisamchk -m` *`tbl_name`*

  This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e` *`tbl_name`*

  This does a complete and thorough check of all data (`-e` means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, `myisamchk` stops after the first error it finds. If you want to obtain more information, you can add the `-v` (verbose) option. This causes `myisamchk` to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i` *`tbl_name`*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This is like the previous command, but the `-i` option tells `myisamchk` to print additional statistical information.

In most cases, a simple `myisamchk` command with no arguments other than the table name is sufficient to check a table.

7.6.3 How to Repair MyISAM Tables

The discussion in this section describes how to use `myisamchk` on `MyISAM` tables (extensions `.MYI` and `.MYD`).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.7.2.3, “CHECK TABLE Syntax”, and Section 13.7.2.6, “REPAIR TABLE Syntax”.

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- *`tbl_name`*`.frm` is locked against change
- Can't find file *`tbl_name`*`.MYI` (Errcode: *`nnn`*)
- Unexpected end of file
- Record file is crashed
- Got error *`nnn`* from table handler

To get more information about the error, run `perror` *`nnn`*, where *`nnn`* is the error number. The following example shows how to use `perror` to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use `ALTER TABLE` to increase the `MAX_ROWS` and `AVG_ROW_LENGTH` table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use `SHOW CREATE TABLE`.

For the other errors, you must repair your tables. `myisamchk` can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that `mysqld` runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This section is for the cases where a table check fails (such as those described in Section 7.6.2, “How to Check MyISAM Tables for Errors”), or you want to use the extended features that `myisamchk` provides.

The `myisamchk` options used for table maintenance with are described in Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”. `myisamchk` also has variables that you can set to control memory allocation that may improve performance. See Section 4.6.3.6, “myisamchk Memory Usage”.

If you are going to repair a table from the command line, you must first stop the `mysqld` server. Note that when you do `mysqladmin shutdown` on a remote server, the `mysqld` server is still available for a while after `mysqladmin` returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run `myisamchk *.MYI` or `myisamchk -e *.MYI` if you have more time. Use the `-s` (silent) option to suppress unnecessary information.

If the `mysqld` server is stopped, you should use the `--update-state` option to tell `myisamchk` to mark the table as “checked.”

You have to repair only those tables for which `myisamchk` announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `myisamchk -r -q tbl_name` (`-r -q` means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `myisamchk -r tbl_name` (`-r` means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use `myisamchk --safe-recover tbl_name`. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

Note

If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size` and `key_buffer_size` variables each to about 25% of your available memory when running `myisamchk`.

If you get unexpected errors when repairing (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.

This documentation is for an older version. If you're

2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

Important

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See Section 13.7.2.6, “REPAIR TABLE Syntax”.

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `myisamchk -r`.

2. If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

7.6.4 MyISAM Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See Section 13.7.2.5, “OPTIMIZE TABLE Syntax”.

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze` or `-a`: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.

- `--sort-index` or `-S`: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--sort-records=index_num` or `-R index_num`: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index.

For a full description of all available options, see Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

7.6.5 Setting Up a MyISAM Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair `MyISAM` tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. See Section 13.7.2, “Table Maintenance Statements”.

Another way to check tables is to use `myisamchk`. For maintenance purposes, you can use `myisamchk -s`. The `-s` option (short for `--silent`) causes `myisamchk` to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic `MyISAM` table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To cause the server to check `MyISAM` tables automatically, start it with the `--myisam-recover` option. See Section 5.1.3, “Server Command Options”.

You should also check your tables regularly during normal system operation. For example, you can run a `cron` job to check important tables once a week, using a line like this in a `crontab` file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute `myisamchk -s` each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to `MyISAM` tables with dynamic-sized rows (tables with `VARCHAR`, `BLOB`, or `TEXT` columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using `OPTIMIZE TABLE` on the tables in question. Alternatively, if you can stop the `mysqld` server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --sort_buffer_size=16M */*.MYI
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 8 Optimization

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This chapter explains how to optimize MySQL performance and provides examples. Optimization involves configuring, tuning, and measuring performance, at several levels. Depending on your job role (developer, DBA, or a combination of both), you might optimize at the level of individual SQL statements, entire applications, a single database server, or multiple networked database servers. Sometimes you can be proactive and plan in advance for performance, while other times you might troubleshoot a configuration or code issue after a problem occurs. Optimizing CPU and memory usage can also improve scalability, allowing the database to handle more load without slowing down.

8.1 Optimization Overview

Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level, which you must minimize and make as efficient as possible. As you work on database performance, you start by learning the high-level rules and guidelines for the software side, and measuring performance using wall-clock time. As you become an expert, you learn more about what happens internally, and start measuring things such as CPU cycles and I/O operations.

Typical users aim to get the best database performance out of their existing software and hardware configurations. Advanced users look for opportunities to improve the MySQL software itself, or develop their own storage engines and hardware appliances to expand the MySQL ecosystem.

Optimizing at the Database Level

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work? For example, applications that perform

This documentation is for an older version. If you're
This documentation is for an older version. If you're

frequent updates often have many tables with few columns, while applications that analyze large amounts of data often have few tables with many columns.

- Are the right indexes in place to make queries efficient?
- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use? In particular, the choice of a nontransactional storage engine such as `MyISAM` or a transactional one such as `InnoDB` can be very important for performance and scalability.
- Does each table use an appropriate row format? This choice also depends on the storage engine used for the table. In particular, compressed tables use less disk space and so require less disk I/O to read and write the data. Compression is available for read-only `MyISAM` tables, and for all kinds of workloads with `InnoDB` tables.
- Does the application use an appropriate locking strategy? For example, by allowing shared access when possible so that database operations can run concurrently, and requesting exclusive access when appropriate so that critical operations get top priority. Again, the choice of storage engine is significant. The `InnoDB` storage engine handles most locking issues without involvement from you, allowing for better concurrency in the database and reducing the amount of experimentation and tuning for your code.
- Are all memory areas used for caching sized correctly? That is, large enough to hold frequently accessed data, but not so large that they overload physical memory and cause paging. The main memory areas to configure are the `MyISAM` key cache, the `InnoDB` buffer pool, and the MySQL query cache.

Optimizing at the Hardware Level

Any database application eventually hits hardware limits as the database becomes more and more busy. A DBA must evaluate whether it is possible to tune the application or reconfigure the server to avoid these bottlenecks, or whether more hardware resources are required. System bottlenecks typically arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read or write the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When the data is in main memory, we must process it to get our result. Having large tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

Balancing Portability and Performance

Because all SQL servers implement different parts of standard SQL, it takes work to write portable database applications. It is very easy to achieve portability for very simple selects and inserts, but becomes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

more difficult the more capabilities you require. If you want an application that is fast with many database systems, it becomes even more difficult.

All database systems have some weak points. That is, they have different design compromises that lead to different behavior.

To make a complex application portable, you need to determine which SQL servers it must work with, and then determine what features those servers support. You can use the MySQL `crash-me` program to find functions, types, and limits that you can use with a selection of database servers. `crash-me` does not check for every possible feature, but it is still reasonably comprehensive, performing about 450 tests. An example of the type of information `crash-me` can provide is that you should not use column names that are longer than 18 characters if you want to be able to use Informix or DB2.

The `crash-me` program and the MySQL benchmarks are all very database independent. By taking a look at how they are written, you can get a feeling for what you must do to make your own applications database independent. The programs can be found in the `sql-bench` directory of MySQL source distributions. They are written in Perl and use the DBI database interface. Use of DBI in itself solves part of the portability problem because it provides database-independent access methods. See Section 8.13.2, “The MySQL Benchmark Suite”.

If you strive for database independence, you need to get a good feeling for each SQL server's bottlenecks. For example, MySQL is very fast in retrieving and updating rows for `MyISAM` tables, but has a problem in mixing slow readers and writers on the same table. Transactional database systems in general are not very good at generating summary tables from log tables, because in this case row locking is almost useless.

To make your application *really* database independent, you should define an easily extendable interface through which you manipulate your data. For example, C++ is available on most systems, so it makes sense to use a C++ class-based interface to the databases.

If you use some feature that is specific to a given database system (such as the `REPLACE` statement, which is specific to MySQL), you should implement the same feature for other SQL servers by coding an alternative method. Although the alternative might be slower, it enables the other servers to perform the same tasks.

To use performance-oriented SQL extensions in a portable MySQL program, you can wrap MySQL-specific keywords in a statement within `/*! */` comment delimiters. Other SQL servers ignore the commented keywords. For information about writing comments, see Section 9.6, “Comment Syntax”.

If high performance is more important than exactness, as for some Web applications, it is possible to create an application layer that caches all results to give you even higher performance. By letting old results expire after a while, you can keep the cache reasonably fresh. This provides a method to handle high load spikes, in which case you can dynamically increase the cache size and set the expiration timeout higher until things get back to normal.

In this case, the table creation information should contain information about the initial cache size and how often the table should normally be refreshed.

An attractive alternative to implementing an application cache is to use the MySQL query cache. By enabling the query cache, the server handles the details of determining whether a query result can be reused. This simplifies your application. See Section 8.10.3, “The MySQL Query Cache”.

8.2 Optimizing SQL Statements

The core logic of a database application is performed through SQL statements, whether issued directly through an interpreter or submitted behind the scenes through an API. The tuning guidelines in this section help to speed up all kinds of MySQL applications. The guidelines cover SQL operations that read and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

write data, the behind-the-scenes overhead for SQL operations in general, and operations used in specific scenarios such as database monitoring.

8.2.1 Optimizing SELECT Statements

Queries, in the form of `SELECT` statements, perform all the lookup operations in the database. Tuning these statements is a top priority, whether to achieve sub-second response times for dynamic web pages, or to chop hours off the time to generate huge overnight reports.

Besides `SELECT` statements, the tuning techniques for queries also apply to constructs such as `CREATE TABLE...AS SELECT`, `INSERT INTO...SELECT`, and `WHERE` clauses in `DELETE` statements. Those statements have additional performance considerations because they combine write operations with the read-oriented query operations.

8.2.1.1 Speed of SELECT Statements

The main considerations for optimizing queries are:

- To make a slow `SELECT ... WHERE` query faster, the first thing to check is whether you can add an index. Set up indexes on columns used in the `WHERE` clause, to speed up evaluation, filtering, and the final retrieval of results. To avoid wasted disk space, construct a small set of indexes that speed up many related queries used in your application.

  Indexes are especially important for queries that reference different tables, using features such as joins and foreign keys. You can use the `EXPLAIN` statement to determine which indexes are used for a `SELECT`. See Section 8.3.1, “How MySQL Uses Indexes” and Section 8.8.1, “Optimizing Queries with EXPLAIN”.

- Isolate and tune any part of the query, such as a function call, that takes excessive time. Depending on how the query is structured, a function could be called once for every row in the result set, or even once for every row in the table, greatly magnifying any inefficiency.

- Minimize the number of full table scans in your queries, particularly for big tables.

- Keep table statistics up to date by using the `ANALYZE TABLE` statement periodically, so the optimizer has the information needed to construct an efficient execution plan.

- Learn the tuning techniques, indexing techniques, and configuration parameters that are specific to the storage engine for each table. Both `InnoDB` and `MyISAM` have sets of guidelines for enabling and sustaining high performance in queries. For details, see Section 8.6.5, “Optimizing InnoDB Queries” and Section 8.5.1, “Optimizing MyISAM Queries”.

- Avoid transforming the query in ways that make it hard to understand, especially if the optimizer does some of the same transformations automatically.

- If a performance issue is not easily solved by one of the basic guidelines, investigate the internal details of the specific query by reading the `EXPLAIN` plan and adjusting your indexes, `WHERE` clauses, join clauses, and so on. (When you reach a certain level of expertise, reading the `EXPLAIN` plan might be your first step for every query.)

- Adjust the size and properties of the memory areas that MySQL uses for caching. With efficient use of the `MyISAM` key cache, `InnoDB` buffer pool, and the MySQL query cache, repeated queries run faster because the results are retrieved from memory the second and subsequent times.

- Even for a query that runs fast using the cache memory areas, you might still optimize further so that they require less cache memory, making your application more scalable. Scalability means that your

This documentation is for an older version. If you're
This documentation is for an older version. If you're

application can handle more simultaneous users, larger requests, and so on without experiencing a big drop in performance.

- Deal with locking issues, where the speed of your query might be affected by other sessions accessing the tables at the same time.

8.2.1.2 How MySQL Optimizes WHERE Clauses

This section discusses optimizations that can be made for processing `WHERE` clauses. The examples use `SELECT` statements, but the same optimizations apply for `WHERE` clauses in `DELETE` and `UPDATE` statements.

Work on the MySQL optimizer is ongoing, so this section is incomplete. MySQL performs a great many optimizations, not all of which are documented here.

Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
   (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- `COUNT(*)` on a single table without a `WHERE` is retrieved directly from the table information for `MyISAM` and `MEMORY` tables. This is also done for any `NOT NULL` expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some `SELECT` statements are impossible and returns no rows.
- `HAVING` is merged with `WHERE` if you do not use `GROUP BY` or aggregate functions (`COUNT()`, `MIN()`, and so on).
- For each table in a join, a simpler `WHERE` is constructed to get a fast `WHERE` evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
  - An empty table or a table with one row.
  - A table that is used with a `WHERE` clause on a `PRIMARY KEY` or a `UNIQUE` index, where all index parts are compared to constant expressions and are defined as `NOT NULL`.

  All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM t1,t2
  WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in `ORDER BY` and `GROUP BY` clauses come from the same table, that table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table.
- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- MySQL can sometimes produce query results using data from the index, without consulting the table data. If all columns used from the index are numeric, only the index data is used to resolve the query.
- Before each row is output, those that do not match the `HAVING` clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

8.2.1.3 Range Optimization

The `range` access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give a detailed description of how intervals are extracted from the `WHERE` clause.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, denoted as *range conditions* rather than “intervals.”

The definition of a range condition for a single-part index is as follows:

- For both `BTREE` and `HASH` indexes, comparison of a key part with a constant value is a range condition when using the `=`, `<=>`, `IN()`, `IS NULL`, or `IS NOT NULL` operators.
- Additionally, for `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>`, `<`, `>=`, `<=`, `BETWEEN`, `!=`, or `<>` operators, or `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character.
- For all index types, multiple range conditions combined with `OR` or `AND` form a range condition.

“Constant value” in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a `const` or `system` table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the `WHERE` clause:

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;

SELECT * FROM t1
  WHERE key_col = 1
  OR key_col IN (15,18,20);

SELECT * FROM t1
  WHERE key_col LIKE 'ab%'
  OR key_col BETWEEN 'bar' AND 'foo';
```

Some nonconstant values may be converted to constants during the optimizer constant propagation phase.

MySQL tries to extract range conditions from the `WHERE` clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where `key1` is an indexed column and `nonkey` is not indexed:

```
SELECT * FROM t1 WHERE
  (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
  (key1 < 'bar' AND nonkey = 4) OR
  (key1 < 'uux' AND key1 > 'z');
```

The extraction process for key `key1` is as follows:

1. Start with original `WHERE` clause:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z')
```

2. Remove `nonkey = 4` and `key1 LIKE '%b'` because they cannot be used for a range scan. The correct way to remove them is to replace them with `TRUE`, so that we do not miss any matching rows when doing the range scan. Having replaced them with `TRUE`, we get:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
   (key1 < 'bar' AND TRUE) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

3. Collapse conditions that are always true or false:

   - `(key1 LIKE 'abcde%' OR TRUE)` is always true
   - `(key1 < 'uux' AND key1 > 'z')` is always false

   Replacing these conditions with constants, we get:

   ```
   (key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
   ```

   Removing unnecessary `TRUE` and `FALSE` constants, we obtain:

   ```
   (key1 < 'abc') OR (key1 < 'bar')
   ```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

   ```
   (key1 < 'bar')
   ```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested `AND`/`OR` constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in `WHERE` clause.

MySQL does not support merging multiple ranges for the `range` access method for spatial indexes. To work around this limitation, you can use a `UNION` with identical `SELECT` statements, except that you put each spatial predicate in a different `SELECT`.

The Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

```
key_part1  key_part2  key_part3
  NULL       1          'abc'
  NULL       1          'xyz'
  NULL       2          'foo'
   1         1          'abc'
   1         1          'xyz'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  1         2          'abc'
  2         1          'aaa'
```

The condition *`key_part1 = 1`* defines this interval:

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition *`key_part3 = 'abc'`* does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For `HASH` indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

  ```
     key_part1 cmp const1
  AND key_part2 cmp const2
  AND ...
  AND key_partN cmp constN;
  ```

  Here, *`const1`*, *`const2`*, ... are constants, *`cmp`* is one of the `=`, `<=>`, or `IS NULL` comparison operators, and the conditions cover all index parts. (That is, there are *`N`* conditions, one for each part of an *`N`*-part index.) For example, the following is a range condition for a three-part `HASH` index:

  ```
  key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
  ```

  For the definition of what is considered to be a constant, see The Range Access Method for Single-Part Indexes.

- For a `BTREE` index, an interval might be usable for conditions combined with `AND`, where each condition compares a key part with a constant value using `=`, `<=>`, `IS NULL`, `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE 'pattern'` (where *`'pattern'`* does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if `<>` or `!=` is used).

  The optimizer attempts to use additional key parts to determine the interval as long as the comparison operator is `=`, `<=>`, or `IS NULL`. If the operator is `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE`, the optimizer uses it but considers no more key parts. For the following expression, the optimizer uses `=` from the first comparison. It also uses `>=` from the second comparison but considers no further key parts and does not use the third comparison for interval construction:

  ```
  key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
  ```

  The single interval is:

  ```
  ('foo',10,-inf) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
  ```

  It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value `('foo', 11, 0)`, which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with `OR`, they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are

This documentation is for an older version. If you're
This documentation is for an older version. If you're

combined with AND, they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

The intervals are:

```
(1,-inf) < (key_part1,key_part2) < (1,2)
(5,-inf) < (key_part1,key_part2)
```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The key_len column in the EXPLAIN output indicates the maximum length of the key prefix used.

In some cases, key_len may indicate that a key part was used, but that might be not what you would expect. Suppose that *key_part1* and *key_part2* can be NULL. Then the key_len column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

The Range Access Method for Single-Part Indexes, describes how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index. Analogous steps are performed for range conditions on multiple-part indexes.

8.2.1.4 Index Merge Optimization

The *Index Merge* method is used to retrieve rows with several range scans and to merge their results into one. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans. This access method merges index scans from a single table; it does not merge scans across multiple tables.

Note

If you have upgraded from a previous version of MySQL, you should be aware that this type of join optimization is first introduced in MySQL 5.0, and represents a significant change in behavior with regard to indexes. (Formerly, MySQL was able to use at most only one index for each referenced table.)

In EXPLAIN output, the Index Merge method appears as index_merge in the type column. In this case, the key column contains a list of indexes used, and key_len contains a list of the longest key parts for those indexes.

Examples:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
  WHERE (key1 = 10 OR key2 = 20) AND non_key=30;

SELECT * FROM t1, t2
  WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
  AND t2.key1=t1.some_col;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM t1, t2
  WHERE t1.key1=1
  AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

The Index Merge method has several access algorithms (seen in the `Extra` field of `EXPLAIN` output):

- `Using intersect(...)`
- `Using union(...)`
- `Using sort_union(...)`

The following sections describe these methods in greater detail.

Note

The Index Merge optimization algorithm has the following known deficiencies:

- If your query has a complex `WHERE` clause with deep `AND`/`OR` nesting and MySQL does not choose the optimal plan, try distributing terms using the following identity laws:

  ```
  (x AND y) OR z = (x OR z) AND (y OR z)
  (x OR y) AND z = (x AND z) OR (y AND z)
  ```

- Index Merge is not applicable to full-text indexes. We plan to extend it to cover these in a future MySQL release.
- If a range scan is possible on some key, the optimizer will not consider using Index Merge Union or Index Merge Sort-Union algorithms. For example, consider this query:

  ```
  SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
  ```

  For this query, two plans are possible:

  - An Index Merge scan using the `(goodkey1 < 10 OR goodkey2 < 20)` condition.
  - A range scan using the `badkey < 30` condition.

  However, the optimizer considers only the second plan.

The choice between different possible variants of the Index Merge access method and other access methods is based on cost estimates of various available options.

The Index Merge Intersection Access Algorithm

This access algorithm can be employed when a `WHERE` clause was converted to several range conditions on different keys combined with `AND`, and each condition is one of the following:

- In this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

  ```
  key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
  ```

- Any range condition over a primary key of an `InnoDB` or `BDB` table.

This documentation is for an older version. If you're

Examples:

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_col1=20;

SELECT * FROM tbl_name
  WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes do not cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over a primary key of an `InnoDB` or `BDB` table, it is not used for row retrieval, but is used to filter out rows retrieved using other conditions.

The Index Merge Union Access Algorithm

The applicability criteria for this algorithm are similar to those for the Index Merge method intersection algorithm. The algorithm can be employed when the table's `WHERE` clause was converted to several range conditions on different keys combined with `OR`, and each condition is one of the following:

- In this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

  ```
  key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
  ```

- Any range condition over a primary key of an `InnoDB` or `BDB` table.

- A condition for which the Index Merge method intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;

SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR
  (key3='foo' AND key4='bar') AND key5=5;
```

The Index Merge Sort-Union Access Algorithm

This access algorithm is employed when the `WHERE` clause was converted to several range conditions combined by `OR`, but for which the Index Merge method union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
  WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them before returning any rows.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

8.2.1.5 Engine Condition Pushdown Optimization

This optimization improves the efficiency of direct comparisons between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the storage engine for evaluation. This optimization can be used only by the `NDB` storage engine.

For MySQL Cluster, this optimization can eliminate the need to send nonmatching rows over the network between the cluster's data nodes and the MySQL Server that issued the query, and can speed up queries where it is used by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that a MySQL Cluster table is defined as follows:

```
CREATE TABLE t1 (
    a INT,
    b INT,
    KEY(a)
) ENGINE=NDB;
```

Condition pushdown can be used with queries such as the one shown here, which includes a comparison between a nonindexed column and a constant:

```
SELECT a, b FROM t1 WHERE b = 10;
```

The use of condition pushdown can be seen in the output of `EXPLAIN`:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where with pushed condition
```

However, condition pushdown *cannot* be used with either of these two queries:

```
SELECT a,b FROM t1 WHERE a = 10;
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

Condition pushdown is not applicable to the first query because an index exists on column `a`. (An index access method would be more efficient and so would be chosen in preference to condition pushdown.) Condition pushdown cannot be employed for the second query because the comparison involving the nonindexed column `b` is indirect. (However, condition pushdown could be applied if you were to reduce `b + 1 = 10` to `b = 9` in the `WHERE` clause.)

Condition pushdown may also be employed when an indexed column is compared with a constant using a `>` or `<` operator:

```
mysql> EXPLAIN SELECT a, b FROM t1 WHERE a < 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
         type: range
possible_keys: a
          key: a
      key_len: 5
          ref: NULL
         rows: 2
        Extra: Using where with pushed condition
```

Other supported comparisons for condition pushdown include the following:

- *`column`* `[NOT] LIKE` *`pattern`*

  `pattern` must be a string literal containing the pattern to be matched; for syntax, see Section 12.5.1, “String Comparison Functions”.

- *`column`* `IS [NOT] NULL`

- *`column`* `IN (`*`value_list`*`)`

  Each item in the *`value_list`* must be a constant, literal value.

- *`column`* `BETWEEN` *`constant1`* `AND` *`constant2`*

  `constant1` and *`constant2`* must each be a constant, literal value.

In all of the cases in the preceding list, it is possible for the condition to be converted into the form of one or more direct comparisons between a column and a constant.

Engine condition pushdown is disabled by default. To enable it at server startup, set the `engine_condition_pushdown` system variable. For example, in a `my.cnf` file, use these lines:

```
[mysqld]
engine_condition_pushdown=1
```

At runtime, enable condition pushdown with either of the following statements:

```
SET engine_condition_pushdown=ON;
```

```
SET engine_condition_pushdown=1;
```

Limitations. Engine condition pushdown is subject to the following limitations:

- Condition pushdown is supported only by the `NDB` storage engine.
- Columns may be compared with constants only; however, this includes expressions which evaluate to constant values.
- Columns used in comparisons cannot be of any of the `BLOB` or `TEXT` types.
- A string value to be compared with a column must use the same collation as the column.
- Joins are not directly supported; conditions involving multiple tables are pushed separately where possible. Use `EXPLAIN EXTENDED` to determine which conditions are actually pushed down.

8.2.1.6 IS NULL Optimization

MySQL can perform the same optimization on *`col_name`* `IS NULL` that it can use for *`col_name`* `=` *`constant_value`*. For example, MySQL can use indexes and ranges to search for `NULL` with `IS NULL`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
  WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a `WHERE` clause includes a *`col_name`* `IS NULL` condition for a column that is declared as `NOT NULL`, that expression is optimized away. This optimization does not occur in cases when the column might produce `NULL` anyway; for example, if it comes from a table on the right side of a `LEFT JOIN`.

MySQL can also optimize the combination *`col_name = expr`* `OR` *`col_name`* `IS NULL`, a form that is common in resolved subqueries. `EXPLAIN` shows `ref_or_null` when this optimization is used.

This optimization can handle one `IS NULL` for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns `a` and `b` of table `t2`:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
  WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
  OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

The optimization can handle only one `IS NULL` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL)
  OR (t1.b=t2.b AND t2.b IS NULL);
```

8.2.1.7 LEFT JOIN and RIGHT JOIN Optimization

MySQL implements an *`A`* `LEFT JOIN` *`B join_condition`* as follows:

- Table *`B`* is set to depend on table *`A`* and all tables on which *`A`* depends.
- Table *`A`* is set to depend on all tables (except *`B`*) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how to retrieve rows from table *`B`*. (In other words, any condition in the `WHERE` clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- All standard `WHERE` optimizations are performed.
- If there is a row in *`A`* that matches the `WHERE` clause, but there is no row in *`B`* that matches the `ON` condition, an extra *`B`* row is generated with all columns set to `NULL`.
- If you use `LEFT JOIN` to find rows that do not exist in some table and you have the following test: *`col_name`* `IS NULL` in the `WHERE` part, where *`col_name`* is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

The implementation of `RIGHT JOIN` is analogous to that of `LEFT JOIN` with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by `LEFT JOIN` or `STRAIGHT_JOIN` helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on `b` because the `LEFT JOIN` forces it to be read before `d`:

```
SELECT *
  FROM a JOIN b LEFT JOIN c ON (c.key=a.key)
  LEFT JOIN d ON (d.key=a.key)
  WHERE b.key=d.key;
```

The fix in this case is reverse the order in which `a` and `b` are listed in the `FROM` clause:

```
SELECT *
  FROM b JOIN a LEFT JOIN c ON (c.key=a.key)
  LEFT JOIN d ON (d.key=a.key)
  WHERE b.key=d.key;
```

For a `LEFT JOIN`, if the `WHERE` condition is always false for the generated `NULL` row, the `LEFT JOIN` is changed to a normal join. For example, the `WHERE` clause would be false in the following query if `t2.column1` were `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table `t2` before table `t1` if doing so would result in a better query plan. To provide a hint about the table join order, use `STRAIGHT_JOIN`. (See Section 13.2.8, “SELECT Syntax”.)

8.2.1.8 Nested-Loop Join Algorithms

MySQL executes joins between tables using a nested-loop algorithm or variations on it.

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables `t1`, `t2`, and `t3` is to be executed using the following join types:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Table   Join Type
t1      range
t2      ref
t3      ALL
```

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, it typically reads tables processed in the inner loops many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. The reduces the number of times the inner table must be read by an order of magnitude.

MySQL uses join buffering under these conditions:

- The `join_buffer_size` system variable determines the size of each join buffer.
- Join buffering can be used when the join is of type `ALL` or `index` (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or `range`.
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is never allocated for the first nonconst table, even if it would be of type `ALL` or `index`.
- A join buffer is allocated prior to executing the join and freed after the query is done.
- Only columns of interest to the join are stored in the join buffer, not whole rows.

For the example join described previously for the NLJ algorithm (without buffering), the join is done as follow using join buffering:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions,
          send to client
        }
      }
      empty buffer
    }
  }
}

if buffer is not empty {
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

If *S* is the size of each stored `t1`, `t2` combination is the join buffer and *C* is the number of combinations in the buffer, the number of times table `t3` is scanned is:

```
(S * C)/join_buffer_size + 1
```

The number of `t3` scans decreases as the value of `join_buffer_size` increases, up to the point when `join_buffer_size` is large enough to hold all previous row combinations. At that point, there is no speed to be gained by making it larger.

8.2.1.9 Nested Join Optimization

As of MySQL 5.0.1, the syntax for expressing joins permits nested joins. The following discussion refers to the join syntax described in Section 13.2.8.2, “JOIN Syntax”.

The syntax of *`table_factor`* is extended in comparison with the SQL Standard. The latter accepts only *`table_reference`*, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of *`table_reference`* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, `CROSS JOIN` is a syntactic equivalent to `INNER JOIN` (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause; `CROSS JOIN` is used otherwise.

In versions of MySQL prior to 5.0.1, parentheses in *`table_references`* were just omitted and all join operations were grouped to the left. In general, parentheses can be ignored in join expressions containing only inner join operations.

After removing parentheses and grouping operations to the left, the join expression:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
   ON t1.a=t2.a
```

transforms into the expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
    ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables `t1`, `t2`, and `t3` have the following state:

- Table `t1` contains rows `(1)`, `(2)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Table `t2` contains row `(1,101)`
- Table `t3` contains row `(101)`

In this case, the first expression returns a result set including the rows `(1,1,101,101)`, `(2,NULL,NULL,NULL)`, whereas the second expression returns the rows `(1,1,101,101)`, `(2,NULL,NULL,101)`:

```
mysql> SELECT *
    -> FROM t1
    ->      LEFT JOIN
    ->      (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
    ->      ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
    -> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
    ->      LEFT JOIN t3
    ->      ON t2.b=t3.b OR t2.b IS NULL;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
    -> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
    -> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

is equivalent to this expression:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

for any tables `t1,t2,t3` and any condition `P` over attributes `t2.b` and `t3.b`.

Whenever the order of execution of the join operations in a join expression (*`join_table`*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
  WHERE t1.a > 1

SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
  WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

The nested join is formed in the first query with a left join operation, whereas in the second query it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression will dictate the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. (In our extended syntax the parentheses in `(t2, t3)` of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because `LEFT JOIN` and `ON` would play the role of the left and right delimiters for the expression `(t2,t3)`.)

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed. You can remove parentheses and evaluate left to right (or, in fact, you can evaluate the tables in any order).
- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall by what algorithmic schema the nested-loop join executes a query. Suppose that we have a join query over 3 tables `T1,T2,T3` of the form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
                 INNER JOIN T3 ON P2(T2,T3)
  WHERE P(T1,T2,T3).
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Here, `P1(T1,T2)` and `P2(T3,T3)` are some join conditions (on expressions), whereas `P(T1,T2,T3)` is a condition over columns of tables `T1,T2,T3`.

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation `t1||t2||t3` means “a row constructed by concatenating the columns of rows `t1`, `t2`, and `t3`.” In some of the following examples, `NULL` where a row name appears means that `NULL` is used for each column of that row. For example, `t1||t2||NULL` means “a row constructed by concatenating the columns of rows `t1` and `t2`, and `NULL` for each column of `t3`.”

Now let's consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
             (T2 LEFT JOIN T3 ON P2(T2,T3))
             ON P1(T1,T2)
  WHERE P(T1,T2,T3).
```

For this query, we modify the nested-loop pattern to get:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by `NULL` values for the columns of the inner tables. The result row is passed to the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In our example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

For the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For the queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer will evaluate two different nestings:

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
  WHERE P(T1,T2,T3)
```

The nestings are these:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

and:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In both nestings, `T1` must be processed in the outer loop because it is used in an outer join. `T2` and `T3` are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, `T2` and `T3` can be processed in either order.

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our `WHERE` condition `P(T1,T2,T3)` can be represented by a conjunctive formula:

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

In this case, MySQL actually uses the following nested-loop schema for the execution of the query with inner joins:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2)  {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

You see that each of the conjuncts `C1(T1)`, `C2(T2)`, `C3(T3)` are pushed out of the most inner loop to the most outer loop where it can be evaluated. If `C1(T1)` is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table `T1` passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the `WHERE` condition is to be checked only after it has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we have to introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

For our example with outer joins with:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

the nested-loop schema using guarded pushed-down conditions looks like this:

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
      such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  }
  IF (!f1 && P(t1,NULL,NULL)) {
      t:=t1||NULL||NULL; OUTPUT t;
  }
}
```

In general, pushed-down predicates can be extracted from join conditions such as `P1(T1,T2)` and `P(T2,T3)`. In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the `NULL`-complemented row generated by the corresponding outer join operation.

Access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the `WHERE` condition. (We could use conditional key access in this case, but this technique is not employed yet in MySQL 5.0.)

8.2.1.10 Outer Join Simplification

Table expressions in the `FROM` clause of a query are simplified in many cases.

At the parser stage, queries with right outer joins operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed according to the following rule:

```
(T1, ...) RIGHT JOIN (T2,...) ON P(T1,...,T2,...) =
(T2, ...) LEFT JOIN (T1,...) ON P(T1,...,T2,...)
```

All inner join expressions of the form `T1 INNER JOIN T2 ON P(T1,T2)` are replaced by the list `T1,T2`, `P(T1,T2)` being joined as a conjunct to the `WHERE` condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for join queries with outer join operation, it takes into consideration only the plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer options are limited because only such plans enables us to execute queries with outer joins operations by the nested loop schema.

Suppose that we have a query of the form:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
  WHERE P(T1,T2) AND R(T2)
```

with `R(T2)` narrowing greatly the number of matching rows from table `T2`. If we executed the query as it is, the optimizer would have no other choice besides to access table `T1` before table `T2` that may lead to a very inefficient execution plan.

Fortunately, MySQL converts such a query into a query without an outer join operation if the `WHERE` condition is null-rejected. A condition is called null-rejected for an outer join operation if it evaluates to `FALSE` or to `UNKNOWN` for any `NULL`-complemented row built for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected:

```
T2.B IS NOT NULL,
T2.B > 3,
T2.C <= T1.C,
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
T2.B IS NULL,
T1.B < 3 OR T2.B IS NOT NULL,
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple. A condition is null-rejected in the following cases:

- If it is of the form `A IS NOT NULL`, where `A` is an attribute of any of the inner tables
- If it is a predicate containing a reference to an inner table that evaluates to `UNKNOWN` when one of its arguments is `NULL`
- If it is a conjunction containing a null-rejected condition as a conjunct
- If it is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

the `WHERE` condition is null-rejected for the second outer join operation but is not null-rejected for the first one.

If the `WHERE` condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, the preceding query is replaced with the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

For the original query, the optimizer would evaluate plans compatible with only one access order `T1,T2,T3`. For the replacing query, it additionally considers the access sequence `T3,T1,T2`.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

will be first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

Now the remaining outer join operation can be replaced by an inner join, too, because the condition `T3.B=T2.B` is null-rejected and we get a query without outer joins at all:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes we succeed in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0
```

is converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2 INNER JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0,
```

That can be rewritten only to the form still containing the embedding outer join operation:

```
SELECT * FROM T1 LEFT JOIN
              (T2,T3)
              ON (T2.A=T1.A AND T3.B=T2.B)
  WHERE T3.C > 0.
```

When trying to convert an embedded outer join operation in a query, we must take into account the join condition for the embedding outer join together with the `WHERE` condition. In the query:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A AND T3.C=T1.C
  WHERE T3.D > 0 OR T1.D > 0
```

the `WHERE` condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join `T2.A=T1.A AND T3.C=T1.C` is null-rejected. So the query can be converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2, T3)
              ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
  WHERE T3.D > 0 OR T1.D > 0
```

The algorithm that converts outer join operations into inner joins was implemented in full measure, as it has been described here, in MySQL 5.0.1. MySQL 4.1 performs only some simple conversions.

8.2.1.11 ORDER BY Optimization

In some cases, MySQL can use an index to satisfy an `ORDER BY` clause without doing extra sorting.

The index can also be used even if the `ORDER BY` does not match the index exactly, as long as all unused portions of the index and all extra `ORDER BY` columns are constants in the `WHERE` clause. The following queries use the index to resolve the `ORDER BY` part:

```
SELECT * FROM t1
  ORDER BY key_part1,key_part2,... ;

SELECT * FROM t1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  WHERE key_part1 = constant
  ORDER BY key_part2;

SELECT * FROM t1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
  WHERE key_part1 = 1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
  WHERE key_part1 > constant
  ORDER BY key_part1 ASC;

SELECT * FROM t1
  WHERE key_part1 < constant
  ORDER BY key_part1 DESC;

SELECT * FROM t1
  WHERE key_part1 = constant1 AND key_part2 > constant2
  ORDER BY key_part2;
```

In some cases, MySQL *cannot* use indexes to resolve the `ORDER BY`, although it still uses indexes to find the rows that match the `WHERE` clause. These cases include the following:

- The query uses `ORDER BY` on different indexes:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- The query uses `ORDER BY` on nonconsecutive parts of an index:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
  ```

- The query mixes `ASC` and `DESC`:

  ```
  SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
  ```

- The index used to fetch the rows differs from the one used in the `ORDER BY`:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- The query uses `ORDER BY` with an expression that includes terms other than the index column name:

  ```
  SELECT * FROM t1 ORDER BY ABS(key);
  SELECT * FROM t1 ORDER BY -key;
  ```

- The query joins many tables, and the columns in the `ORDER BY` are not all from the first nonconstant table that is used to retrieve rows. (This is the first table in the `EXPLAIN` output that does not have a `const` join type.)

- The query has different `ORDER BY` and `GROUP BY` expressions.

- There is an index on only a prefix of a column named in the `ORDER BY` clause. In this case, the index cannot be used to fully resolve the sort order. For example, if only the first 10 bytes of a `CHAR(20)` column are indexed, the index cannot distinguish values past the 10th byte and a `filesort` will be needed.

- The index does not store rows in order. For example, this is true for a `HASH` index in a `MEMORY` table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Availability of an index for sorting may be affected by the use of column aliases. Suppose that the column `t1.a` is indexed. In this statement, the name of the column in the select list is `a`. It refers to `t1.a`, so for the reference to `a` in the `ORDER BY`, the index can be used:

```
SELECT a FROM t1 ORDER BY a;
```

In this statement, the name of the column in the select list is also `a`, but it is the alias name. It refers to `ABS(a)`, so for the reference to `a` in the `ORDER BY`, the index cannot be used:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

In the following statement, the `ORDER BY` refers to a name that is not the name of a column in the select list. But there is a column in `t1` named `a`, so the `ORDER BY` uses that and the index can be used. (The resulting sort order may be completely different from the order for `ABS(a)`, of course.)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

By default, MySQL sorts all `GROUP BY col1, col2, ...` queries as if you specified `ORDER BY col1, col2, ...` in the query as well. If you include an explicit `ORDER BY` clause that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs.

If a query includes `GROUP BY` but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying `ORDER BY NULL`. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

The optimizer may still choose to use sorting to implement grouping operations. `ORDER BY NULL` suppresses sorting of the result, not prior sorting done by grouping operations to determine the result.

With `EXPLAIN SELECT ... ORDER BY`, you can check whether MySQL can use indexes to resolve the query. It cannot if you see `Using filesort` in the `Extra` column. See Section 8.8.1, “Optimizing Queries with EXPLAIN”. Filesort uses a fixed-length row-storage format similar to that used by the `MEMORY` storage engine. Variable-length types such as `VARCHAR` are stored using a fixed length.

MySQL has two `filesort` algorithms for sorting and retrieving results. The original method uses only the `ORDER BY` columns. The modified method uses not just the `ORDER BY` columns, but all the columns referenced by the query.

The optimizer selects which `filesort` algorithm to use. It normally uses the modified algorithm except when `BLOB` or `TEXT` columns are involved, in which case it uses the original algorithm. For both algorithms, the sort buffer size is the `sort_buffer_size` system variable value.

The original `filesort` algorithm works as follows:

1. Read all rows according to key or by table scanning. Skip rows that do not match the `WHERE` clause.
2. For each row, store in the sort buffer a tuple consisting of a pair of values (the sort key value and the row ID).
3. If all pairs fit into the sort buffer, no temporary file is created. Otherwise, when the sort buffer becomes full, run a qsort (quicksort) on it in memory and write it to a temporary file. Save a pointer to the sorted block.
4. Repeat the preceding steps until all rows have been read.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

5. Do a multi-merge of up to MERGEBUFF (7) regions to one block in another temporary file. Repeat until all blocks from the first file are in the second file.

6. Repeat the following until there are fewer than MERGEBUFF2 (15) blocks left.

7. On the last multi-merge, only the row ID (the last part of the value pair) is written to a result file.

8. Read the rows in sorted order using the row IDs in the result file. To optimize this, read in a large block of row IDs, sort them, and use them to read the rows in sorted order into a row buffer. The row buffer size is the read_rnd_buffer_size system variable value. The code for this step is in the sql/records.cc source file.

One problem with this approach is that it reads rows twice: One time during WHERE clause evaluation, and again after sorting the value pairs. And even if the rows were accessed successively the first time (for example, if a table scan is done), the second time they are accessed randomly. (The sort keys are ordered, but the row positions are not.)

The modified filesort algorithm incorporates an optimization to avoid reading the rows twice: It records the sort key value, but instead of the row ID, it records the columns referenced by the query. The modified filesort algorithm works like this:

1. Read the rows that match the WHERE clause.

2. For each row, store in the sort buffer a tuple consisting of the sort key value and the columns referenced by the query.

3. When the sort buffer becomes full, sort the tuples by sort key value in memory and write it to a temporary file.

4. After merge-sorting the temporary file, retrieve the rows in sorted order, but read the columns required by the query directly from the sorted tuples rather than by accessing the table a second time.

The tuples used by the modified filesort algorithm are longer than the pairs used by the original algorithm, and fewer of them fit in the sort buffer. As a result, it is possible for the extra I/O to make the modified approach slower, not faster. To avoid a slowdown, the optimizer uses the modified algorithm only if the total size of the extra columns in the sort tuple does not exceed the value of the max_length_for_sort_data system variable. (A symptom of setting the value of this variable too high is a combination of high disk activity and low CPU activity.)

If a filesort is done, EXPLAIN output includes Using filesort in the Extra column.

Suppose that a table t1 has four VARCHAR columns a, b, c, and d and that the optimizer uses filesort for this query:

```
SELECT * FROM t1 ORDER BY a, b;
```

The query sorts by a and b, but returns all columns, so the columns referenced by the query are a, b, c, and d. Depending on which filesort algorithm the optimizer chooses, the query executes as follows:

For the original algorithm, sort buffer tuples have these contents:

```
(fixed size a value, fixed size b value,
row ID into t1)
```

The optimizer sorts on the fixed size values. After sorting, the optimizer reads the tuples in order and uses the row ID in each tuple to read rows from t1 to obtain the select list column values.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For the modified algorithm, sort buffer tuples have these contents:

```
(fixed size a value, fixed size b value,
a value, b value, c value, d value)
```

The optimizer sorts on the fixed size values. After sorting, the optimizer reads the tuples in order and uses the values for `a`, `b`, `c`, and `d` to obtain the select list column values without reading `t1` again.

For slow queries for which `filesort` is not used, try lowering `max_length_for_sort_data` to a value that is appropriate to trigger a `filesort`.

To increase `ORDER BY` speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the `sort_buffer_size` variable value.
- Increase the `read_rnd_buffer_size` variable value.
- Use less RAM per row by declaring columns only as large as they need to be to hold the values stored in them. For example, `CHAR(16)` is better than `CHAR(200)` if values never exceed 16 characters.
- Change the `tmpdir` system variable to point to a dedicated file system with large amounts of free space. The variable value can list several paths that are used in round-robin fashion; you can use this feature to spread the load across several directories. Paths should be separated by colon characters (“`:`”) on Unix and semicolon characters (“`;`”) on Windows, NetWare, and OS/2. The paths should name directories in file systems located on different *physical* disks, not different partitions on the same disk.

8.2.1.12 GROUP BY Optimization

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and that the index stores its keys in order (for example, this is a `BTREE` index and not a `HASH` index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a `GROUP BY` query through index access, as detailed in the following sections. In the first method, the grouping operation is applied together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

In MySQL, `GROUP BY` is used for sorting, so the server may also apply `ORDER BY` optimizations to grouping. See Section 8.2.1.11, “ORDER BY Optimization”.

Loose Index Scan

The most efficient way to process `GROUP BY` is when an index is used to directly retrieve the grouping columns. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, `BTREE`). This property enables use of lookup groups in an index without having to consider all keys in the index that satisfy all `WHERE` conditions. This access method considers only a fraction of the keys in an index, so it is called a *loose index scan*. When there is no `WHERE` clause, a loose index scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the `WHERE` clause contains range predicates (see the discussion of the `range` join type in Section 8.8.1, “Optimizing Queries with EXPLAIN”), a loose index scan looks up the first key of each group

This documentation is for an older version. If you're
This documentation is for an older version. If you're

that satisfies the range conditions, and again reads the least possible number of keys. This is possible under the following conditions:

- The query is over a single table.
- The `GROUP BY` names only columns that form a leftmost prefix of the index and no other columns. (If, instead of `GROUP BY`, the query has a `DISTINCT` clause, all distinct attributes refer to columns that form a leftmost prefix of the index.) For example, if a table `t1` has an index on `(c1,c2,c3)`, loose index scan is applicable if the query has `GROUP BY c1, c2,`. It is not applicable if the query has `GROUP BY c2, c3` (the columns are not a leftmost prefix) or `GROUP BY c1, c2, c4` (`c4` is not in the index).
- The only aggregate functions used in the select list (if any) are `MIN()` and `MAX()`, and all of them refer to the same column. The column must be in the index and must follow the columns in the `GROUP BY`.
- Any other parts of the index than those from the `GROUP BY` referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of `MIN()` or `MAX()` functions.
- For columns in the index, full column values must be indexed, not just a prefix. For example, with `c1 VARCHAR(20), INDEX (c1(10))`, the index cannot be used for loose index scan.

If loose index scan is applicable to a query, the `EXPLAIN` output shows `Using index for group-by` in the `Extra` column.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The loose index scan access method can be used for the following queries:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than `MIN()` or `MAX()`:

  ```
  SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
  ```

- The columns in the `GROUP BY` clause do not form a leftmost prefix of the index:

  ```
  SELECT c1, c2 FROM t1 GROUP BY c2, c3;
  ```

- The query refers to a part of a key that comes after the `GROUP BY` part, and for which there is no equality with a constant:

  ```
  SELECT c1, c3 FROM t1 GROUP BY c1, c2;
  ```

  Were the query to include `WHERE c3 = const`, loose index scan could be used.

Tight Index Scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When the conditions for a loose index scan are not met, it still may be possible to avoid creation of temporary tables for `GROUP BY` queries. If there are range conditions in the `WHERE` clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the `WHERE` clause, or scans the whole index if there are no range conditions, we term it a *tight index scan*. With a tight index scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there is a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the `GROUP BY` key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If we require sorting of the `GROUP BY` result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The following queries do not work with the loose index scan access method described earlier, but still work with the tight index scan access method.

- There is a gap in the `GROUP BY`, but it is covered by the condition `c2 = 'a'`:

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- The `GROUP BY` does not begin with the first part of the key, but there is a condition that provides a constant for that part:

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

8.2.1.13 DISTINCT Optimization

`DISTINCT` combined with `ORDER BY` needs a temporary table in many cases.

Because `DISTINCT` may use `GROUP BY`, learn how MySQL works with columns in `ORDER BY` or `HAVING` clauses that are not part of the selected columns. See Section 12.16.3, “MySQL Handling of GROUP BY”.

In most cases, a `DISTINCT` clause can be considered as a special case of `GROUP BY`. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;

SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to `GROUP BY` queries can be also applied to queries with a `DISTINCT` clause. Thus, for more details on the optimization possibilities for `DISTINCT` queries, see Section 8.2.1.12, “GROUP BY Optimization”.

When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds `row_count` unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you can check with `EXPLAIN`), MySQL stops reading from `t2` (for any particular row in `t1`) when it finds the first row in `t2`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

8.2.1.14 Optimizing Subqueries with EXISTS Strategy

Certain optimizations are applicable to comparisons that use the `IN` operator to test subquery results (or that use `=ANY`, which is equivalent). This section discusses these optimizations, particularly with regard to the challenges that `NULL` values present. The last part of the discussion includes suggestions on what you can do to help the optimizer.

Consider the following subquery comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL evaluates queries “from outside to inside.” That is, it first obtains the value of the outer expression *`outer_expr`*, and then runs the subquery and captures the rows that it produces.

A very useful optimization is to “inform” the subquery that the only rows of interest are those where the inner expression *`inner_expr`* is equal to *`outer_expr`*. This is done by pushing down an appropriate equality into the subquery's `WHERE` clause. That is, the comparison is converted to this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

After the conversion, MySQL can use the pushed-down equality to limit the number of rows that it must examine when evaluating the subquery.

More generally, a comparison of *`N`* values to a subquery that returns *`N`*-value rows is subject to the same conversion. If *`oe_i`* and *`ie_i`* represent corresponding outer and inner expression values, this subquery comparison:

```
(oe_1, ..., oe_N) IN
  (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

Becomes:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND oe_1 = ie_1
                          AND ...
                          AND oe_N = ie_N)
```

The following discussion assumes a single pair of outer and inner expression values for simplicity.

The conversion just described has its limitations. It is valid only if we ignore possible `NULL` values. That is, the “pushdown” strategy works as long as both of these two conditions are true:

- *`outer_expr`* and *`inner_expr`* cannot be `NULL`.
- You do not need to distinguish `NULL` from `FALSE` subquery results. (If the subquery is a part of an `OR` or `AND` expression in the `WHERE` clause, MySQL assumes that you do not care.)

When either or both of those conditions do not hold, optimization is more complex.

Suppose that *`outer_expr`* is known to be a non-`NULL` value but the subquery does not produce a row such that *`outer_expr`* = *`inner_expr`*. Then *`outer_expr`* `IN (SELECT ...)` evaluates as follows:

- `NULL`, if the `SELECT` produces any row where *`inner_expr`* is `NULL`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `FALSE`, if the `SELECT` produces only non-`NULL` values or produces nothing

In this situation, the approach of looking for rows with *`outer_expr = inner_expr`* is no longer valid. It is necessary to look for such rows, but if none are found, also look for rows where *`inner_expr`* is `NULL`. Roughly speaking, the subquery can be converted to:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND
        (outer_expr=inner_expr OR inner_expr IS NULL))
```

The need to evaluate the extra `IS NULL` condition is why MySQL has the `ref_or_null` access method:

```
mysql> EXPLAIN
    -> SELECT outer_expr IN (SELECT t2.maybe_null_key
    ->                       FROM t2, t3 WHERE ...)
    -> FROM t1;
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: ref_or_null
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Using index
...
```

The `unique_subquery` and `index_subquery` subquery-specific access methods also have “or `NULL`” variants. However, they are not visible in `EXPLAIN` output, so you must use `EXPLAIN EXTENDED` followed by `SHOW WARNINGS` (note the `checking NULL` in the warning message):

```
mysql> EXPLAIN EXTENDED
    -> SELECT outer_expr IN (SELECT maybe_null_key FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using index

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select (`test`.`t1`.`outer_expr`,
         (((`test`.`t1`.`outer_expr`) in t2 on
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        maybe_null_key checking NULL))) AS `outer_expr IN (SELECT
        maybe_null_key FROM t2)` from `test`.`t1`
```

The additional `OR ... IS NULL` condition makes query execution slightly more complicated (and some optimizations within the subquery become inapplicable), but generally this is tolerable.

The situation is much worse when *`outer_expr`* can be `NULL`. According to the SQL interpretation of `NULL` as "unknown value," `NULL IN (SELECT inner_expr ...)` should evaluate to:

- `NULL`, if the `SELECT` produces any rows
- `FALSE`, if the `SELECT` produces no rows

For proper evaluation, it is necessary to be able to check whether the `SELECT` has produced any rows at all, so *`outer_expr = inner_expr`* cannot be pushed down into the subquery. This is a problem, because many real world subqueries become very slow unless the equality can be pushed down.

Essentially, there must be different ways to execute the subquery depending on the value of *`outer_expr`*. In MySQL 5.0 before 5.0.36, the optimizer chose speed over distinguishing a `NULL` from `FALSE` result, so for some queries, you might get a `FALSE` result rather than `NULL`.

As of MySQL 5.0.36, the optimizer chooses SQL compliance over speed, so it accounts for the possibility that *`outer_expr`* might be `NULL`.

If *`outer_expr`* is `NULL`, to evaluate the following expression, it is necessary to run the `SELECT` to determine whether it produces any rows:

```
NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

It is necessary to run the original `SELECT` here, without any pushed-down equalities of the kind mentioned earlier.

On the other hand, when *`outer_expr`* is not `NULL`, it is absolutely essential that this comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

be converted to this expression that uses a pushed-down condition:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

Without this conversion, subqueries will be slow. To solve the dilemma of whether to push down or not push down conditions into the subquery, the conditions are wrapped in "trigger" functions. Thus, an expression of the following form:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

is converted into:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND trigcond(outer_expr=inner_expr))
```

More generally, if the subquery comparison is based on several pairs of outer and inner expressions, the conversion takes this comparison:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

and converts it to this expression:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                           AND trigcond(oe_1=ie_1)
                           AND ...
                           AND trigcond(oe_N=ie_N)
       )
```

Each `trigcond(X)` is a special function that evaluates to the following values:

- *X* when the “linked” outer expression *oe_i* is not `NULL`
- `TRUE` when the “linked” outer expression *oe_i* is `NULL`

Note

Trigger functions are *not* triggers of the kind that you create with `CREATE TRIGGER`.

Equalities that are wrapped into `trigcond()` functions are not first class predicates for the query optimizer. Most optimizations cannot deal with predicates that may be turned on and off at query execution time, so they assume any `trigcond(X)` to be an unknown function and ignore it. At the moment, triggered equalities can be used by those optimizations:

- Reference optimizations: `trigcond(X=Y [OR Y IS NULL])` can be used to construct `ref`, `eq_ref`, or `ref_or_null` table accesses.
- Index lookup-based subquery execution engines: `trigcond(X=Y)` can be used to construct `unique_subquery` or `index_subquery` accesses.
- Table-condition generator: If the subquery is a join of several tables, the triggered condition will be checked as soon as possible.

When the optimizer uses a triggered condition to create some kind of index lookup-based access (as for the first two items of the preceding list), it must have a fallback strategy for the case when the condition is turned off. This fallback strategy is always the same: Do a full table scan. In `EXPLAIN` output, the fallback shows up as `Full scan on NULL key` in the `Extra` column:

```
mysql> EXPLAIN SELECT t1.col1,
    -> t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
        ...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: key1
          key: key1
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Full scan on NULL key
```

If you run `EXPLAIN EXTENDED` followed by `SHOW WARNINGS`, you can see the triggered condition:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select `test`.`t1`.`col1` AS `col1`,
         <in_optimizer>(`test`.`t1`.`col1`,
         <exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
         on key1 checking NULL
         where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
         trigcond(<is_not_null_test>(`test`.`t2`.`key1`))))) AS
         `t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
         from `test`.`t1`
```

The use of triggered conditions has some performance implications. A `NULL IN (SELECT ...)` expression now may cause a full table scan (which is slow) when it previously did not. This is the price paid for correct results (the goal of the trigger-condition strategy was to improve compliance and not speed).

For multiple-table subqueries, execution of `NULL IN (SELECT ...)` will be particularly slow because the join optimizer does not optimize for the case where the outer expression is `NULL`. It assumes that subquery evaluations with `NULL` on the left side are very rare, even if there are statistics that indicate otherwise. On the other hand, if the outer expression might be `NULL` but never actually is, there is no performance penalty.

To help the query optimizer better execute your queries, use these tips:

- Declare a column as `NOT NULL` if it really is. (This also helps other aspects of the optimizer by simplifying condition testing for the column.)
- If you do not need to distinguish a `NULL` from `FALSE` subquery result, you can easily avoid the slow execution path. Replace a comparison that looks like this:

  ```
  outer_expr IN (SELECT inner_expr FROM ...)
  ```

  with this expression:

  ```
  (outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
  ```

  Then `NULL IN (SELECT ...)` will never be evaluated because MySQL stops evaluating `AND` parts as soon as the expression result is clear.

8.2.1.15 LIMIT Query Optimization

In some cases, MySQL handles a query differently when you are using `LIMIT row_count` and not using `HAVING`:

- If you select only a few rows with `LIMIT`, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you combine `LIMIT row_count` with `ORDER BY`, MySQL ends the sorting as soon as it has found the first `row_count` rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the `LIMIT` clause must be selected, and most or all of them must be sorted, before it can be ascertained that the first `row_count` rows have been found. In either case, after the initial rows have been found, there is no need to sort any remainder of the result set, and MySQL does not do so.
- If you combine `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds `row_count` unique rows.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- In some cases, a `GROUP BY` can be resolved by reading the index in order (or doing a sort on the index) and then calculating summaries until the index value changes. In this case, `LIMIT row_count` does not calculate any unnecessary `GROUP BY` values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using `SQL_CALC_FOUND_ROWS`. The number of rows can then be retrieved with `SELECT FOUND_ROWS()`. See Section 12.13, “Information Functions”.
- `LIMIT 0` quickly returns an empty set. This can be useful for checking the validity of a query. When using one of the MySQL APIs, it can also be employed for obtaining the types of the result columns. This technique does not work with the `mysql` client program, which merely displays `Empty set` in such cases. Instead, use `SHOW COLUMNS` or `DESCRIBE` for this purpose.
- When the server uses temporary tables to resolve the query, it uses the `LIMIT row_count` clause to calculate how much space is required.

8.2.1.16 Row Constructor Expression Optimization

Row constructors permit simultaneous comparisons of multiple values. For example, these two statements are semantically equivalent:

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

In addition, the optimizer handles both expressions the same way.

The optimizer is less likely to use available indexes if the row constructor columns do not cover the prefix of an index. Consider the following table, which has a primary key on `(c1, c2, c3)`:

```
CREATE TABLE t1 (
  c1 INT, c2 INT, c3 INT, c4 CHAR(100),
  PRIMARY KEY(c1,c2,c3)
);
```

In this query, the `WHERE` clause uses all columns in the index. However, the row constructor itself does not cover an index prefix, with the result that the optimizer uses only `c1` (`key_len=4`, the size of `c1`):

```
mysql> EXPLAIN SELECT * FROM t1
    -> WHERE c1=1 AND (c2,c3) > (1,1)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 3
        Extra: Using where
```

In such cases, rewriting the row constructor expression using an equivalent nonconstructor expression may result in more complete index use. For the given query, the row constructor and equivalent nonconstructor expressions are:

```
(c2,c3) > (1,1)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
c2 > 1 OR ((c2 = 1) AND (c3 > 1))
```

Rewriting the query to use the nonconstructor expression results in the optimizer using all three columns in the index (`key_len=12`):

```
mysql> EXPLAIN SELECT * FROM t1
    -> WHERE c1 = 1 AND (c2 > 1 OR ((c2 = 1) AND (c3 > 1)))\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 12
          ref: NULL
         rows: 3
        Extra: Using where
```

Thus, for better results, avoid mixing row constructors with `AND`/`OR` expressions. Use one or the other.

8.2.1.17 How to Avoid Full Table Scans

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.
- There are no usable restrictions in the `ON` or `WHERE` clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”.
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
    WHERE t1.col_name=t2.col_name;
  ```

  See Section 8.9.2, “Index Hints”.
- Start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.4, “Server System Variables”.

8.2.2 Optimizing DML Statements

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This section explains how to speed up the data manipulation language (DML) statements, `INSERT`, `UPDATE`, and `DELETE`. Traditional OLTP applications and modern web applications typically do many small DML operations, where concurrency is vital. Data analysis and reporting applications typically run DML operations that affect many rows at once, where the main considerations is the I/O to write large amounts of data and keep indexes up-to-date. For inserting and updating large volumes of data (known in the industry as ETL, for “extract-transform-load”), sometimes you use other SQL statements or external commands, that mimic the effects of `INSERT`, `UPDATE`, and `DELETE` statements.

8.2.2.1 Speed of INSERT Statements

To optimize insert speed, combine many small operations into a single large operation. Ideally, you make a single connection, send the data for many new rows at once, and delay all index updates and consistency checking until the very end.

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by log *`N`*, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use `INSERT` statements with multiple `VALUES` lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row `INSERT` statements. If you are adding data to a nonempty table, you can tune the `bulk_insert_buffer_size` variable to make data insertion even faster. See Section 5.1.4, “Server System Variables”.
- When loading a table from a text file, use `LOAD DATA INFILE`. This is usually 20 times faster than using `INSERT` statements. See Section 13.2.6, “LOAD DATA INFILE Syntax”.
- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- See Section 8.5.2, “Bulk Data Loading for MyISAM Tables” for tips specific to `MyISAM` tables.

8.2.2.2 Speed of UPDATE Statements

An update statement is optimized like a `SELECT` query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a `MyISAM` table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use `OPTIMIZE TABLE` occasionally. See Section 13.7.2.5, “OPTIMIZE TABLE Syntax”.

8.2.2.3 Speed of DELETE Statements

The time required to delete individual rows is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the `key_buffer_size` system variable. See Section 8.12.2, “Tuning Server Parameters”.

To delete all rows from a table, `TRUNCATE TABLE tbl_name` is faster than `DELETE FROM tbl_name`. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See Section 13.1.21, “TRUNCATE TABLE Syntax”.

8.2.3 Optimizing Database Privileges

The more complex your privilege setup, the more overhead applies to all SQL statements. Simplifying the privileges established by `GRANT` statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, consider using a simplified grant structure to reduce permission-checking overhead.

8.2.4 Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- If your application makes several database requests to perform related updates, combining the statements into a stored routine can help performance. Similarly, if your application computes a single result based on several column values or large volumes of data, combining the computation into a UDF (user-defined function) can help performance. The resulting fast database operations are then available to be reused by other queries, applications, and even code written in different programming languages. See Section 18.2, “Using Stored Routines (Procedures and Functions)” and Section 21.2, “Adding New Functions to MySQL” for more information.
- To fix any compression issues that occur with `ARCHIVE` tables, use `OPTIMIZE TABLE`. See Section 14.8, “The ARCHIVE Storage Engine”.
- If possible, classify reports as “live” or as “statistical”, where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.
- If you have data that does not conform well to a rows-and-columns table structure, you can pack and store data into a `BLOB` column. In this case, you must provide code in your application to pack and unpack information, but this might save I/O operations to read and write the sets of related values.
- With Web servers, store images and other binary assets as files, with the path name stored in the database rather than the file itself. Most Web servers are better at caching files than database contents, so using files is generally faster. (Although you must handle backups and storage issues yourself in this case.)
- If you need really high speed, look at the low-level MySQL interfaces. For example, by accessing the MySQL `InnoDB` or `MyISAM` storage engine directly, you could get a substantial speed increase compared to using the SQL interface.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See Chapter 16, *Replication*.

8.3 Optimization and Indexes

The best way to improve the performance of `SELECT` operations is to create indexes on one or more of the columns that are tested in the query. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the `WHERE` clause, and retrieve the other column values for those rows. All MySQL data types can be indexed.

Although it can be tempting to create an indexes for every possible column used in a query, unnecessary indexes waste space and waste time for MySQL to determine which indexes to use. Indexes also add to the cost of inserts, updates, and deletes because each index must be updated. You must find the right balance to achieve fast queries using the optimal set of indexes.

8.3.1 How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

Most MySQL indexes (`PRIMARY KEY`, `UNIQUE`, `INDEX`, and `FULLTEXT`) are stored in B-trees. Exceptions are that indexes on spatial data types use R-trees, and that `MEMORY` tables also support hash indexes.

Strings are automatically prefix- and end-space compressed. See Section 13.1.8, “CREATE INDEX Syntax”.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in `MEMORY` tables) are described in Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows.
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.5, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

  For comparisons between nonbinary string columns, both columns should use the same character set. For example, comparing a `utf8` column with a `latin1` column precludes use of an index.

  Comparison of dissimilar columns (comparing a string column to a temporal or numeric column, for example) may prevent use of indexes if values cannot be compared directly without conversion. For a

This documentation is for an older version. If you're
This documentation is for an older version. If you're

given value such as `1` in the numeric column, it might compare equal to any number of values in the string column such as `'1'`, `' 1'`, `'00001'`, or `'01.e1'`. This rules out use of any indexes for the string column.

- To find the `MIN()` or `MAX()` value for a specific indexed column *`key_col`*. This is optimized by a preprocessor that checks whether you are using `WHERE key_part_N = constant` on all key parts that occur before *`key_col`* in the index. In this case, MySQL does a single key lookup for each `MIN()` or `MAX()` expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable index (for example, `ORDER BY key_part1, key_part2`). If all key parts are followed by `DESC`, the key is read in reverse order. See Section 8.2.1.11, “ORDER BY Optimization”, and Section 8.2.1.12, “GROUP BY Optimization”.

- In some cases, a query can be optimized to retrieve values without consulting the data rows. If a query uses from a table only columns that are included in some index, the selected values can be be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
  WHERE key_part1=1
```

Indexes are less important for queries on small tables, or big tables where report queries process most or all of the rows. When a query needs to access most of the rows, reading sequentially is faster than working through an index. Sequential reads minimize disk seeks, even if not all the rows are needed for the query. See Section 8.2.1.17, “How to Avoid Full Table Scans” for details.

8.3.2 Using Primary Keys

The primary key for a table represents the column or set of columns that you use in your most vital queries. It has an associated index, for fast query performance. Query performance benefits from the `NOT NULL` optimization, because it cannot include any `NULL` values. With the `InnoDB` storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

If your table is big and important, but does not have an obvious column or set of columns to use as a primary key, you might create a separate column with auto-increment values to use as the primary key. These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

8.3.3 Using Foreign Keys

If a table has many columns, and you query many different combinations of columns, it might be efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table. That way, each small table can have a primary key for fast lookups of its data, and you can query just the set of columns that you need using a join operation. Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk. (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

8.3.4 Column Indexes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

All MySQL data types can be indexed. Use of indexes on the relevant columns is the best way to improve the performance of `SELECT` operations.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See Chapter 14, *Storage Engines*. All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

For additional information about column indexes, see Section 13.1.8, “CREATE INDEX Syntax”.

Prefix Indexes

With *`col_name(N)`* syntax in an index specification for a string column, you can create an index that uses only the first *`N`* characters of the column. Indexing only a prefix of column values in this way can make the index file much smaller. When you index a `BLOB` or `TEXT` column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for `InnoDB` tables).

Note

Prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements is interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

For additional information about index prefixes, see Section 13.1.8, “CREATE INDEX Syntax”.

FULLTEXT Indexes

`FULLTEXT` indexes are used for full-text searches. Only the `MyISAM` storage engine supports `FULLTEXT` indexes and only for `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see Section 12.9, “Full-Text Search Functions”.

Spatial Indexes

You can create indexes on spatial data types. Only `MyISAM` supports R-tree indexes on spatial types. As of MySQL 5.0.16, other storage engines use B-trees for indexing spatial types (except for `ARCHIVE` and `NDB`, which do not support spatial type indexing).

Indexes in the MEMORY Storage Engine

The `MEMORY` storage engine uses `HASH` indexes by default, but also supports `BTREE` indexes.

8.3.5 Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 16 columns. For certain data types, you can index a prefix of the column (see Section 8.3.4, “Column Indexes”).

MySQL can use multiple-column indexes for queries that test all the columns in the index, or queries that test just the first column, the first two columns, the first three columns, and so on. If you specify the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

columns in the right order in the index definition, a single composite index can speed up several kinds of queries on the same table.

A multiple-column index can be considered a sorted array, the rows of which contain values that are created by concatenating the values of the indexed columns.

Note

As an alternative to a composite index, you can introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it might be faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
  WHERE hash_col=MD5(CONCAT(val1,val2))
  AND col1=val1 AND col2=val2;
```

Suppose that a table has the following specification:

```
CREATE TABLE test (
    id         INT NOT NULL,
    last_name  CHAR(30) NOT NULL,
    first_name CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name)
);
```

The `name` index is an index over the `last_name` and `first_name` columns. The index can be used for lookups in queries that specify values in a known range for combinations of `last_name` and `first_name` values. It can also be used for queries that specify just a `last_name` value because that column is a leftmost prefix of the index (as described later in this section). Therefore, the `name` index is used for lookups in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';

SELECT * FROM test
  WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius'
  AND (first_name='Michael' OR first_name='Monty');

SELECT * FROM test
  WHERE last_name='Widenius'
  AND first_name >='M' AND first_name < 'N';
```

However, the `name` index is *not* used for lookups in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

Suppose that you issue the following `SELECT` statement:

```
SELECT * FROM tbl_name
  WHERE col1=val1 AND col2=val2;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If a multiple-column index exists on `col1` and `col2`, the appropriate rows can be fetched directly. If separate single-column indexes exist on `col1` and `col2`, the optimizer attempts to use the Index Merge optimization (see Section 8.2.1.4, “Index Merge Optimization”), or attempts to find the most restrictive index by deciding which index excludes more rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`.

MySQL cannot use the index to perform lookups if the columns do not form a leftmost prefix of the index. Suppose that you have the `SELECT` statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first two queries use the index. The third and fourth queries do involve indexed columns, but `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

8.3.6 Verifying Index Usage

Always check whether all your queries really use the indexes that you have created in the tables. Use the `EXPLAIN` statement, as described in Section 8.8.1, “Optimizing Queries with EXPLAIN”.

8.3.7 MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how may rows must be read for each `ref` access
- To estimate how many row a partial join will produce; that is, the number of rows that an operation of this form will produce:

  ```
  (...) JOIN tbl_name ON tbl_name.key = expr
  ```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on *`N/S`*, where *`N`* is the number of rows in the table and *`S`* is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as *`N`* `<=>` *`N`* for any other *`N`*.

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: *`expr1`* `=` *`expr2`* is not true when *`expr1`* or *`expr2`* (or both) are `NULL`. This affects `ref` accesses for comparisons of the form

This documentation is for an older version. If you're
This documentation is for an older version. If you're

tbl_name.key = expr: MySQL will not access the table if the current value of *expr* is NULL, because the comparison cannot be true.

For = comparisons, it does not matter how many NULL values are in the table. For optimization purposes, the relevant value is the average size of the non-NULL value groups. However, MySQL does not currently enable that average size to be collected or used.

For MyISAM tables, you have some control over collection of table statistics by means of the myisam_stats_method system variable. This variable has three possible values, which differ as follows:

- When myisam_stats_method is nulls_equal, all NULL values are treated as identical (that is, they all form a single value group).

  If the NULL value group size is much higher than the average non-NULL value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-NULL values. Consequently, the nulls_equal method may cause the optimizer not to use the index for ref accesses when it should.

- When myisam_stats_method is nulls_unequal, NULL values are not considered the same. Instead, each NULL value forms a separate value group of size 1.

  If you have many NULL values, this method skews the average value group size downward. If the average non-NULL value group size is large, counting NULL values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-NULL values. Consequently, the nulls_unequal method may cause the optimizer to use this index for ref lookups when other methods may be better.

- When myisam_stats_method is nulls_ignored, NULL values are ignored.

If you tend to use many joins that use <=> rather than =, NULL values are not special in comparisons and one NULL is equal to another. In this case, nulls_equal is the appropriate statistics method.

The myisam_stats_method system variable has global and session values. Setting the global value affects MyISAM statistics collection for all MyISAM tables. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of myisam_stats_method.

To regenerate MyISAM table statistics, you can use any of the following methods:

- Execute myisamchk --stats_method=*method_name* --analyze
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set myisam_stats_method and issue an ANALYZE TABLE statement

Some caveats regarding the use of myisam_stats_method:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some ALTER TABLE statements, for example.) If this happens, the statistics are collected using whatever value myisam_stats_method has at the time. Thus, if you collect statistics using one method, but myisam_stats_method is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given MyISAM table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `myisam_stats_method` applies only to `MyISAM` tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the `nulls_equal` method.

8.3.8 Comparison of B-Tree and Hash Indexes

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the `MEMORY` storage engine that lets you choose B-tree or hash indexes.

B-Tree Index Characteristics

A B-tree index can be used for column comparisons in expressions that use the `=`, `>`, `>=`, `<`, `<=`, or `BETWEEN` operators. The index also can be used for `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character. For example, the following `SELECT` statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with `'Patrick' <= key_col < 'Patricl'` are considered. In the second statement, only rows with `'Pat' <= key_col < 'Pau'` are considered.

The following `SELECT` statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the `LIKE` value begins with a wildcard character. In the second statement, the `LIKE` value is not a constant.

If you use `... LIKE '%string%'` and *`string`* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using `col_name IS NULL` employs indexes if `col_name` is indexed.

Any index that does not span all `AND` levels in the `WHERE` clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every `AND` group.

The following `WHERE` clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3

    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2

    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5

    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These `WHERE` clauses do *not* use indexes:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

    /*  Index is not used in both parts of the WHERE clause  */
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
... WHERE index=1 OR A=10

    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses `LIMIT` to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash Index Characteristics

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the `=` or `<=>` operators (but are *very* fast). They are not used for comparison operators such as `<` that find a range of values. Systems that rely on this type of single-value lookup are known as “key-value stores”; to use MySQL for such applications, use hash indexes wherever possible.
- The optimizer cannot use a hash index to speed up `ORDER BY` operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a `MyISAM` or `InnoDB` table to a hash-indexed `MEMORY` table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

8.4 Optimizing Database Structure

In your role as a database designer, look for the most efficient way to organize your schemas, tables, and columns. As when tuning application code, you minimize I/O, keep related items together, and plan ahead so that performance stays high as the data volume increases. Starting with an efficient database design makes it easier for team members to write high-performing application code, and makes the database likely to endure as applications evolve and are rewritten.

8.4.1 Optimizing Data Size

Design your tables to minimize their space on the disk. This can result in huge improvements by reducing the amount of data written to and read from disk. Smaller tables normally require less main memory while their contents are being actively processed during query execution. Any space reduction for table data also results in smaller indexes that can be processed faster.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application can give you a big performance gain. See Chapter 14, *Storage Engines*.

You can get better performance for a table and minimize storage space by using the techniques listed here:

Table Columns

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Declare columns to be `NOT NULL` if possible. It makes SQL operations faster, by enabling better use of indexes and eliminating overhead for testing whether each value is `NULL`. You also save some storage space, one bit per column. If you really need `NULL` values in your tables, use them. Just avoid the default setting that allows `NULL` values in every column.

Row Format

- For `MyISAM` tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but unfortunately may waste some space. See Section 14.1.3, “MyISAM Table Storage Formats”. You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE TABLE` option `ROW_FORMAT=FIXED`.

- Starting with MySQL 5.0.3, `InnoDB` tables use a more compact storage format. In earlier versions of MySQL, `InnoDB` rows contain some redundant information, such as the number of columns and the length of each column, even for fixed-size columns. By default, tables are created in the compact format (`ROW_FORMAT=COMPACT`). If you wish to downgrade to older versions of MySQL, you can request the old format with `ROW_FORMAT=REDUNDANT`.

  The presence of the compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

  The compact `InnoDB` format also changes how `CHAR` columns containing UTF-8 data are stored. With `ROW_FORMAT=REDUNDANT`, a UTF-8 `CHAR(`*`N`*`)` occupies 3 × *`N`* bytes, given that the maximum length of a UTF-8 encoded character is three bytes. Many languages can be written primarily using single-byte UTF-8 characters, so a fixed storage length often wastes space. With `ROW_FORMAT=COMPACT` format, `InnoDB` allocates a variable amount of storage in the range from *`N`* to 3 × *`N`* bytes for these columns by stripping trailing spaces if necessary. The minimum storage length is kept as *`N`* bytes to facilitate in-place updates in typical cases.

Indexes

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient.

- Create only the indexes that you need to improve query performance. Indexes are good for retrieval, but slow down insert and update operations. If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.

- If it is very likely that a string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see Section 13.1.8, “CREATE INDEX Syntax”). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See Section 8.12.2, “Tuning Server Parameters”.

Joins

- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.

- Declare columns with identical information in different tables with identical data types, to speed up joins based on the corresponding columns.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Keep column names simple, so that you can use the same name across different tables and simplify join queries. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.

Normalization

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). Instead of repeating lengthy values such as names and addresses, assign them unique IDs, repeat these IDs as needed across multiple smaller tables, and join the tables in queries by referencing the IDs in the join clause.
- If speed is more important than disk space and the maintenance costs of keeping multiple copies of data, for example in a business intelligence scenario where you analyze all the data from large tables, you can relax the normalization rules, duplicating information or creating summary tables to gain more speed.

8.4.2 Optimizing MySQL Data Types

8.4.2.1 Optimizing for Numeric Data

- For unique IDs or other values that can be represented as either strings or numbers, prefer numeric columns to string columns. Since large numeric values can be stored in fewer bytes than the corresponding strings, it is faster and takes less memory to transfer and compare them.
- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you can avoid parsing the text file to find line and column boundaries.

8.4.2.2 Optimizing for Character and String Types

For character and string columns, follow these guidelines:

- Use binary collation order for fast comparison and sort operations, when you do not need language-specific collation features. You can use the BINARY operator to use binary collation within a particular query.
- When comparing values from different columns, declare those columns with the same character set and collation wherever possible, to avoid string conversions while running the query.
- For column values less than 8KB in size, use binary `VARCHAR` instead of `BLOB`. The `GROUP BY` and `ORDER BY` clauses can generate temporary tables, and these temporary tables can use the `MEMORY` storage engine if the original table does not contain any `BLOB` columns.
- If a table contains string columns such as name and address, but many queries do not retrieve those columns, consider splitting the string columns into a separate table and using join queries with a foreign key when necessary. When MySQL retrieves any value from a row, it reads a data block containing all the columns of that row (and possibly other adjacent rows). Keeping each row small, with only the most frequently used columns, allows more rows to fit in each data block. Such compact tables reduce disk I/O and memory usage for common queries.
- When you use a randomly generated value as a primary key in an `InnoDB` table, prefix it with an ascending value such as the current date and time if possible. When consecutive primary values are physically stored near each other, `InnoDB` can insert and retrieve them faster.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- See Section 8.4.2.1, “Optimizing for Numeric Data” for reasons why a numeric column is usually preferable to an equivalent string column.

8.4.2.3 Optimizing for BLOB Types

- When storing a large blob containing textual data, consider compressing it first. Do not use this technique when the entire table is compressed by `InnoDB` or `MyISAM`.
- For a table with several columns, to reduce memory requirements for queries that do not use the BLOB column, consider splitting the BLOB column into a separate table and referencing it with a join query when needed.
- Since the performance requirements to retrieve and display a BLOB value might be very different from other data types, you could put the BLOB-specific table on a different storage device or even a separate database instance. For example, to retrieve a BLOB might require a large sequential disk read that is better suited to a traditional hard drive than to an SSD device.
- See Section 8.4.2.2, “Optimizing for Character and String Types” for reasons why a binary `VARCHAR` column is sometimes preferable to an equivalent BLOB column.
- Rather than testing for equality against a very long text string, you can store a hash of the column value in a separate column, index that column, and test the hashed value in queries. (Use the `MD5()` or `CRC32()` function to produce the hash value.) Since hash functions can produce duplicate results for different inputs, you still include a clause `AND blob_column = long_string_value` in the query to guard against false matches; the performance benefit comes from the smaller, easily scanned index for the hashed values.

8.4.2.4 Using PROCEDURE ANALYSE

`ANALYSE([max_elements[,max_memory]])`

`ANALYSE()` examines the result from a query and returns an analysis of the results that suggests optimal data types for each column that may help reduce table sizes. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- *`max_elements`* (default 256) is the maximum number of distinct values that `ANALYSE()` notices per column. This is used by `ANALYSE()` to check whether the optimal data type should be of type `ENUM`; if there are more than *`max_elements`* distinct values, then `ENUM` is not a suggested type.
- *`max_memory`* (default 8192) is the maximum amount of memory that `ANALYSE()` should allocate per column while trying to find all distinct values.

A `PROCEDURE` clause is not permitted in a `UNION` statement.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

8.4.3 Optimizing for Many Tables

Some techniques for keeping individual queries fast involve splitting data across many tables. When the number of tables runs into the thousands or even millions, the overhead of dealing with all these tables becomes a new performance consideration.

8.4.3.1 How MySQL Opens and Closes Tables

When you execute a `mysqladmin status` command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client sessions having different states on the same table, the table is opened independently by each concurrent session. This uses additional memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all sessions.)

The `table_cache` and `max_connections` system variables affect the maximum number of files the server keeps open. If you increase one or both of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems permit you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_cache` is related to `max_connections`. For example, for 200 concurrent running connections, you should have a table cache size of at least `200 * N`, where *`N`* is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_cache` setting. If `table_cache` is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available to MySQL using the `--open-files-limit` startup option to `mysqld`. See Section B.5.2.18, “File Not Found and Similar Errors”.

The cache of open tables is kept at a level of `table_cache` entries. The default value is 64; this can be changed with the `--table_cache` option to `mysqld`. Note that MySQL may temporarily open more tables than this to execute queries.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_cache` entries and a table in the cache is no longer being used by any threads.
- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, beginning with the table least recently used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary. When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A `MyISAM` table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any `MyISAM` table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER tbl_name OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER tbl_name CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See Section 13.2.4, “HANDLER Syntax”.

You can determine whether your table cache is too small by checking the `mysqld` status variable `Opened_tables`, which indicates the number of table-opening operations since the server started:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

If the value is very large or increases rapidly, even when you have not issued many `FLUSH TABLES` statements, you should increase the table cache size. See Section 5.1.4, “Server System Variables”, and Section 5.1.6, “Server Status Variables”.

8.4.3.2 Disadvantages of Creating Many Tables in the Same Database

If you have many `MyISAM` tables in the same database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by increasing the number of entries permitted in the table cache.

8.4.4 Internal Temporary Table Use in MySQL

In some cases, the server creates internal temporary tables while processing statements. Users have no direct control over when this occurs.

The server creates temporary tables under conditions such as these:

- Evaluation of `UNION` statements.
- Evaluation of some views, such those that use the `TEMPTABLE` algorithm, `UNION`, or aggregation.
- Evaluation of statements that contain an `ORDER BY` clause and a different `GROUP BY` clause, or for which the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue.
- Evaluation of `DISTINCT` combined with `ORDER BY` may require a temporary table.
- For queries that use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.
- Evaluation of multiple-table `UPDATE` statements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Evaluation of `GROUP_CONCAT()` or `COUNT(DISTINCT)` expressions.

To determine whether a statement requires a temporary table, use `EXPLAIN` and check the `Extra` column to see whether it says `Using temporary` (see Section 8.8.1, “Optimizing Queries with EXPLAIN”).

Storage Engines Used for Temporary Tables

An internal temporary table can be held in memory and processed by the `MEMORY` storage engine, or stored on disk and processed by the `MyISAM` storage engine.

If an internal temporary table is created as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` and `max_heap_table_size` values. This differs from `MEMORY` tables explicitly created with `CREATE TABLE`: For such tables, only the `max_heap_table_size` system variable determines how large the table is permitted to grow and there is no conversion to on-disk format.

Some conditions prevent the use of an in-memory temporary table, in which case the server uses an on-disk table instead:

- Presence of a `BLOB` or `TEXT` column in the table
- Presence of any string column in a `GROUP BY` or `DISTINCT` clause larger than 512 bytes
- Presence of any string column with a maximum length larger than 512 (bytes for binary strings, characters for nonbinary strings) in the `SELECT` list, if `UNION` or `UNION ALL` is used
- The `SHOW COLUMNS` and `DESCRIBE` statements use `BLOB` as the type for some columns, thus the temporary table used for the results is an on-disk table.

When the server creates an internal temporary table (either in memory or on disk), it increments the `Created_tmp_tables` status variable. If the server creates the table on disk (either initially or by converting an in-memory table) it increments the `Created_tmp_disk_tables` status variable.

Temporary Table Storage Format

Internal temporary tables are stored using fixed-length row format, whether managed by the `MEMORY` or `MyISAM` storage engine.

8.5 Optimizing for MyISAM Tables

The `MyISAM` storage engine performs best with read-mostly data or with low-concurrency operations, because table locks limit the ability to perform simultaneous updates.

8.5.1 Optimizing MyISAM Queries

Some general tips for speeding up queries on `MyISAM` tables:

- To help MySQL better optimize queries, use `ANALYZE TABLE` or run `myisamchk --analyze` on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a nonconstant expression. You can check the result from the table analysis by using `SHOW INDEX FROM` *`tbl_name`* and examining the `Cardinality` value. `myisamchk --description --verbose` shows index distribution information.
- To sort an index and data according to an index, use `myisamchk --sort-index --sort-records=1` (assuming that you want to sort on index 1). This is a good way to make queries faster if

This documentation is for an older version. If you're
This documentation is for an older version. If you're

you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.

- Try to avoid complex `SELECT` queries on `MyISAM` tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.
- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the `concurrent_insert` variable. You can force new rows to be appended (and therefore permit concurrent inserts), even in tables that have deleted rows. See Section 8.11.3, “Concurrent Inserts”.
- For `MyISAM` tables that change frequently, try to avoid all variable-length columns (`VARCHAR`, `BLOB`, and `TEXT`). The table uses dynamic row format if it includes even a single variable-length column. See Chapter 14, *Storage Engines*.
- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a `MyISAM` table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See Chapter 14, *Storage Engines*.
- Use `ALTER TABLE ... ORDER BY` *`expr1, expr2, ...`* if you usually retrieve rows in *`expr1, expr2, ...`* order. By using this option after extensive changes to the table, you may be able to get higher performance.
- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

  ```
  UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
  ```

  This is very important when you use a MySQL storage engine such as `MyISAM` that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.
- Use `INSERT DELAYED` when you do not need to know when your data is written. This reduces the overall insertion impact because many rows can be written with a single disk write.
- Use `INSERT LOW_PRIORITY` when you want to give `SELECT` statements higher priority than your inserts.

  Use `SELECT HIGH_PRIORITY` to get retrievals that jump the queue. That is, the `SELECT` is executed even if there is another client waiting to do a write.

  `LOW_PRIORITY` and `HIGH_PRIORITY` have an effect only for storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- Use `OPTIMIZE TABLE` periodically to avoid fragmentation with dynamic-format `MyISAM` tables. See Section 14.1.3, “MyISAM Table Storage Formats”.
- Declaring a `MyISAM` table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you must ensure that the table is okay by running the server with the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`--myisam-recover` option, or by running `myisamchk` before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)

- Strings are automatically prefix- and end-space compressed in `MyISAM` indexes. See Section 13.1.8, “CREATE INDEX Syntax”.
- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. Locking the table during this operation ensures that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see Section 8.10.3, “The MySQL Query Cache”.

8.5.2 Bulk Data Loading for MyISAM Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.2.1, “Speed of INSERT Statements”.

- To improve performance when multiple clients insert a lot of rows, use the `INSERT DELAYED` statement. See Section 13.2.5.2, “INSERT DELAYED Syntax”. This technique works for `MyISAM` and some other storage engines, but not `InnoDB`.
- For a `MyISAM` table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running, if there are no deleted rows in middle of the data file. See Section 8.11.3, “Concurrent Inserts”.
- With some extra work, it is possible to make `LOAD DATA INFILE` run even faster for a `MyISAM` table when the table has many indexes. Use the following procedure:

  1. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.
  2. Use `myisamchk --keys-used=0 -rq /path/to/db/tbl_name` to remove all use of indexes for the table.
  3. Insert data into the table with `LOAD DATA INFILE`. This does not update any indexes and therefore is very fast.
  4. If you intend only to read from the table in the future, use `myisampack` to compress it. See Section 14.1.3.3, “Compressed Table Characteristics”.
  5. Re-create the indexes with `myisamchk -rq /path/to/db/tbl_name`. This creates the index tree in memory before writing it to disk, which is much faster than updating the index during `LOAD DATA INFILE` because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
  6. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.

  `LOAD DATA INFILE` performs the preceding optimization automatically if the `MyISAM` table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let `myisamchk` allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA INFILE` statement.

  You can also disable or enable the nonunique indexes for a `MyISAM` table by using the following statements rather than `myisamchk`. If you use these statements, you can skip the `FLUSH TABLE` operations:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up `INSERT` operations that are performed with multiple statements for nontransactional tables, lock your tables:

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

  This benefits performance because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally, there would be as many index buffer flushes as there are `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single `INSERT`.

  Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

  - Connection 1 does 1000 inserts
  - Connections 2, 3, and 4 do 1 insert
  - Connection 5 does 1000 inserts

  If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

  `INSERT`, `UPDATE`, and `DELETE` operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a `LOCK TABLES` followed by an `UNLOCK TABLES` once in a while (each 1,000 rows or so) to permit other threads to access table. This would still result in a nice performance gain.

  `INSERT` is still much slower for loading data than `LOAD DATA INFILE`, even when using the strategies just outlined.

- To increase performance for `MyISAM` tables, for both `LOAD DATA INFILE` and `INSERT`, enlarge the key cache by increasing the `key_buffer_size` system variable. See Section 8.12.2, “Tuning Server Parameters”.

8.5.3 Speed of REPAIR TABLE Statements

`REPAIR TABLE` for `MyISAM` tables is similar to using `myisamchk` for repair operations, and some of the same performance optimizations apply:

- `myisamchk` has variables that control memory allocation. You may be able to its improve performance by setting these variables, as described in Section 4.6.3.6, “myisamchk Memory Usage”.
- For `REPAIR TABLE`, the same principle applies, but because the repair is done by the server, you set server system variables instead of `myisamchk` variables. Also, In addition to setting memory-allocation variables, increasing the `myisam_max_sort_file_size` system variable increases the likelihood that the repair will use the faster filesort method and avoid the slower repair by key cache method. Set the variable to the maximum file size for your system, after checking to be sure that there is enough free space to hold a copy of the table files. The free space must be available in the file system containing the original table files.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Suppose that a `myisamchk` table-repair operation is done using the following options to set its memory-allocation variables:

```
--key_buffer_size=128M --sort_buffer_size=256M
--read_buffer_size=64M --write_buffer_size=64M
```

Some of those `myisamchk` variables correspond to server system variables:

`myisamchk` Variable	**System Variable**
`key_buffer_size`	`key_buffer_size`
`sort_buffer_size`	`myisam_sort_buffer_size`
`read_buffer_size`	`read_buffer_size`
`write_buffer_size`	none

Each of the server system variables can be set at runtime, and some of them (`myisam_sort_buffer_size`, `read_buffer_size`) have a session value in addition to a global value. Setting a session value limits the effect of the change to your current session and does not affect other users. Changing a global-only variable (`key_buffer_size`, `myisam_max_sort_file_size`) affects other users as well. For `key_buffer_size`, you must take into account that the buffer is shared with those users. For example, if you set the `myisamchk key_buffer_size` variable to 128MB, you could set the corresponding `key_buffer_size` system variable larger than that (if it is not already set larger), to permit key buffer use by activity in other sessions. However, changing the global key buffer size invalidates the buffer, causing increased disk I/O and slowdown for other sessions. An alternative that avoids this problem is to use a separate key cache, assign to it the indexes from the table to be repaired, and deallocate it when the repair is complete. See Section 8.10.1.2, “Multiple Key Caches”.

Based on the preceding remarks, a `REPAIR TABLE` operation can be done as follows to use settings similar to the `myisamchk` command. Here a separate 128MB key buffer is allocated and the file system is assumed to permit a file size of at least 100GB.

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name ;
SET GLOBAL repair_cache.key_buffer_size = 0;
```

If you intend to change a global variable but want to do so only for the duration of a `REPAIR TABLE` operation to minimally affect other users, save its value in a user variable and restore it afterward. For example:

```
SET @old_myisam_sort_buffer_size = @@global.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name ;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

The system variables that affect `REPAIR TABLE` can be set globally at server startup if you want the values to be in effect by default. For example, add these lines to the server `my.cnf` file:

```
[mysqld]
myisam_sort_buffer_size=256M
key_buffer_size=1G
myisam_max_sort_file_size=100G
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

These settings do not include `read_buffer_size`. Setting `read_buffer_size` globally to a large value does so for all sessions and can cause performance to suffer due to excessive memory allocation for a server with many simultaneous sessions.

8.6 Optimizing for InnoDB Tables

This section explains how to optimize database operations for `InnoDB` tables.

8.6.1 Optimizing Storage Layout for InnoDB Tables

- Once your data reaches a stable size, or a growing table has increased by tens or some hundreds of megabytes, consider using the `OPTIMIZE TABLE` statement to reorganize the table and compact any wasted space. The reorganized tables require less disk I/O to perform full table scans. This is a straightforward technique that can improve performance when other techniques such as improving index usage or tuning application code are not practical.

  `OPTIMIZE TABLE` copies the data part of the table and rebuilds the indexes. The benefits come from improved packing of data within indexes, and reduced fragmentation within the tablespaces and on disk. The benefits vary depending on the data in each table. You may find that there are significant gains for some and not for others, or that the gains decrease over time until you next optimize the table. This operation can be slow if the table is large or if the indexes being rebuilt do not fit into the buffer pool. The first run after adding a lot of data to a table is often much slower than later runs.

- In `InnoDB`, having a long `PRIMARY KEY` (either a single column with a lengthy value, or several columns that form a long composite value) wastes a lot of disk space. The primary key value for a row is duplicated in all the secondary index records that point to the same row. (See Section 14.2.10, “InnoDB Table and Index Structures”.) Create an `AUTO_INCREMENT` column as the primary key if your primary key is long, or index a prefix of a long `VARCHAR` column instead of the entire column.

- Use the `VARCHAR` data type instead of `CHAR` to store variable-length strings or for columns with many `NULL` values. A `CHAR(N)` column always takes `N` characters to store data, even if the string is shorter or its value is `NULL`. Smaller tables fit better in the buffer pool and reduce disk I/O.

  When using `COMPACT` row format (the default `InnoDB` format) and variable-length character sets, such as `utf8` or `sjis`, `CHAR(N)` columns occupy a variable amount of space, but still at least `N` bytes.

- For tables that are big, or contain lots of repetitive text or numeric data, consider using `COMPRESSED` row format. Less disk I/O is required to bring data into the buffer pool, or to perform full table scans. Before making a permanent decision, measure the amount of compression you can achieve by using `COMPRESSED` versus `COMPACT` row format.

8.6.2 Optimizing InnoDB Transaction Management

To optimize `InnoDB` transaction processing, find the ideal balance between the performance overhead of transactional features and the workload of your server. For example, an application might encounter performance issues if it commits thousands of times per second, and different performance issues if it commits only every 2-3 hours.

- The default MySQL setting `AUTOCOMMIT=1` can impose performance limitations on a busy database server. Where practical, wrap several related DML operations into a single transaction, by issuing `SET AUTOCOMMIT=0` or a `START TRANSACTION` statement, followed by a `COMMIT` statement after making all the changes.

  `InnoDB` must flush the log to disk at each transaction commit if that transaction made modifications to the database. When each change is followed by a commit (as with the default autocommit setting), the I/O throughput of the storage device puts a cap on the number of potential operations per second.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Avoid performing rollbacks after inserting, updating, or deleting huge numbers of rows. If a big transaction is slowing down server performance, rolling it back can make the problem worse, potentially taking several times as long to perform as the original DML operations. Killing the database process does not help, because the rollback starts again on server startup.

  To minimize the chance of this issue occurring:

  - Increase the size of the buffer pool so that all the DML changes can be cached rather than immediately written to disk.
  - Consider issuing `COMMIT` statements periodically during the big DML operation, possibly breaking a single delete or update into multiple statements that operate on smaller numbers of rows.

  To get rid of a runaway rollback once it occurs, increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or kill the server and restart with `innodb_force_recovery=3`, as explained in Section 14.2.6.1, “The InnoDB Recovery Process”.

- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the `innodb_flush_log_at_trx_commit` parameter to 0. `InnoDB` tries to flush the log once per second anyway, although the flush is not guaranteed. Also, set the value of `innodb_support_xa` to 0, which will reduce the number of disk flushes due to synchronizing on disk data and the binary log.

- When rows are modified or deleted, the rows and associated undo logs are not physically removed immediately, or even immediately after the transaction commits. The old data is preserved until transactions that started earlier or concurrently are finished, so that those transactions can access the previous state of modified or deleted rows. Thus, a long-running transaction can prevent `InnoDB` from purging data that was changed by a different transaction.

- When rows are modified or deleted within a long-running transaction, other transactions using the `READ COMMITTED` and `REPEATABLE READ` isolation levels have to do more work to reconstruct the older data if they read those same rows.

- When a long-running transaction modifies a table, queries against that table from other transactions do not make use of the covering index technique. Queries that normally could retrieve all the result columns from a secondary index, instead look up the appropriate values from the table data.

8.6.3 Optimizing InnoDB Redo Logging

Consider the following guidelines for optimizing redo logging:

- Make your redo log files big, even as big as the buffer pool. When `InnoDB` has written the redo log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small redo log files cause many unnecessary disk writes. Although historically big redo log files caused lengthy recovery times, recovery is now much faster and you can confidently use large redo log files.

  The size and number of redo log files are configured using the `innodb_log_file_size` and `innodb_log_files_in_group` configuration options. For information about modifying an existing redo log file configuration, see Section 14.2.4, “Changing the Number or Size of InnoDB Redo Log Files”.

- Consider increasing the size of the log_buffer. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. Log buffer size is configured using the `innodb_log_buffer_size` configuration option.

8.6.4 Bulk Data Loading for InnoDB Tables

This documentation is for an older version. If you're
This documentation is for an older version. If you're

These performance tips supplement the general guidelines for fast inserts in Section 8.2.2.1, “Speed of INSERT Statements”.

- When importing data into `InnoDB`, turn off autocommit mode, because it performs a log flush to disk for every insert. To disable autocommit during your import operation, surround it with `SET autocommit` and `COMMIT` statements:

```
SET autocommit=0;
... SQL import statements ...
COMMIT;
```

  The `mysqldump` option `--opt` creates dump files that are fast to import into an `InnoDB` table, even without wrapping them with the `SET autocommit` and `COMMIT` statements.

- If you have `UNIQUE` constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET unique_checks=0;
... SQL import statements ...
SET unique_checks=1;
```

  For big tables, this saves a lot of disk I/O because `InnoDB` can use its change buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys.

- If you have `FOREIGN KEY` constraints in your tables, you can speed up table imports by turning off the foreign key checks for the duration of the import session:

```
SET foreign_key_checks=0;
... SQL import statements ...
SET foreign_key_checks=1;
```

  For big tables, this can save a lot of disk I/O.

- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

  This tip is valid for inserts into any table, not just `InnoDB` tables.

8.6.5 Optimizing InnoDB Queries

To tune queries for `InnoDB` tables, create an appropriate set of indexes on each table. See Section 8.3.1, “How MySQL Uses Indexes” for details. Follow these guidelines for `InnoDB` indexes:

- Because each `InnoDB` table has a primary key (whether you request one or not), specify a set of primary key columns for each table, columns that are used in the most important and time-critical queries.

- Do not specify too many or too long columns in the primary key, because these column values are duplicated in each secondary index. When an index contains unnecessary data, the I/O to read this data and memory to cache it reduce the performance and scalability of the server.

- Do not create a separate secondary index for each column, because each query can only make use of one index. Indexes on rarely tested columns or columns with only a few different values might not be helpful for any queries. If you have many queries for the same table, testing different combinations of

This documentation is for an older version. If you're
This documentation is for an older version. If you're

columns, try to create a small number of concatenated indexes rather than a large number of single-column indexes. If an index contains all the columns needed for the result set (known as a covering index), the query might be able to avoid reading the table data at all.

- If an indexed column cannot contain any `NULL` values, declare it as `NOT NULL` when you create the table. The optimizer can better determine which index is most effective to use for a query, when it knows whether each column contains `NULL` values.

- If you often have recurring queries for tables that are not updated frequently, enable the query cache:

```
[mysqld]
query_cache_type = 1
query_cache_size = 10M
```

8.6.6 Optimizing InnoDB DDL Operations

- "Fast index creation" makes it faster in some cases to drop an index before loading data into a table, then re-create the index after loading the data.

- Use `TRUNCATE TABLE` to empty a table, not `DELETE FROM tbl_name`. Foreign key constraints can make a `TRUNCATE` statement work like a regular `DELETE` statement, in which case a sequence of commands like `DROP TABLE` and `CREATE TABLE` might be fastest.

- Because the primary key is integral to the storage layout of each `InnoDB` table, and changing the definition of the primary key involves reorganizing the whole table, always set up the primary key as part of the `CREATE TABLE` statement, and plan ahead so that you do not need to `ALTER` or `DROP` the primary key afterward.

8.6.7 Optimizing InnoDB Disk I/O

If you follow the best practices for database design and the tuning techniques for SQL operations, but your database is still slowed by heavy disk I/O activity, explore these low-level techniques related to disk I/O. If the Unix `top` tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound.

- When table data is cached in the `InnoDB` buffer pool, it can be accessed repeatedly by queries without requiring any disk I/O. Specify the size of the buffer pool with the `innodb_buffer_pool_size` option. This memory area is important enough that busy databases often specify a size approximately 80% of the amount of physical memory. For more information, see Section 8.10.2, "The InnoDB Buffer Pool".

- In some versions of GNU/Linux and Unix, flushing files to disk with the Unix `fsync()` call (which `InnoDB` uses by default) and similar methods is surprisingly slow. If database write performance is an issue, conduct benchmarks with the `innodb_flush_method` parameter set to `O_DSYNC`.

- When using the `InnoDB` storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), use direct I/O for `InnoDB`-related files, to avoid degradation of `InnoDB` performance. To use direct I/O for an entire UFS file system used for storing `InnoDB`-related files, mount it with the `forcedirectio` option; see `mount_ufs(1M)`. (The default on Solaris 10/x86_64 is *not* to use this option.) Alternatively, as of MySQL 5.0.42, to apply direct I/O only to `InnoDB` file operations rather than the whole file system, set `innodb_flush_method = O_DIRECT`. With this setting, `InnoDB` calls `directio()` instead of `fcntl()` for I/O to data files (not for I/O to log files).

- When using the `InnoDB` storage engine with a large `innodb_buffer_pool_size` value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), conduct benchmarks with `InnoDB` data files and log files on raw devices or on a separate direct I/O UFS file system, using the `forcedirectio` mount option as described earlier. (It is necessary to use the mount option rather than

This documentation is for an older version. If you're
This documentation is for an older version. If you're

setting `innodb_flush_method` if you want direct I/O for the log files.) Users of the Veritas file system VxFS should use the `convosync=direct` mount option.

Do not place other MySQL data files, such as those for `MyISAM` tables, on a direct I/O file system. Executables or libraries *must not* be placed on a direct I/O file system.

- If you have additional storage devices available to set up a RAID configuration or symbolic links to different disks, Section 8.12.3, “Optimizing Disk I/O” for additional low-level I/O tips.
- Other `InnoDB` configuration options to consider when tuning I/O-bound workloads include `innodb_log_buffer_size`, `innodb_log_file_size`, `innodb_max_dirty_pages_pct`, `innodb_max_purge_lag`, `innodb_open_files`, and `sync_binlog`.

8.6.8 Optimizing InnoDB for Systems with Many Tables

- `InnoDB` computes index cardinality values for a table the first time that table is accessed after startup, instead of storing such values in the table. This step can take significant time on systems that partition the data into many tables. Since this overhead only applies to the initial table open operation, to “warm up” a table for later use, access it immediately after startup by issuing a statement such as `SELECT 1 FROM tbl_name LIMIT 1`.

8.7 Optimizing for MEMORY Tables

Consider using `MEMORY` tables for noncritical data that is accessed often, and is read-only or rarely updated. Benchmark your application against equivalent `InnoDB` or `MyISAM` tables under a realistic workload, to confirm that any additional performance is worth the risk of losing data, or the overhead of copying data from a disk-based table at application start.

For best performance with `MEMORY` tables, examine the kinds of queries against each table, and specify the type to use for each associated index, either a B-tree index or a hash index. On the `CREATE INDEX` statement, use the clause `USING BTREE` or `USING HASH`. B-tree indexes are fast for queries that do greater-than or less-than comparisons through operators such as `>` or `BETWEEN`. Hash indexes are only fast for queries that look up single values through the `=` operator, or a restricted set of values through the `IN` operator. For why `USING BTREE` is often a better choice than the default `USING HASH`, see Section 8.2.1.17, “How to Avoid Full Table Scans”. For implementation details of the different types of `MEMORY` indexes, see Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

8.8 Understanding the Query Execution Plan

Depending on the details of your tables, columns, indexes, and the conditions in your `WHERE` clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query. A query on a huge table can be performed without reading all the rows; a join involving several tables can be performed without comparing every combination of rows. The set of operations that the optimizer chooses to perform the most efficient query is called the “query execution plan”, also known as the `EXPLAIN` plan. Your goals are to recognize the aspects of the `EXPLAIN` plan that indicate a query is optimized well, and to learn the SQL syntax and indexing techniques to improve the plan if you see some inefficient operations.

8.8.1 Optimizing Queries with EXPLAIN

The `EXPLAIN` statement can be used to obtain information about how MySQL executes a statement:

- When you precede a `SELECT` statement with the keyword `EXPLAIN`, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using `EXPLAIN` to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `EXPLAIN EXTENDED` can be used to obtain additional execution plan information. See Section 8.8.3, “EXPLAIN EXTENDED Output Format”.

With the help of `EXPLAIN`, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 13.2.8, “SELECT Syntax”.)

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.

Note

`EXPLAIN` can also be used to obtain information about the columns in a table. `EXPLAIN tbl_name` is synonymous with `DESCRIBE tbl_name` and `SHOW COLUMNS FROM tbl_name`. For more information, see Section 13.8.1, “DESCRIBE Syntax”, and Section 13.7.5.5, “SHOW COLUMNS Syntax”.

8.8.2 EXPLAIN Output Format

The `EXPLAIN` statement provides information about the execution plan for a `SELECT` statement.

`EXPLAIN` returns a row of information for each table used in the `SELECT` statement. It lists the tables in the output in the order that MySQL would read them while processing the statement. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

When the `EXTENDED` keyword is used, `EXPLAIN` produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. See Section 8.8.3, “EXPLAIN EXTENDED Output Format”.

- `EXPLAIN` Output Columns
- `EXPLAIN` Join Types
- `EXPLAIN` Extra Information
- `EXPLAIN` Output Interpretation

EXPLAIN Output Columns

This section describes the output columns produced by `EXPLAIN`. Later sections provide additional information about the `type` and `Extra` columns.

Each output row from `EXPLAIN` provides information about one table. Each row contains the values summarized in Table 8.1, “EXPLAIN Output Columns”, and described in more detail following the table.

Table 8.1 EXPLAIN Output Columns

Column	Meaning
`id`	The `SELECT` identifier
`select_type`	The `SELECT` type

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Column	Meaning
`table`	The table for the output row
`type`	The join type
`possible_keys`	The possible indexes to choose
`key`	The index actually chosen
`key_len`	The length of the chosen key
`ref`	The columns compared to the index
`rows`	Estimate of rows to be examined
`Extra`	Additional information

- `id`

  The `SELECT` identifier. This is the sequential number of the `SELECT` within the query. The value can be `NULL` if the row refers to the union result of other rows. In this case, the `table` column shows a value like `<unionM,N>` to indicate that the row refers to the union of the rows with `id` values of *`M`* and *`N`*.

- `select_type`

  The type of `SELECT`, which can be any of those shown in the following table.

`select_type` Value	Meaning
`SIMPLE`	Simple `SELECT` (not using `UNION` or subqueries)
`PRIMARY`	Outermost `SELECT`
`UNION`	Second or later `SELECT` statement in a `UNION`
`DEPENDENT UNION`	Second or later `SELECT` statement in a `UNION`, dependent on outer query
`UNION RESULT`	Result of a `UNION`.
`SUBQUERY`	First `SELECT` in subquery
`DEPENDENT SUBQUERY`	First `SELECT` in subquery, dependent on outer query
`DERIVED`	Derived table `SELECT` (subquery in `FROM` clause)
`UNCACHEABLE SUBQUERY`	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query

  `DEPENDENT` typically signifies the use of a correlated subquery. See Section 13.2.9.7, “Correlated Subqueries”.

  `DEPENDENT SUBQUERY` evaluation differs from `UNCACHEABLE SUBQUERY` evaluation. For `DEPENDENT SUBQUERY`, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For `UNCACHEABLE SUBQUERY`, the subquery is re-evaluated for each row of the outer context.

  Cacheability of subqueries differs from caching of query results in the query cache (which is described in Section 8.10.3.1, “How the Query Cache Operates”). Subquery caching occurs during query execution, whereas the query cache is used to store results only after query execution finishes.

- `table`

  The name of the table to which the row of output refers. This can also be one of the following values:

  - `<unionM,N>`: The row refers to the union of the rows with `id` values of *`M`* and *`N`*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - `<derived*N*>`: The row refers to the derived table result for the row with an `id` value of *`N`*. A derived table may result, for example, from a subquery in the `FROM` clause.

- `type`

  The join type. For descriptions of the different types, see `EXPLAIN` Join Types.

- `possible_keys`

  The `possible_keys` column indicates which indexes MySQL can choose from use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from `EXPLAIN`. That means that some of the keys in `possible_keys` might not be usable in practice with the generated table order.

  If this column is `NULL`, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the `WHERE` clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with `EXPLAIN` again. See Section 13.1.4, “ALTER TABLE Syntax”.

  To see what indexes a table has, use `SHOW INDEX FROM *tbl_name*`.

- `key`

  The `key` column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the `possible_keys` indexes to look up rows, that index is listed as the key value.

  It is possible that `key` will name an index that is not present in the `possible_keys` value. This can happen if none of the `possible_keys` indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

  For `InnoDB`, a secondary index might cover the selected columns even if the query also selects the primary key because `InnoDB` stores the primary key value with each secondary index. If `key` is `NULL`, MySQL found no index to use for executing the query more efficiently.

  To force MySQL to use or ignore an index listed in the `possible_keys` column, use `FORCE INDEX`, `USE INDEX`, or `IGNORE INDEX` in your query. See Section 8.9.2, “Index Hints”.

  For `MyISAM`, `NDB`, and `BDB` tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For `MyISAM` tables, `myisamchk --analyze` does the same as `ANALYZE TABLE`. See Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

- `key_len`

  The `key_len` column indicates the length of the key that MySQL decided to use. The length is `NULL` if the `key` column says `NULL`. Note that the value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses.

- `ref`

  The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

- `rows`

  The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For `InnoDB` tables, this number is an estimate, and may not always be exact.

- `Extra`

  This column contains additional information about how MySQL resolves the query. For descriptions of the different values, see `EXPLAIN` Extra Information.

EXPLAIN Join Types

The `type` column of `EXPLAIN` output describes how tables are joined. The following list describes the join types, ordered from the best type to the worst:

- `system`

  The table has only one row (= system table). This is a special case of the `const` join type.

- `const`

  The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` tables are very fast because they are read only once.

  `const` is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index to constant values. In the following queries, *`tbl_name`* can be used as a `const` table:

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
  WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- `eq_ref`

  One row is read from this table for each combination of rows from the previous tables. Other than the `system` and `const` types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE NOT NULL` index.

  `eq_ref` can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an `eq_ref` join to process *`ref_table`*:

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `ref`

  All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

  `ref` can be used for indexed columns that are compared using the `=` or `<=>` operator. In the following examples, MySQL can use a `ref` join to process *`ref_table`*:

This documentation is for an older version. If you're

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `fulltext`

  The join is performed using a `FULLTEXT` index.

- `ref_or_null`

  This join type is like `ref`, but with the addition that MySQL does an extra search for rows that contain `NULL` values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a `ref_or_null` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table
    WHERE key_column=expr OR key_column IS NULL;
  ```

  See Section 8.2.1.6, “IS NULL Optimization”.

- `index_merge`

  This join type indicates that the Index Merge optimization is used. In this case, the `key` column in the output row contains a list of indexes used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see Section 8.2.1.4, “Index Merge Optimization”.

- `unique_subquery`

  This type replaces `eq_ref` for some `IN` subqueries of the following form:

  ```
  value IN (SELECT primary_key FROM single_table WHERE some_expr)
  ```

  `unique_subquery` is just an index lookup function that replaces the subquery completely for better efficiency.

- `index_subquery`

  This join type is similar to `unique_subquery`. It replaces `IN` subqueries, but it works for nonunique indexes in subqueries of the following form:

  ```
  value IN (SELECT key_column FROM single_table WHERE some_expr)
  ```

- `range`

  Only rows that are in a given range are retrieved, using an index to select the rows. The `key` column in the output row indicates which index is used. The `key_len` contains the longest key part that was used. The `ref` column is `NULL` for this type.

  `range` can be used when a key column is compared to a constant using any of the `=`, `<>`, `>`, `>=`, `<`, `<=`, `IS NULL`, `<=>`, `BETWEEN`, or `IN()` operators:

  ```
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM tbl_name
  WHERE key_column = 10;

SELECT * FROM tbl_name
  WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
  WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
  WHERE key_part1= 10 AND key_part2 IN (10,20,30);
```

- `index`

  The `index` join type is the same as `ALL`, except that the index tree is scanned. This occurs two ways:

  - If the index is a covering index for the queries and can be used to satisfy all data required from the table, only the index tree is scanned. In this case, the `Extra` column says `Using index`. An index-only scan usually is faster than `ALL` because the size of the index usually is smaller than the table data.
  - A full table scan is performed using reads from the index to look up data rows in index order. `Uses index` does not appear in the `Extra` column.

  MySQL can use this join type when the query uses only columns that are part of a single index.

- `ALL`

  A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked `const`, and usually *very* bad in all other cases. Normally, you can avoid `ALL` by adding indexes that enable row retrieval from the table based on constant values or column values from earlier tables.

EXPLAIN Extra Information

The `Extra` column of `EXPLAIN` output contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. If you want to make your queries as fast as possible, look out for `Extra` values of `Using filesort` and `Using temporary`.

- `const row not found`

  For a query such as `SELECT ... FROM tbl_name`, the table was empty.

- `Distinct`

  MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- `Full scan on NULL key`

  This occurs for subquery optimization as a fallback strategy when the optimizer cannot use an index-lookup access method.

- `Impossible HAVING`

  The `HAVING` clause is always false and cannot select any rows.

- `Impossible WHERE`

  The `WHERE` clause is always false and cannot select any rows.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Impossible WHERE noticed after reading const tables`

  MySQL has read all `const` (and `system`) tables and notice that the `WHERE` clause is always false.

- `No matching min/max row`

  No row satisfies the condition for a query such as `SELECT MIN(...) FROM ... WHERE condition`.

- `no matching row in const table`

  For a query with a join, there was an empty table or a table with no rows satisfying a unique index condition.

- `No tables used`

  The query has no `FROM` clause, or has a `FROM DUAL` clause.

- `Not exists`

  MySQL was able to do a `LEFT JOIN` optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the `LEFT JOIN` criteria. Here is an example of the type of query that can be optimized this way:

  ```
  SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
    WHERE t2.id IS NULL;
  ```

  Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

- `Range checked for each record (index map: N)`

  MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a `range` or `index_merge` access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in Section 8.2.1.3, “Range Optimization”, and Section 8.2.1.4, “Index Merge Optimization”, with the exception that all column values for the preceding table are known and considered to be constants.

  Indexes are numbered beginning with 1, in the same order as shown by `SHOW INDEX` for the table. The index map value *`N`* is a bitmask value that indicates which indexes are candidates. For example, a value of `0x19` (binary 11001) means that indexes 1, 4, and 5 will be considered.

- `Select tables optimized away`

  The optimizer determined 1) that at most one row should be returned, and 2) that to produce this row, a deterministic set of rows must be read. When the rows to be read can be read during the optimization phase (for example, by reading index rows), there is no need to read any tables during query execution.

  The first condition is fulfilled when the query is implicitly grouped (contains an aggregate function but no `GROUP BY` clause). The second condition is fulfilled when one row lookup is performed per index used. The number of indexes read determines the number of rows to read.

  Consider the following implicitly grouped query:

  ```
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT MIN(c1), MIN(c2) FROM t1;
```

Suppose that `MIN(c1)` can be retrieved by reading one index row and `MIN(c2)` can be retrieved by reading one row from a different index. That is, for each column `c1` and `c2`, there exists an index where the column is the first column of the index. In this case, one row is returned, produced by reading two deterministic rows.

This `Extra` value does not occur if the rows to read are not deterministic. Consider this query:

```
SELECT MIN(c2) FROM t1 WHERE c1 <= 10;
```

Suppose that `(c1, c2)` is a covering index. Using this index, all rows with `c1 <= 10` must be scanned to find the minimum `c2` value. By contrast, consider this query:

```
SELECT MIN(c2) FROM t1 WHERE c1 = 10;
```

In this case, the first index row with `c1 = 10` contains the minimum `c2` value. Only one row must be read to produce the returned row.

For storage engines that maintain an exact row count per table (such as `MyISAM`, but not `InnoDB`), this `Extra` value can occur for `COUNT(*)` queries for which the `WHERE` clause is missing or always true and there is no `GROUP BY` clause. (This is an instance of an implicitly grouped query where the storage engine influences whether a deterministic number of rows can be read.)

- `unique row not found`

  For a query such as `SELECT ... FROM tbl_name`, no rows satisfy the condition for a `UNIQUE` index or `PRIMARY KEY` on the table.

- `Using filesort`

  MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 8.2.1.11, “ORDER BY Optimization”.

- `Using index`

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

- `Using index for group-by`

  Similar to the `Using index` table access method, `Using index for group-by` indicates that MySQL found an index that can be used to retrieve all columns of a `GROUP BY` or `DISTINCT` query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 8.2.1.12, “GROUP BY Optimization”.

- `Using sort_union(...)`, `Using union(...)`, `Using intersect(...)`

  These indicate how index scans are merged for the `index_merge` join type. See Section 8.2.1.4, “Index Merge Optimization”.

- `Using temporary`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains `GROUP BY` and `ORDER BY` clauses that list columns differently.

- `Using where`

  A `WHERE` clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the `Extra` value is not `Using where` and the table join type is `ALL` or `index`. Even if you are using an index for all parts of a `WHERE` clause, you may see `Using where` if the column can be `NULL`.

- `Using where with pushed condition`

  This item applies to `NDB` tables *only*. It means that MySQL Cluster is using the Condition Pushdown optimization to improve the efficiency of a direct comparison between a nonindexed column and a constant. In such cases, the condition is "pushed down" to the cluster's data nodes and is evaluated on all data nodes simultaneously. This eliminates the need to send nonmatching rows over the network, and can speed up such queries by a factor of 5 to 10 times over cases where Condition Pushdown could be but is not used. For more information, see Section 8.2.1.5, "Engine Condition Pushdown Optimization".

EXPLAIN Output Interpretation

You can get a good indication of how good a join is by taking the product of the values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the `max_join_size` system variable, this row product also is used to determine which multiple-table `SELECT` statements to execute and which to abort. See Section 8.12.2, "Tuning Server Parameters".

The following example shows how a multiple-table join can be optimized progressively based on the information provided by `EXPLAIN`.

Suppose that you have the `SELECT` statement shown here and that you plan to examine it using `EXPLAIN`:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
               tt.ProjectReference, tt.EstimatedShipDate,
               tt.ActualShipDate, tt.ClientID,
               tt.ServiceCodes, tt.RepetitiveID,
               tt.CurrentProcess, tt.CurrentDPPerson,
               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
               et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
          AND tt.ActualPC = et.EMPLOYID
          AND tt.AssignedPC = et_1.EMPLOYID
          AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows.

Table	Column	Data Type
`tt`	`ActualPC`	`CHAR(10)`
`tt`	`AssignedPC`	`CHAR(10)`
`tt`	`ClientID`	`CHAR(10)`
`et`	`EMPLOYID`	`CHAR(15)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table	Column	Data Type
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes.

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The tt.ActualPC values are not evenly distributed.

Initially, before any optimizations have been performed, the EXPLAIN statement produces the following information:

```
table type possible_keys key  key_len ref  rows  Extra
et    ALL  PRIMARY       NULL NULL    NULL 74
do    ALL  PRIMARY       NULL NULL    NULL 2135
et_1  ALL  PRIMARY       NULL NULL    NULL 74
tt    ALL  AssignedPC,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      Range checked for each record (index map: 0x23)
```

Because type is ALL for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is 74 × 2135 × 74 × 3872 = 45,268,558,720 rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. tt.ActualPC is declared as CHAR(10) and et.EMPLOYID is CHAR(15), so there is a length mismatch.

To fix this disparity between column lengths, use ALTER TABLE to lengthen ActualPC from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now tt.ActualPC and et.EMPLOYID are both VARCHAR(15). Executing the EXPLAIN statement again produces this result:

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                        where
             ActualPC
do    ALL    PRIMARY       NULL    NULL    NULL        2135
      Range checked for each record (index map: 0x1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      Range checked for each record (index map: 0x1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This is not perfect, but is much better: The product of the `rows` values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    ->                MODIFY ClientID   VARCHAR(15);
```

After that modification, `EXPLAIN` produces the output shown here:

```
table type   possible_keys key      key_len ref           rows Extra
et    ALL    PRIMARY       NULL     NULL    NULL          74
tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using
             ClientID,                                        where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and `EXPLAIN` produces this result:

```
table type   possible_keys key     key_len ref           rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                        where
             ActualPC
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC   1
et_1  eq_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY 15      tt.ClientID   1
```

The `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. You should check whether the numbers are even close to the truth by comparing the `rows` product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause.

It is possible in some cases to execute statements that modify data when `EXPLAIN SELECT` is used with a subquery; for more information, see Section 13.2.9.8, “Subqueries in the FROM Clause”.

8.8.3 EXPLAIN EXTENDED Output Format

When `EXPLAIN` is used with the `EXTENDED` keyword, the output includes a `filtered` column not otherwise displayed. This column indicates the estimated percentage of table rows that will be filtered by the table condition. In addition, the statement produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. The `Message` value in `SHOW WARNINGS` output displays how the optimizer qualifies table and column names in the `SELECT` statement, what the `SELECT` looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process.

Here is an example of extended output:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> EXPLAIN EXTENDED
    -> SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 4
        Extra: Using index
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: a
          key: a
      key_len: 5
          ref: func
         rows: 2
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select `test`.`t1`.`a` AS `a`,
         <in_optimizer>(`test`.`t1`.`a`,
         <exists>(<index_lookup>(<cache>(`test`.`t1`.`a`)
         in t2 on a checking NULL having
         <is_not_null_test>(`test`.`t2`.`a`)))) AS `t1.a
         IN (SELECT t2.a FROM t2)` from `test`.`t1`
1 row in set (0.00 sec)
```

Because the statement displayed by `SHOW WARNINGS` may contain special markers to provide information about query rewriting or optimizer actions, the statement is not necessarily valid SQL and is not intended to be executed. The output may also include rows with `Message` values that provide additional non-SQL explanatory notes about actions taken by the optimizer.

The following list describes special markers that can appear in `EXTENDED` output displayed by `SHOW WARNINGS`:

- `<cache>(`*`expr`*`)`

  The expression (such as a scalar subquery) is executed once and the resulting value is saved in memory for later use.

- `<exists>(`*`query fragment`*`)`

  The subquery predicate is converted to an `EXISTS` predicate and the subquery is transformed so that it can be used together with the `EXISTS` predicate.

- `<in_optimizer>(`*`query fragment`*`)`

  This is an internal optimizer object with no user significance.

- `<index_lookup>(`*`query fragment`*`)`

  The query fragment is processed using an index lookup to find qualifying rows.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `<is_not_null_test>(expr)`

  A test to verify that the expression does not evaluate to `NULL`.

- `<primary_index_lookup>(query fragment)`

  The query fragment is processed using a primary key lookup to find qualifying rows.

- `<ref_null_helper>(expr)`

  This is an internal optimizer object with no user significance.

When some tables are of `const` or `system` type, expressions involving columns from these tables are evaluated early by the optimizer and are not part of the displayed statement. However, with `FORMAT=JSON`, some `const` table accesses are displayed as a `ref` access that uses a const value.

8.8.4 Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: `log(row_count) / log(index_block_length / 3 * 2 / (index_length + data_pointer_length)) + 1`.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with a key value length of three bytes (the size of `MEDIUMINT`), the formula indicates `log(500,000)/log(1024/3*2/(3+4)) + 1` = `4` seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2MB (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

The preceding discussion does not mean that your application performance slowly degenerates by log *N*. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by log *N*). To avoid this, increase the key cache size as the data grows. For `MyISAM` tables, the key cache size is controlled by the `key_buffer_size` system variable. See Section 8.12.2, “Tuning Server Parameters”.

8.9 Controlling the Query Optimizer

MySQL provides optimizer control through system variables that affect how query plans are evaluated index hints.

8.9.1 Controlling Query Plan Evaluation

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7 to 10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server's performance.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL 5.0.1 introduces a more flexible method for query optimization that enables the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled using two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of "educated guess" rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.

- The `optimizer_search_depth` variable tells how far into the "future" of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if `optimizer_search_depth` is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth`, this variable can be set to 0 to tell the optimizer to determine the value automatically.

8.9.2 Index Hints

Index hints give the optimizer information about how to choose indexes during query processing. Index hints are specified following a table name. (For the general syntax for specifying tables in a `SELECT` statement, see Section 13.2.8.2, "JOIN Syntax".) The syntax for referring to an individual table, including index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint]

index_hint:
    USE {INDEX|KEY} [FOR JOIN] (index_list)
  | IGNORE {INDEX|KEY} [FOR JOIN] (index_list)
  | FORCE {INDEX|KEY} [FOR JOIN] (index_list)

index_list:
    index_name [, index_name] ...
```

The `USE INDEX (index_list)` hint tells MySQL to use only one of the named indexes to find rows in the table. The alternative syntax `IGNORE INDEX (index_list)` tells MySQL to not use some particular index or indexes. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

The `FORCE INDEX` hint acts like `USE INDEX (index_list)`, with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the named indexes to find rows in the table.

Each hint requires the names of *indexes*, not the names of columns. To refer to a primary key, use the name `PRIMARY`. To see the index names for a table, use `SHOW INDEX`.

An *`index_name`* value need not be a full index name. It can be an unambiguous prefix of an index name. If a prefix is ambiguous, an error occurs.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`USE INDEX`, `IGNORE INDEX`, and `FORCE INDEX` affect only which indexes are used when MySQL decides how to find rows in the table and how to do the join. They do not affect whether an index is used when resolving an `ORDER BY` or `GROUP BY` clause. As of MySQL 5.0.40, the optional `FOR JOIN` clause can be added to make this explicit.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

For `FULLTEXT` searches, index hints do not work before MySQL 5.0.74. As of 5.0.74, index hints work as follows:

- For natural language mode searches, index hints are silently ignored. For example, `IGNORE INDEX(i1)` is ignored with no warning and the index is still used.
- For boolean mode searches, index hints are honored.

8.10 Buffering and Caching

MySQL uses several strategies that cache information in memory buffers to increase performance.

8.10.1 The MyISAM Key Cache

To minimize disk I/O, the `MyISAM` storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system file system cache.

This section first describes the basic operation of the `MyISAM` key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Access to the key cache no longer is serialized among threads. Multiple sessions can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native file system buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the `MyISAM` index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are nonleaf nodes.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains all used blocks in a special list (*LRU chain*) ordered by time of use. When a block is accessed, it is the most recently used and is placed at the end of the list. When blocks need to be replaced, blocks at the beginning of the list are the least recently used and become the first candidates for eviction.

The `InnoDB` storage engine also uses an LRU algorithm, to manage its buffer pool. See Section 8.10.2, “The InnoDB Buffer Pool”.

8.10.1.1 Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple sessions.
- A buffer that is being updated causes sessions that need to use it to wait until the update is complete.
- Multiple sessions can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

8.10.1.2 Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among sessions entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given `MyISAM` table. By default, all `MyISAM` table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement (see Section 13.7.6.1, “CACHE INDEX Syntax”). For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+---------+--------------------+----------+----------+
```

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

You cannot destroy the default key cache. Any attempt to do this is ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;

mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8384512 |
+-----------------+---------+
```

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` is the cache component. See Section 5.1.5.1, “Structured System Variables”, for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, you can use a strategy that involves three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.
- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.
- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to nonleaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `CACHE INDEX` statement sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init-file` option that names a file containing `CACHE INDEX` statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in `mysqld_init.sql` are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

8.10.1.3 Midpoint Insertion Strategy

By default, the key cache management system uses a simple LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy.*

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sublist and a warm sublist. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not “too short,” always containing at least `key_cache_division_limit` percent of the key cache blocks. `key_cache_division_limit` is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sublist. After a certain number of hits (accesses of the block), it is promoted to the hot sublist. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sublist is placed at the end of the list. The block then circulates within this sublist. If the block stays at the beginning of the sublist for a long enough time, it is demoted to the warm sublist. This time is determined by the value of the `key_cache_age_threshold` component of the key cache.

The threshold value prescribes that, for a key cache containing *`N`* blocks, the block at the beginning of the hot sublist not accessed within the last *`N`* `* key_cache_age_threshold / 100` hits is to be moved to the beginning of the warm sublist. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sublist.

The midpoint insertion strategy enables you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the `key_cache_division_limit` value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the `key_cache_division_limit` set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sublist during an index scan operation as well.

8.10.1.4 Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its nonleaf nodes, it makes sense to preload the key cache with index blocks before

This documentation is for an older version. If you're
This documentation is for an older version. If you're

starting to use it. Preloading enables you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the nonleaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

8.10.1.5 Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the `key_cache_block_size` variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

To control the size of blocks in the `.MYI` index file of `MyISAM` tables, use the `--myisam-block-size` option at server startup.

8.10.1.6 Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` or `key_cache_block_size` key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native file system caching. File system caching is not as efficient as using a key cache, so although queries execute,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of file system caching for the indexes ceases.

8.10.2 The InnoDB Buffer Pool

`InnoDB` maintains a buffer pool for caching data and indexes in memory. `InnoDB` manages the pool as a list, using a least recently used (LRU) algorithm incorporating a midpoint insertion strategy. When room is needed to add a new block to the pool, `InnoDB` evicts the least recently used block and adds the new block to the middle of the list. The midpoint insertion strategy in effect causes the list to be treated as two sublists:

- At the head, a sublist of “new” (or “young”) blocks that have been recently used.
- At the tail, a sublist of “old” blocks that are less recently used.

As a result of the algorithm, the new sublist contains blocks that are heavily used by queries. The old sublist contains less-used blocks, and candidates for eviction are taken from this sublist.

The LRU algorithm operates as follows by default:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- When `InnoDB` reads a block into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A block can be read in as a result of two types of read requests: Because it is required (for example, to satisfy query execution), or as part of read-ahead performed in anticipation that it will be required.
- The first access to a block in the old sublist makes it “young”, causing it to move to the head of the buffer pool (the head of the new sublist). If the block was read in because it was required, the first access occurs immediately and the block is made young. If the block was read in due to read-ahead, the first access does not occur immediately (and might not occur at all before the block is evicted).
- As long as no accesses occur for a block in the pool, it “ages” by moving toward the tail of the list. Blocks in both the new and old sublists age as other blocks are made new. Blocks in the old sublist also age as blocks are inserted at the midpoint. Eventually, a block that remains unused for long enough reaches the tail of the old sublist and is evicted.

In the default operation of the buffer pool, a block when read in is loaded at the midpoint and then moved immediately to the head of the new sublist as soon as an access occurs. In the case of a table scan (such as performed for a `mysqldump` operation), each block read by the scan ends up moving to the head of the new sublist because multiple rows are accessed from each block. This occurs even for a one-time scan, where the blocks are not otherwise used by other queries. Blocks may also be loaded by the read-ahead background thread and then moved to the head of the new sublist by a single access. These effects can be disadvantageous because they push blocks that are in heavy use by other queries out of the new sublist to the old sublist where they become subject to eviction.

The `innodb_buffer_pool_size` system variable specifies the size of the buffer pool. If your buffer pool is small and you have sufficient memory, making the pool larger can improve performance by reducing the amount of disk I/O needed as queries access `InnoDB` tables.

The `MyISAM` storage engine also uses an LRU algorithm, to manage its key cache. See Section 8.10.1, “The MyISAM Key Cache”.

8.10.3 The MySQL Query Cache

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.

The query cache can be useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content.

The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

Note

The query cache does not work in an environment where you have multiple `mysqld` servers updating the same `MyISAM` tables.

Note

The query cache is not used for prepared statements. If you are using prepared statements, consider that these statements will not be satisfied by the query cache. See Section 20.6.8, “C API Prepared Statements”.

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2×500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` system variable to 0. By disabling the query cache code, there is no noticeable overhead. If you build MySQL from source, query cache capabilities can be excluded from the server entirely by invoking `configure` with the `--without-query-cache` option.

The query cache offers the potential for substantial performance improvement, but you should not assume that it will do so under all circumstances. With some query cache configurations or server workloads, you might actually see a performance decrease:

- Be cautious about sizing the query cache excessively large, which increases the overhead required to maintain the cache, possibly beyond the benefit of enabling it. Sizes in tens of megabytes are usually beneficial. Sizes in the hundreds of megabytes might not be.
- Server workload has a significant effect on query cache efficiency. A query mix consisting almost entirely of a fixed set of `SELECT` statements is much more likely to benefit from enabling the cache than a mix in which frequent `INSERT` statements cause continual invalidation of results in the cache. In some cases, a workaround is to use the `SQL_NO_CACHE` option to prevent results from even entering the cache for `SELECT` statements that use frequently modified tables. (See Section 8.10.3.2, “Query Cache SELECT Options”.)

To verify that enabling the query cache is beneficial, test the operation of your MySQL server with the cache enabled and disabled. Then retest periodically because query cache efficiency may change as server workload changes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

8.10.3.1 How the Query Cache Operates

This section describes how the query cache works when it is operational. Section 8.10.3.3, “Query Cache Configuration”, describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

Because comparison of a query against those in the cache occurs before parsing, the cache is not used for queries of the following types:

- Prepared statements
- Queries that are a subquery of an outer query
- Queries executed within the body of a stored function or trigger

Before a query result is fetched from the query cache, MySQL checks whether the user has `SELECT` privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See Section 8.10.3.4, “Query Cache Status and Maintenance”.

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

The query cache also works within transactions when using `InnoDB` tables.

The result from a `SELECT` query on a view is cached.

Before MySQL 5.0, a query that began with a leading comment could be cached, but could not be fetched from the cache. This problem is fixed in MySQL 5.0.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS ...` queries and stores a value that is returned by a following `SELECT FOUND_ROWS()` query. `FOUND_ROWS()` returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache. The `SELECT FOUND_ROWS()` query itself cannot be cached.

A query cannot be cached if it contains any of the functions shown in the following table.

`BENCHMARK()`	`CONNECTION_ID()`	`CONVERT_TZ()`
`CURDATE()`	`CURRENT_DATE()`	`CURRENT_TIME()`
`CURRENT_TIMESTAMP()`	`CURRENT_USER()`	`CURTIME()`
`DATABASE()`	`ENCRYPT()` with one parameter	`FOUND_ROWS()`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`GET_LOCK()`	`IS_FREE_LOCK()`	`IS_USED_LOCK()`
`LAST_INSERT_ID()`	`LOAD_FILE()`	`MASTER_POS_WAIT()`
`NOW()`	`RAND()`	`RELEASE_ALL_LOCKS()`
`RELEASE_LOCK()`	`SLEEP()`	`SYSDATE()`
`UNIX_TIMESTAMP()` with no parameters	`USER()`	`UUID()`

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs) or stored functions.
- It refers to user variables or local stored program variables.
- It refers to tables in the `mysql` or `INFORMATION_SCHEMA` system database.
- It is of any of the following forms:

```
SELECT ... LOCK IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

  The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the Connector/ODBC section of Chapter 20, *Connectors and APIs*.

  Statements within transactions that use `SERIALIZABLE` isolation level also cannot be cached because they use `LOCK IN SHARE MODE` locking.

- It was issued as a prepared statement, even if no placeholders were employed. For example, the query used here is not cached:

```
char *my_sql_stmt = "SELECT a, b FROM table_c";
/* ... */
mysql_stmt_prepare(stmt, my_sql_stmt, strlen(my_sql_stmt));
```

  See Section 20.6.8, “C API Prepared Statements”.

- It uses `TEMPORARY` tables.
- It does not use any tables.
- It generates warnings.
- The user has a column-level privilege for any of the involved tables.

8.10.3.2 Query Cache SELECT Options

Two query cache-related options may be specified in `SELECT` statements:

- `SQL_CACHE`

  The query result is cached if it is cacheable and the value of the `query_cache_type` system variable is `ON` or `DEMAND`.

-

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`SQL_NO_CACHE`

The server does not use the query cache. It neither checks the query cache to see whether the result is already cached, nor does it cache the query result. (Due to a limitation in the parser, a space character must precede and follow the `SQL_NO_CACHE` keyword; a nonspace such as a newline causes the server to check the query cache to see whether the result is already cached.)

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

8.10.3.3 Query Cache Configuration

The `have_query_cache` server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

When using a standard MySQL binary, this value is always `YES`, even if query caching is disabled.

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting `mysqld`. The query cache system variables all have names that begin with `query_cache_`. They are described briefly in Section 5.1.4, “Server System Variables”, with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` system variable. Setting it to 0 disables the query cache. The default size is 0, so the query cache is disabled by default.

Be careful not to set the size of the cache too large. Due to the need for threads to lock the cache during updates, you may see lock contention issues with a very large cache.

Note

When using the Windows Configuration Wizard to install or configure MySQL, the default value for `query_cache_size` will be configured automatically for you based on the different configuration types available. When using the Windows Configuration Wizard, the query cache may be enabled (that is, set to a nonzero value) due to the selected configuration. The query cache is also controlled by the setting of the `query_cache_type` variable. You should check the values of these variables as set in your `my.ini` file after configuration has taken place.

When you set `query_cache_size` to a nonzero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
   Code: 1282
Message: Query cache failed to set size 39936;
         new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 41984 |
+------------------+-------+
```

For the query cache to actually be able to hold any query results, its size must be set larger:

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| query_cache_size | 999424 |
+------------------+--------+
1 row in set (0.00 sec)
```

The `query_cache_size` value is aligned to the nearest 1024 byte block. The value reported may therefore be different from the value that you assign.

If the query cache size is greater than 0, the `query_cache_type` variable influences how it works. This variable can be set to the following values:

- A value of `0` or `OFF` prevents caching or retrieval of cached results.
- A value of `1` or `ON` enables caching except of those statements that begin with `SELECT SQL_NO_CACHE`.
- A value of `2` or `DEMAND` causes caching of only those statements that begin with `SELECT SQL_CACHE`.

Setting the `GLOBAL query_cache_type` value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the `SESSION query_cache_type` value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

If you set `query_cache_type` at server startup (rather than at runtime with a `SET` statement), only the numeric values are permitted.

To control the maximum size of individual query results that can be cached, set the `query_cache_limit` system variable. The default value is 1MB.

When a query is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the `query_cache_min_res_unit` system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of `query_cache_min_res_unit`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The default value of `query_cache_min_res_unit` is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, you should decrease the value of `query_cache_min_res_unit`. The number of free blocks and queries removed due to pruning are given by the values of the `Qcache_free_blocks` and `Qcache_lowmem_prunes` status variables.
- If most of your queries have large results (check the `Qcache_total_blocks` and `Qcache_queries_in_cache` status variables), you can increase performance by increasing `query_cache_min_res_unit`. However, be careful to not make it too large (see the previous item).

8.10.3.4 Query Cache Status and Maintenance

To check whether the query cache is present in your MySQL server, use the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The `RESET QUERY CACHE` statement removes all query results from the query cache. The `FLUSH TABLES` statement also does this.

To monitor query cache performance, use `SHOW STATUS` to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| Qcache_free_blocks      | 36     |
| Qcache_free_memory      | 138488 |
| Qcache_hits             | 79570  |
| Qcache_inserts          | 27087  |
| Qcache_lowmem_prunes    | 3114   |
| Qcache_not_cached       | 22989  |
| Qcache_queries_in_cache | 415    |
| Qcache_total_blocks     | 912    |
+-------------------------+--------+
```

Descriptions of each of these variables are given in Section 5.1.6, “Server Status Variables”. Some uses for them are described here.

The total number of `SELECT` queries is given by this formula:

```
  Com_select
+ Qcache_hits
+ queries with errors found by parser
```

The `Com_select` value is given by this formula:

```
  Qcache_inserts
+ Qcache_not_cached
```


This documentation is for an older version. If you're

```
+ queries with errors found during the column-privileges check
```

The query cache uses variable-length blocks, so `Qcache_total_blocks` and `Qcache_free_blocks` may indicate query cache memory fragmentation. After `FLUSH QUERY CACHE`, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the `Qcache_lowmem_prunes` status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in Section 8.10.3.3, “Query Cache Configuration”.

8.11 Optimizing Locking Operations

When your database is busy with multiple sessions reading and writing data, the mechanism that controls access to data files and memory areas can become a consideration for performance tuning. Otherwise, sessions can spend time waiting for access to resources when they could be running concurrently.

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. See Section 8.11.1, “Internal Locking Methods”.
- External locking occurs when the server and other programs lock `MyISAM` table files to coordinate among themselves which program can access the tables at which time. See Section 8.11.4, “External Locking”.

8.11.1 Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server itself to manage contention for table contents by multiple sessions. This type of locking is internal because it is performed entirely by the server and involves no other programs. External locking occurs when the server and other programs lock `MyISAM` table files to coordinate among themselves which program can access the tables at which time. See Section 8.11.4, “External Locking”.

MySQL uses table-level locking for `MyISAM`, `MEMORY` and `MERGE` tables, page-level locking for `BDB` tables, and row-level locking for `InnoDB` tables.

In many cases, you can make an educated guess about which locking type is best for an application, but generally it is difficult to say that a given lock type is better than another. Everything depends on the application and different parts of an application may require different lock types.

To decide whether you want to use a storage engine with row-level locking, you should look at what your application does and what mix of select and update statements it uses. For example, most Web applications perform many selects, relatively few deletes, updates based mainly on key values, and inserts into a few specific tables. The base MySQL `MyISAM` setup is very well tuned for this.

Table locking in MySQL is deadlock-free for storage engines that use table-level locking. Deadlock avoidance is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

MySQL grants table write locks as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

1. If there are no locks on the table, put a write lock on it.

2. Otherwise, put the lock request in the write lock queue.

MySQL grants table read locks as follows:

1. If there are no write locks on the table, put a read lock on it.

2. Otherwise, put the lock request in the read lock queue.

Table updates are given higher priority than table retrievals. Therefore, when a lock is released, the lock is made available to the requests in the write lock queue and then to the requests in the read lock queue. This ensures that updates to a table are not "starved" even when there is heavy `SELECT` activity for the table. However, if there are many updates for a table, `SELECT` statements wait until there are no more updates.

For information on altering the priority of reads and writes, see Section 8.11.2, "Table Locking Issues".

You can analyze the table lock contention on your system by checking the `Table_locks_immediate` and `Table_locks_waited` status variables, which indicate the number of times that requests for table locks could be granted immediately and the number that had to wait, respectively:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent `INSERT` and `SELECT` statements for a `MyISAM` table without locks. That is, you can insert rows into a `MyISAM` table at the same time other clients are reading from it. Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are enabled again automatically when all holes have been filled with new data. This behavior is altered by the `concurrent_insert` system variable. See Section 8.11.3, "Concurrent Inserts".

If you acquire a table lock explicitly with `LOCK TABLES`, you can request a `READ LOCAL` lock rather than a `READ` lock to enable other sessions to perform concurrent inserts while you have the table locked.

To perform many `INSERT` and `SELECT` operations on a table `t1` when concurrent inserts are not possible, you can insert rows into a temporary table `temp_t1` and update the real table with the rows from the temporary table:

```
mysql> LOCK TABLES t1 WRITE, temp_t1 WRITE;
mysql> INSERT INTO t1 SELECT * FROM temp_t1;
mysql> DELETE FROM temp_t1;
mysql> UNLOCK TABLES;
```

`InnoDB` uses row locks and `BDB` uses page locks. Deadlocks are possible for these storage engines because they automatically acquire locks during the processing of SQL statements, not at the start of the transaction.

Advantages of row-level locking:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Fewer lock conflicts when different sessions access different rows
- Fewer changes for rollbacks
- Possible to lock a single row for a long time

Disadvantages of row-level locking:

- Requires more memory than page-level or table-level locks
- Slower than page-level or table-level locks when used on a large part of the table because you must acquire many more locks
- Slower than other locks if you often do `GROUP BY` operations on a large part of the data or if you must scan the entire table frequently

Generally, table locks are superior to page-level or row-level locks in the following cases:

- Most statements for the table are reads
- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` or `DELETE` statements
- Many scans or `GROUP BY` operations on the entire table without any writers

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level or page-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”
- Copy on demand is in many cases superior to page-level or row-level locking. However, in the worst case, it can use much more memory than using normal locks.
- Instead of using row-level locks, you can employ application-level locks, such as those provided by `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only with applications that cooperate with each other. See Section 12.15, “Miscellaneous Functions”.

8.11.2 Table Locking Issues

To achieve a very high lock speed, MySQL uses table locking (instead of page, row, or column locking) for all storage engines except `InnoDB`, `BDB`, and `NDB`.

For `InnoDB` and `BDB` tables, MySQL uses table locking only if you explicitly lock the table with `LOCK TABLES`. For these storage engines, avoid using `LOCK TABLES` at all, because `InnoDB` uses automatic row-level locking and `BDB` uses page-level locking to ensure transaction isolation.

For large tables, table locking is often better than row locking, but there are some disadvantages:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Table locking enables many sessions to read from a table at the same time, but if a session wants to write to a table, it must first get exclusive access. During the update, all other sessions that want to access this particular table must wait until the update is done.
- Table locking causes problems in cases such as when a session is waiting because the disk is full and free space needs to become available before the session can proceed. In this case, all sessions that want to access the problem table are also put in a waiting state until more disk space is made available.

Table locking is also disadvantageous under the following scenario:

- A session issues a `SELECT` that takes a long time to run.
- Another session then issues an `UPDATE` on the same table. This session waits until the `SELECT` is finished.
- Another session issues another `SELECT` statement on the same table. Because `UPDATE` has higher priority than `SELECT`, this `SELECT` waits for the `UPDATE` to finish, *after* waiting for the first `SELECT` to finish.

The following items describe some ways to avoid or reduce contention caused by table locking:

- Try to get the `SELECT` statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates`. For storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`), this gives all statements that update (modify) a table lower priority than `SELECT` statements. In this case, the second `SELECT` statement in the preceding scenario would execute before the `UPDATE` statement, and would not need to wait for the first `SELECT` to finish.
- To specify that all updates issued in a specific connection should be done with low priority, set the `low_priority_updates` server system variable equal to 1.
- To give a specific `INSERT`, `UPDATE`, or `DELETE` statement lower priority, use the `LOW_PRIORITY` attribute.
- To give a specific `SELECT` statement higher priority, use the `HIGH_PRIORITY` attribute. See Section 13.2.8, “SELECT Syntax”.
- Start `mysqld` with a low value for the `max_write_lock_count` system variable to force MySQL to temporarily elevate the priority of all `SELECT` statements that are waiting for a table after a specific number of inserts to the table occur. This permits `READ` locks after a certain number of `WRITE` locks.
- If you have problems with `INSERT` combined with `SELECT`, consider switching to `MyISAM` tables, which support concurrent `SELECT` and `INSERT` statements. (See Section 8.11.3, “Concurrent Inserts”.)
- If you mix inserts and deletes on the same table, `INSERT DELAYED` may be of great help. See Section 13.2.5.2, “INSERT DELAYED Syntax”.
- If you have problems with mixed `SELECT` and `DELETE` statements, the `LIMIT` option to `DELETE` may help. See Section 13.2.2, “DELETE Syntax”.
- Using `SQL_BUFFER_RESULT` with `SELECT` statements can help to make the duration of table locks shorter. See Section 13.2.8, “SELECT Syntax”.
- You could change the locking code in `mysys/thr_lock.c` to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

Here are some tips concerning table locks in MySQL:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Concurrent users are not a problem if you do not mix updates with selects that need to examine many rows in the same table.
- You can use `LOCK TABLES` to increase speed, because many updates within a single lock is much faster than updating without locks. Splitting table contents into separate tables may also help.
- If you encounter speed problems with table locks in MySQL, you may be able to improve performance by converting some of your tables to `InnoDB` or `BDB` tables. See Section 14.2, “The InnoDB Storage Engine”, and Section 14.5, “The BDB (BerkeleyDB) Storage Engine”.

8.11.3 Concurrent Inserts

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no holes in the data file (deleted rows in the middle), an `INSERT` statement can be executed to add rows to the end of the table at the same time that `SELECT` statements are reading rows from the table. If there are multiple `INSERT` statements, they are queued and performed in sequence, concurrently with the `SELECT` statements. The results of a concurrent `INSERT` may not be visible immediately.

The `concurrent_insert` system variable can be set to modify the concurrent-insert processing. By default, the variable is set to 1 and concurrent inserts are handled as just described. If `concurrent_insert` is set to 0, concurrent inserts are disabled. If the variable is set to 2, concurrent inserts at the end of the table are permitted even for tables that have deleted rows. See also the description of the `concurrent_insert` system variable.

Under circumstances where concurrent inserts can be used, there is seldom any need to use the `DELAYED` modifier for `INSERT` statements. See Section 13.2.5.2, “INSERT DELAYED Syntax”.

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See Section 5.4.3, “The Binary Log”. In addition, for those statements a read lock is placed on the selected-from table such that inserts into that table are blocked. The effect is that concurrent inserts for that table must wait as well.

With `LOAD DATA INFILE`, if you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other sessions can retrieve data from the table while `LOAD DATA` is executing. Use of the `CONCURRENT` option affects the performance of `LOAD DATA` a bit, even if no other session is using the table at the same time.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used.

For `LOCK TABLE`, the difference between `READ LOCAL` and `READ` is that `READ LOCAL` permits nonconflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

8.11.4 External Locking

External locking is the use of file system locking to manage contention for `MyISAM` database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If you use `myisamchk` to perform table maintenance operations on `MyISAM` tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with `myisamchk` for access to the tables. The same is true for use of `myisampack` to pack `MyISAM` tables.

  If the server is run with external locking enabled, you can use `myisamchk` at any time for read operations such a checking tables. In this case, if the server tries to update a table that `myisamchk` is using, the server will wait for `myisamchk` to finish before it continues.

  If you use `myisamchk` for write operations such as repairing or optimizing tables, or if you use `myisampack` to pack tables, you *must* always ensure that the `mysqld` server is not using the table. If you do not stop `mysqld`, you should at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

With external locking in effect, each process that requires access to a table acquires a file system lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as `myisamchk` need to modify tables while the server is running. If you only *read* tables with other programs, external locking is not required, although `myisamchk` might report warnings if the server changes tables while `myisamchk` is reading them.

With external locking disabled, to use `myisamchk`, you must either stop the server while `myisamchk` executes or else lock and flush the tables before running `myisamchk`. (See Section 8.12.1, “System Factors and Startup Parameter Tuning”.) To avoid this requirement, use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables.

For `mysqld`, external locking is controlled by the value of the `skip_external_locking` system variable. When this variable is enabled, external locking is disabled, and vice versa. External locking is disabled by default.

Use of external locking can be controlled at server startup by using the `--external-locking` or `--skip-external-locking` option.

If you do use external locking option to enable updates to `MyISAM` tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- You should not use the query cache for queries that use tables that are updated by another process.
- You should not start the server with the `--delay-key-write=ALL` option or use the `DELAY_KEY_WRITE=1` table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy these conditions is to always use `--external-locking` together with `--delay-key-write=OFF` and `--query-cache-size=0`. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

8.12 Optimizing the MySQL Server

8.12.1 System Factors and Startup Parameter Tuning

This documentation is for an older version. If you're
This documentation is for an older version. If you're

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing factors that apply at this level.

Before using MySQL in production, we advise you to test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Avoid external locking for `MyISAM` tables. The default is for external locking to be disabled. The `--external-locking` and `--skip-external-locking` options explicitly enable and disable external locking.

  Disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run `myisamchk`. On some systems it is mandatory to disable external locking because it does not work, anyway.

  The only case in which you cannot disable external locking is when you run multiple MySQL *servers* (not clients) on the same data, or if you run `myisamchk` to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using MySQL Cluster.

  The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

8.12.2 Tuning Server Parameters

You can determine the default buffer sizes used by the `mysqld` server using this command:

```
shell> mysqld --verbose --help
```

This command produces a list of all `mysqld` options and configurable system variables. The output includes the default variable values and looks something like this:

```
help                            TRUE
abort-slave-event-count         0
allow-suspicious-udfs           FALSE
auto-increment-increment        1
auto-increment-offset           1
automatic-sp-privileges         TRUE
basedir                         /home/jon/bin/mysql-5.0/
...
tmpdir                          (No default value)
transaction_alloc_block_size    8192
transaction_prealloc_size       4096
updatable_views_with_limit      1
use-symbolic-links              TRUE
verbose                         TRUE
wait_timeout                    28800
warnings                        1
```

For a `mysqld` server that is currently running, you can see the current values of its system variables by connecting to it and issuing this statement:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using `mysqladmin`:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all system and status variables, see Section 5.1.4, “Server System Variables”, and Section 5.1.6, “Server Status Variables”.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally better performance results from giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` and `table_cache`. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations.

- If you have at least 1-2GB of memory and many tables and want maximum performance with a moderate number of clients, use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=384M --table_open_cache=4000 \
             --sort_buffer_size=4M --read_buffer_size=1M &
  ```

- If you have only 256MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=64M --sort_buffer_size=1M
  ```

  If there are very many simultaneous connections, swapping problems may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections.

- With little memory and lots of connections, use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
             --read_buffer_size=100K &
  ```

  Or even this:

  ```
  shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
             --table_cache=32 --read_buffer_size=8K \
             --net_buffer_length=1K &
  ```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, increase the value of `read_rnd_buffer_size` to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see Section 5.1.2, “Server Configuration Defaults”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this:

```
shell> mysqld --key_buffer_size=128M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` and `--help` options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the `InnoDB` storage engine, see Section 8.6, “Optimizing for InnoDB Tables”.

8.12.3 Optimizing Disk I/O

This section describes ways to configure storage devices when you can devote more and faster storage hardware to the database server.

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:
  - Using symbolic links

    This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See Section 8.12.4, “Using Symbolic Links”.
  - Striping

    Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the *`N`*-th block on the (*`N MOD number_of_disks`*) disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See Section 8.13.3, “Using Your Own Benchmarks”.

    The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.
- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need 2 × *`N`* drives to hold *`N`* drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.
- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID *`N`* disk. RAID *`N`* can be a problem if you have many writes, due to the time required to update the parity bits.
- On Linux, you can get much better performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
hdparm -m 16 -d 1
```

Performance and reliability when using this command depend on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, file system corruption may result, so back up everything before experimenting!

- You can also set the parameters for the file system that the database uses:

  If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your file systems with the `-o noatime` option. That skips updates to the last access time in inodes on the file system, which avoids some disk seeks.

  On many operating systems, you can set a file system to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you better performance without sacrificing too much reliability. (This flag is on by default on Linux.)

8.12.4 Using Symbolic Links

You can move databases or tables from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disks.

The recommended way to do this is to symlink entire database directories to a different disk. Symlink `MyISAM` tables only as a last resort.

To determine the location of your data directory, use this statement:

```
SHOW VARIABLES LIKE 'datadir';
```

8.12.4.1 Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a soft link to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, or any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

8.12.4.2 Using Symbolic Links for MyISAM Tables on Unix

Symlinks are fully supported only for `MyISAM` tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Do not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). To determine whether your system supports symbolic links, check the value of the `have_symlink` system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

The handling of symbolic links for `MyISAM` tables works as follows:

- In the data directory, you always have the table format (`.frm`) file, the data (`.MYD`) file, and the index (`.MYI`) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.
- You can symlink the data file and the index file independently to different directories.
- To instruct a running MySQL server to perform the symlinking, use the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See Section 13.1.10, “CREATE TABLE Syntax”. Alternatively, if `mysqld` is not running, symlinking can be accomplished manually using `ln -s` from the command line.

  Note

  Beginning with MySQL 5.0.60, the path used with either or both of the `DATA DIRECTORY` and `INDEX DIRECTORY` options may not include the MySQL `data` directory. (Bug #32167)

- `myisamchk` does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the `ALTER TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
- 

  Note

  When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason *not* to run `mysqld` as the system `root` or permit system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` or `RENAME TABLE` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you use `ALTER TABLE ... RENAME` or `RENAME TABLE` to move a table to another database, the table is moved to the other database directory. If the table name changed, the symlinks in the new database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you are not using symlinks, start `mysqld` with the `--skip-symbolic-links` option to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

These table symlink operations are not supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.
- `BACKUP TABLE` and `RESTORE TABLE` do not respect symbolic links.
- As indicated previously, only the data and index files can be symbolic links. The `.frm` file must *never* be a symbolic link. Attempting to do this (for example, to make one table name a synonym for another) produces incorrect results. Suppose that you have a database `db1` under the MySQL data directory, a table `tbl1` in this database, and in the `db1` directory you make a symlink `tbl2` that points to `tbl1`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> cd /path/to/datadir/db1
shell> ln -s tbl1.frm tbl2.frm
shell> ln -s tbl1.MYD tbl2.MYD
shell> ln -s tbl1.MYI tbl2.MYI
```

Problems result if one thread reads `db1.tbl1` and another thread updates `db1.tbl2`:

- The query cache is “fooled” (it has no way of knowing that `tbl1` has not been updated, so it returns outdated results).
- `ALTER` statements on `tbl2` fail.

8.12.4.3 Using Symbolic Links for Databases on Windows

On Windows, symbolic links can be used for database directories. This enables you to put a database directory at a different location (for example, on a different disk) by setting up a symbolic link to it. Use of database symlinks on Windows is similar to their use on Unix, although the procedure for setting up the link differs.

Suppose that you want to place the database directory for a database named `mydb` at `D:\data\mydb`. To do this, create a symbolic link in the MySQL data directory that points to `D:\data\mydb`. However, before creating the symbolic link, make sure that the `D:\data\mydb` directory exists by creating it if necessary. If you already have a database directory named `mydb` in the data directory, move it to `D:\data`. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.

On Windows, create a symbolic link to a MySQL database by creating a `.sym` file in the data directory that contains the path to the destination directory. The file should be named *`db_name`*`.sym`, where *`db_name`* is the database name.

Support for database symbolic links on Windows using `.sym` files is enabled by default. If you do not need `.sym` file symbolic links, you can disable support for them by starting `mysqld` with the `--skip-symbolic-links` option. To determine whether your system supports `.sym` file symbolic links, check the value of the `have_symlink` system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

To create a `.sym` file symlink, use this procedure:

1. Change location into the data directory:

   ```
   C:\> cd \path\to\datadir
   ```

2. In the data directory, create a text file named `mydb.sym` that contains this path name: `D:\data\mydb\`

Note

The path name to the new database and tables should be absolute. If you specify a relative path, the location will be relative to the `mydb.sym` file.

After this, all tables created in the database `mydb` are created in `D:\data\mydb`.

The following limitations apply to the use of `.sym` files for database symbolic linking on Windows:

This documentation is for an older version. If you're

- The symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.
- The `--innodb_file_per_table` option cannot be used.
- If you run `mysqld` as a service, you cannot use a mapped drive to a remote server as the destination of the symbolic link. As a workaround, you can use the full path (`\\servername\path\`).

8.12.5 Optimizing Memory Use

8.12.5.1 How MySQL Uses Memory

MySQL allocates buffers and caches to improve performance of database operations. You can improve MySQL performance by increasing the values of certain cache and buffer-related system variables. You can also modify these variables to run MySQL on systems with limited memory.

The following list describes some of the ways that MySQL uses memory. Where applicable, relevant system variables are referenced. Some items are storage engine specific.

- The `InnoDB` buffer pool is a memory area that holds cached `InnoDB` data for tables, indexes, and other auxiliary buffers. For efficiency of high-volume read operations, the buffer pool is divided into pages that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the LRU algorithm. For more information, see Section 8.10.2, “The InnoDB Buffer Pool”.

  The size of the buffer pool is important for system performance.

  - Typically, it is recommended that `innodb_buffer_pool_size` is configured to 50 to 75 percent of system memory.
  - `InnoDB` allocates memory for the entire buffer pool at server startup. Memory allocation is performed by `malloc()` operations. Buffer pool size is defined by the `innodb_buffer_pool_size` configuration option.
  - On systems with a large amount of memory, you can improve concurrency by dividing the buffer pool into multiple buffer pool instances. The number of buffer pool instances is controlled by `innodb_buffer_pool_instances`.
  - A buffer pool that is too small may cause excessive churning as pages are flushed from the buffer pool only to be required again a short time later.
  - A buffer pool that is too large may cause swapping due to competition for memory.
- All threads share the `MyISAM` key buffer; its size is determined by the `key_buffer_size` variable. Other buffers used by the server are allocated as needed. See Section 8.12.2, “Tuning Server Parameters”.

  For each `MyISAM` table that is opened, the index file is opened once; the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size `3 * N` are allocated (where *`N`* is the maximum row length, not counting `BLOB` columns). A `BLOB` column requires five to eight bytes plus the length of the `BLOB` data. The `MyISAM` storage engine maintains one extra row buffer for internal use.
- Each thread that is used to manage client connections uses some thread-specific space. The following list indicates these and which variables control their size:
  - A stack (variable `thread_stack`)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - A connection buffer (variable `net_buffer_length`)
  - A result buffer (variable `net_buffer_length`)

  The connection buffer and result buffer each begin with a size equal to `net_buffer_length` bytes, but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` bytes after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.

- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.
- Only compressed `MyISAM` tables are memory mapped. This is because the 32-bit memory space of 4GB is not large enough for most big tables. When systems with a 64-bit address space become more common, we may add general support for memory mapping.
- Each request that performs a sequential scan of a table allocates a *read buffer* (variable `read_buffer_size`).
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* (variable `read_rnd_buffer_size`) may be allocated to avoid disk seeks.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain `BLOB` columns are stored on disk.

  If an internal in-memory temporary table becomes too large, MySQL handles this automatically by changing the table from in-memory to on-disk format, to be handled by the `MyISAM` storage engine. You can increase the permissible temporary table size as described in Section 8.4.4, “Internal Temporary Table Use in MySQL”.
- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Section B.5.3.5, “Where MySQL Stores Temporary Files”.
- Almost all parsing and calculating is done in thread-local and reusable memory pools. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings.
- For each table having `BLOB` columns, a buffer is enlarged dynamically to read in larger `BLOB` values. If you scan a table, a buffer as large as the largest `BLOB` value is allocated.
- MySQL requires memory and descriptors for the table cache. Handler structures for all in-use tables are saved in the table cache and managed as “First In, First Out” (FIFO). The initial table cache size is defined by the `table_cache` system variable; see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.
- A `FLUSH TABLES` statement or `mysqladmin flush-tables` command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.
- The server caches information in memory as a result of `GRANT` and `CREATE USER` statements. This memory is not released by the corresponding `REVOKE` and `DROP USER` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused memory between stacks as used memory. To verify this, check available swap with `swap -s`. We test `mysqld` with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

8.12.5.2 Enabling Large Page Support

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

In MySQL, large pages can be used by InnoDB, to allocate memory for its buffer pool and additional memory pool.

MySQL supports only the Linux implementation of large page support (which is called HugeTLB in Linux).

Before large pages can be used on Linux, the kernel must be enabled to support them and it is necessary to configure the HugeTLB memory pool. For reference, the HugeTBL API is documented in the `Documentation/vm/hugetlbpage.txt` file of your Linux sources.

The kernel for some recent systems such as Red Hat Enterprise Linux appear to have the large pages feature enabled by default. To check whether this is true for your kernel, use the following command and look for output lines containing “huge”:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The nonempty command output indicates that large page support is present, but the zero values indicate that no pages are configured for use.

If your kernel needs to be reconfigured to support large pages, consult the `hugetlbpage.txt` file for instructions.

Assuming that your Linux kernel has large page support enabled, configure it for use by MySQL using the following commands. Normally, you put these in an `rc` file or equivalent startup file that is executed during the system boot sequence, so that the commands execute each time the system starts. The commands should execute early in the boot sequence, before the MySQL server starts. Be sure to change the allocation numbers and the group number as appropriate for your system.

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
echo 20 > /proc/sys/vm/nr_hugepages

# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.
echo 102 > /proc/sys/vm/hugetlb_shm_group

# Increase the amount of shmem permitted per segment
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
# (12G in this case).
echo 1560281088 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory.  The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.
echo 4194304 > /proc/sys/kernel/shmall
```

For MySQL usage, you normally want the value of `shmmax` to be close to the value of `shmall`.

To verify the large page configuration, check `/proc/meminfo` again as described previously. Now you should see some nonzero values:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The final step to make use of the `hugetlb_shm_group` is to give the `mysql` user an “unlimited” value for the memlock limit. This can by done either by editing `/etc/security/limits.conf` or by adding the following command to your `mysqld_safe` script:

```
ulimit -l unlimited
```

Adding the `ulimit` command to `mysqld_safe` causes the `root` user to set the memlock limit to `unlimited` before switching to the `mysql` user. (This assumes that `mysqld_safe` is started by `root`.)

Large page support in MySQL is disabled by default. To enable it, start the server with the `--large-pages` option. For example, you can use the following lines in your server's `my.cnf` file:

```
[mysqld]
large-pages
```

With this option, `InnoDB` uses large pages automatically for its buffer pool and additional memory pool. If `InnoDB` cannot do this, it falls back to use of traditional memory and writes a warning to the error log: `Warning: Using conventional memory pool`

To verify that large pages are being used, check `/proc/meminfo` again:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        2
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

8.12.6 Optimizing Network Use

8.12.6.1 How MySQL Uses Threads for Client Connections

Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In this connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See Section 5.1.4, “Server System Variables”, and Section 5.1.6, “Server Status Variables”.)

The thread cache has a size determined by the `thread_cache_size` system variable. The default value is 0 (no caching), which causes a thread to be set up for each new connection and disposed of when the connection terminates. Set `thread_cache_size` to *`N`* to enable *`N`* inactive connection threads to be cached. `thread_cache_size` can be set at server startup or changed while the server runs. A connection thread becomes inactive when the client connection with which it was associated terminates.

To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, monitor the `Threads_cached` and `Threads_created` status variables.

You can set `max_connections` at server startup or at runtime to control the maximum number of clients that can connect simultaneously.

When the thread stack is too small, this limits the complexity of the SQL statements which the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *`N`* bytes for each thread, start the server with `--thread_stack=`*`N`*.

8.12.6.2 DNS Lookup Optimization and the Host Cache

The MySQL server maintains a host cache in memory that contains information about clients: IP address, host name, and error information. The server uses this cache for nonlocal TCP connections. It does not use the cache for TCP connections established using the loopback interface address (`127.0.0.1`), or for connections established using a Unix socket file, named pipe, or shared memory.

For each new client connection, the server uses the client IP address to check whether the client host name is in the host cache. If not, the server attempts to resolve the host name. First, it resolves the IP address to a host name and resolves that host name back to an IP address. Then it compares the result to the original IP address to ensure that they are the same. The server stores information about the result of this operation in the host cache. If the cache is full, the least recently used entry is discarded.

The server performs host name resolution using the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls if the operating system supports them. Otherwise, the thread performing the lookup locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no

This documentation is for an older version. If you're
This documentation is for an older version. If you're

other thread can resolve host names that are not in the host cache until the thread holding the mutex lock releases it.

The server uses the host cache for several purposes:

- By caching the results of IP-to-host name lookups, the server avoids doing a DNS lookup for each client connection. Instead, for a given host, it needs to perform a lookup only for the first connection from that host.
- The cache contains information about errors that occur during the connection process. Some errors are considered “blocking.” If too many of these occur successively from a given host without a successful connection, the server blocks further connections from that host. The `max_connect_errors` system variable determines the number of permitted errors before blocking occurs. See Section B.5.2.6, “Host 'host_name' is blocked”.

To unblock blocked hosts, flush the host cache by issuing a `FLUSH HOSTS` statement or executing a `mysqladmin flush-hosts` command.

It is possible for a blocked host to become unblocked even without `FLUSH HOSTS` if activity from other hosts has occurred since the last connection attempt from the blocked host. This can occur because the server discards the least recently used cache entry to make room for a new entry if the cache is full when a connection arrives from a client IP not in the cache. If the discarded entry is for a blocked host, that host becomes unblocked.

The host cache is enabled by default. To disable it, start the server with the `--skip-host-cache` option.

To disable DNS host name lookups, start the server with the `--skip-name-resolve` option. In this case, the server uses only IP addresses and not host names to match connecting hosts to rows in the MySQL grant tables. Only accounts specified in those tables using IP addresses can be used. (Be sure that an account exists that specifies an IP address or you may not be able to connect.)

If you have a very slow DNS and many hosts, you might be able to improve performance either by disabling DNS lookups with `--skip-name-resolve` or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling the server

To disallow TCP/IP connections entirely, start the server with the `--skip-networking` option.

Some connection errors are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions).

8.13 Measuring Performance (Benchmarking)

To measure performance, consider the following factors:

- Whether you are measuring the speed of a single operation on a quiet system, or how a set of operations (a “workload”) works over a period of time. With simple tests, you usually test how changing one aspect (a configuration setting, the set of indexes on a table, the SQL clauses in a query) affects performance. Benchmarks are typically long-running and elaborate performance tests, where the results could dictate high-level choices such as hardware and storage configuration, or how soon to upgrade to a new MySQL version.
- For benchmarking, sometimes you must simulate a heavy database workload to get an accurate picture.
- Performance can vary depending on so many different factors that a difference of a few percentage points might not be a decisive victory. The results might shift the opposite way when you test in a different environment.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Certain MySQL features help or do not help performance depending on the workload. For completeness, always test performance with those features turned on and turned off. The two most important features to try with each workload are the MySQL query cache, and the adaptive hash index for `InnoDB` tables.

This section progresses from simple and direct measurement techniques that a single developer can do, to more complicated ones that require additional expertise to perform and interpret the results.

8.13.1 Measuring the Speed of Expressions and Functions

To measure the speed of a specific MySQL expression or function, invoke the `BENCHMARK()` function using the `mysql` client program. Its syntax is `BENCHMARK(loop_count,expression)`. The return value is always zero, but `mysql` prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

The built-in MySQL functions are typically highly optimized, but there may be some exceptions. `BENCHMARK()` is an excellent tool for finding out if some function is a problem for your queries.

8.13.2 The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from http://dev.mysql.com/downloads/, or use the current development source tree. (See Section 2.17.2, “Installing MySQL Using a Development Source Tree”.)
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See Section 2.22, “Perl Installation Notes”.

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

For more information about benchmark results, visit http://www.mysql.com/why-mysql/benchmarks/.

8.13.3 Using Your Own Benchmarks

Benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a "dummy" module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See Section 8.13.2, "The MySQL Benchmark Suite". You can take any program from this suite and modify it for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at http://osdb.sourceforge.net/.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, benchmark your whole application under the worst possible load. For example, you can try benchmarking packages such as SysBench and DBT2, available at https://launchpad.net/sysbench, and http://osdldbt.sourceforge.net/#dbt2. These packages can bring a system to its knees, so be sure to use them only on your development systems.

8.14 Examining Thread Information

When you are attempting to ascertain what your MySQL server is doing, it can be helpful to examine the process list, which is the set of threads currently executing within the server. Process list information is available from these sources:

- The `SHOW [FULL] PROCESSLIST` statement: Section 13.7.5.27, "SHOW PROCESSLIST Syntax"
- The `SHOW PROFILE` statement: Section 13.7.5.29, "SHOW PROFILES Syntax"
- The `mysqladmin processlist` command: Section 4.5.2, "`mysqladmin` — Client for Administering a MySQL Server"

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can always view information about your own threads. To view information about threads being executed for other accounts, you must have the `PROCESS` privilege.

Each process list entry contains several pieces of information:

- `Id` is the connection identifier for the client associated with the thread.
- `User` and `Host` indicate the account associated with the thread.
- `db` is the default database for the thread, or `NULL` if none is selected.
- `Command` and `State` indicate what the thread is doing.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.
- `Time` indicates how long the thread has been in its current state. The thread's notion of the current time may be altered in some cases: The thread can change the time with `SET TIMESTAMP = value`. For a thread running on a slave that is processing events from the master, the thread time is set to the time found in the events and thus reflects current time on the master and not the slave.
- `Info` contains the text of the statement being executed by the thread, or `NULL` if it is not executing one. By default, this value contains only the first 100 characters of the statement. To see the complete statements, use `SHOW FULL PROCESSLIST`.

The following sections list the possible `Command` values, and `State` values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided.

8.14.1 Thread Command Values

A thread can have any of the following `Command` values:

- `Binlog Dump`

  This is a thread on a master server for sending binary log contents to a slave server.
- `Change user`

  The thread is executing a change-user operation.
- `Close stmt`

  The thread is closing a prepared statement.
- `Connect`

  A replication slave is connected to its master.
- `Connect Out`

  A replication slave is connecting to its master.
- `Create DB`

  The thread is executing a create-database operation.
- `Daemon`

  This thread is internal to the server, not a thread that services a client connection.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Debug`

  The thread is generating debugging information.

- `Delayed insert`

  The thread is a delayed-insert handler.

- `Drop DB`

  The thread is executing a drop-database operation.

- `Error`

- `Execute`

  The thread is executing a prepared statement.

- `Fetch`

  The thread is fetching the results from executing a prepared statement.

- `Field List`

  The thread is retrieving information for table columns.

- `Init DB`

  The thread is selecting a default database.

- `Kill`

  The thread is killing another thread.

- `Long Data`

  The thread is retrieving long data in the result of executing a prepared statement.

- `Ping`

  The thread is handling a server-ping request.

- `Prepare`

  The thread is preparing a prepared statement.

- `Processlist`

  The thread is producing information about server threads.

- `Query`

  The thread is executing a statement.

- `Quit`

  The thread is terminating.

- `Refresh`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The thread is flushing table, logs, or caches, or resetting status variable or replication server information.

- `Register Slave`

  The thread is registering a slave server.

- `Reset stmt`

  The thread is resetting a prepared statement.

- `Set option`

  The thread is setting or resetting a client statement-execution option.

- `Shutdown`

  The thread is shutting down the server.

- `Sleep`

  The thread is waiting for the client to send a new statement to it.

- `Statistics`

  The thread is producing server-status information.

- `Table Dump`

  The thread is sending table contents to a slave server.

- `Time`

  Unused.

8.14.2 General Thread States

The following list describes thread `State` values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- `After create`

  This occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.

- `Analyzing`

  The thread is calculating a `MyISAM` table key distributions (for example, for `ANALYZE TABLE`).

- `checking permissions`

  The thread is checking whether the server has the required privileges to execute the statement.

- `Checking table`

  The thread is performing a table check operation.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `cleaning up`

  The thread has processed one command and is preparing to free memory and reset certain state variables.

- `closing tables`

  The thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, you should verify that you do not have a full disk and that the disk is not in very heavy use.

- `converting HEAP to MyISAM`

  The thread is converting an internal temporary table from a `MEMORY` table to an on-disk `MyISAM` table.

- `copy to tmp table`

  The thread is processing an `ALTER TABLE` statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

- `Copying to group table`

  If a statement has different `ORDER BY` and `GROUP BY` criteria, the rows are sorted by group and copied to a temporary table.

- `Copying to tmp table`

  The server is copying to a temporary table in memory.

- `Copying to tmp table on disk`

  The server is copying to a temporary table on disk. The temporary result set has become too large (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Consequently, the thread is changing the temporary table from in-memory to disk-based format to save memory.

- `Creating index`

  The thread is processing `ALTER TABLE ... ENABLE KEYS` for a `MyISAM` table.

- `Creating sort index`

  The thread is processing a `SELECT` that is resolved using an internal temporary table.

- `creating table`

  The thread is creating a table. This includes creation of temporary tables.

- `Creating tmp table`

  The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation will be `Copying to tmp table on disk`.

- `deleting from main table`

  The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving columns and offsets to be used for deleting from the other (reference) tables.

- `deleting from reference tables`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `discard_or_import_tablespace`

  The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

  This occurs at the end but before the cleanup of `ALTER TABLE`, `CREATE VIEW`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `executing`

  The thread has begun executing a statement.

- `Execution of init_command`

  The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

  The thread has executed a command. Some freeing of items done during this state involves the query cache. This state is usually followed by `cleaning up`.

- `Flushing tables`

  The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- `FULLTEXT initialization`

  The server is preparing to perform a natural-language full-text search.

- `init`

  This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions taken by the server in this state include flushing the binary log, the `InnoDB` log, and some query cache cleanup operations.

  For the `end` state, the following operations could be happening:

  - Removing query cache entries after data in a table is changed
  - Writing an event to the binary log
  - Freeing memory buffers, including for blobs

- `Killed`

  Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `Locked`

  The query is locked by another query.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `logging slow query`

  The thread is writing a statement to the slow-query log.

- `login`

  The initial state for a connection thread until the client has been authenticated successfully.

- `NULL`

  This state is used for the `SHOW PROCESSLIST` state.

- `Opening tables`, `Opening table`

  The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening a table until the statement is finished. It is also worth checking that your `table_cache` value is large enough.

- `optimizing`

  The server is performing initial optimizations for a query.

- `preparing`

  This state occurs during query optimization.

- `Purging old relay logs`

  The thread is removing unneeded relay log files.

- `query end`

  This state occurs after processing a query but before the `freeing items` state.

- `Reading from net`

  The server is reading a packet from the network.

- `Removing duplicates`

  The query was using `SELECT DISTINCT` in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `removing tmp table`

  The thread is removing an internal temporary table after processing a `SELECT` statement. This state is not used if no temporary table was created.

- `rename`

  The thread is renaming a table.

- `rename result table`

  The thread is processing an `ALTER TABLE` statement, has created the new table, and is renaming it to replace the original table.

- `Reopen tables`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

  The repair code is using a sort to create indexes.

- `Repair done`

  The thread has completed a multi-threaded repair for a `MyISAM` table.

- `Repair with keycache`

  The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Rolling back`

  The thread is rolling back a transaction.

- `Saving state`

  For `MyISAM` table operations such as repair or analysis, the thread is saving the new table state to the `.MYI` file header. State includes information such as number of rows, the `AUTO_INCREMENT` counter, and key distributions.

- `Searching rows for update`

  The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sending data`

  The thread is reading and processing rows for a `SELECT` statement, and sending data to the client. Because operations occurring during this this state tend to perform large amounts of disk access (reads), it is often the longest-running state over the lifetime of a given query.

- `setup`

  The thread is beginning an `ALTER TABLE` operation.

- `Sorting for group`

  The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

  The thread is doing a sort to satisfy an `ORDER BY`.

- `Sorting index`

  The thread is sorting index pages for more efficient access during a `MyISAM` table optimization operation.

- `Sorting result`

  For a `SELECT` statement, this is similar to `Creating sort index`, but for nontemporary tables.

- `statistics`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound performing other work.

- `System lock`

  The thread is going to request or is waiting for an internal or external system lock for the table. For example, this can occur when `InnoDB` waits for a table-level lock during execution of `LOCK TABLES`. If this state is being caused by requests for external locks and you are not using multiple `mysqld` servers that are accessing the same `MyISAM` tables, you can disable external system locks with the `--skip-external-locking` option. However, external locking is disabled by default, so it is likely that this option will have no effect. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `Table lock`

  The next thread state after `System lock`. After acquiring a system lock, the thread is going to request an internal table lock (a `THR_LOCK` lock).

  For more information about table lock indicators, see Section 8.11.1, “Internal Locking Methods”.

- `update`

  The thread is getting ready to start updating the table.

- `Updating`

  The thread is searching for rows to update and is updating them.

- `updating main table`

  The server is executing the first part of a multiple-table update. It is updating only the first table, and saving columns and offsets to be used for updating the other (reference) tables.

- `updating reference tables`

  The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- `User lock`

  The thread is going to request or is waiting for an advisory lock requested with a `GET_LOCK()` call. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `Waiting for release of readlock`

  The thread is waiting for a global read lock obtained by another thread (with `FLUSH TABLES WITH READ LOCK`) to be released.

- `Waiting for tables`, `Waiting for table`

  The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Waiting on cond`

  A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Waiting to get readlock`

  The thread has issued a `FLUSH TABLES WITH READ LOCK` statement to obtain a global read lock and is waiting to obtain the lock.

- `Writing to net`

  The server is writing a packet to the network.

8.14.3 Delayed-Insert Thread States

These thread states are associated with processing for `DELAYED` inserts (see Section 13.2.5.2, “INSERT DELAYED Syntax”). Some states are associated with connection threads that process `INSERT DELAYED` statements from clients. Other states are associated with delayed-insert handler threads that insert the rows. There is a delayed-insert handler thread for each table for which `INSERT DELAYED` statements are issued.

States associated with a connection thread that processes an `INSERT DELAYED` statement from the client:

- `allocating local table`

  The thread is preparing to feed rows to the delayed-insert handler thread.

- `Creating delayed handler`

  The thread is creating a handler for `DELAYED` inserts.

- `got handler lock`

  This occurs before the `allocating local table` state and after the `waiting for handler lock` state, when the connection thread gets access to the delayed-insert handler thread.

- `got old table`

  This occurs after the `waiting for handler open` state. The delayed-insert handler thread has signaled that it has ended its initialization phase, which includes opening the table for delayed inserts.

- `storing row into queue`

  The thread is adding a new row to the list of rows that the delayed-insert handler thread must insert.

- `waiting for delay_list`

  This occurs during the initialization phase when the thread is trying to find the delayed-insert handler thread for the table, and before attempting to gain access to the list of delayed-insert threads.

- `waiting for handler insert`

  An `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

- `waiting for handler lock`

  This occurs before the `allocating local table` state when the connection thread waits for access to the delayed-insert handler thread.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `waiting for handler open`

  This occurs after the `Creating delayed handler` state and before the `got old table` state. The delayed-insert handler thread has just been started, and the connection thread is waiting for it to initialize.

States associated with a delayed-insert handler thread that inserts the rows:

- `insert`

  The state that occurs just before inserting rows into the table.

- `reschedule`

  After inserting a number of rows, the delayed-insert thread sleeps to let other threads do work.

- `upgrading lock`

  A delayed-insert handler is trying to get a lock for the table to insert rows.

- `Waiting for INSERT`

  A delayed-insert handler is waiting for a connection thread to add rows to the queue (see `storing row into queue`).

8.14.4 Query Cache Thread States

These thread states are associated with the query cache (see Section 8.10.3, “The MySQL Query Cache”).

- `checking privileges on cached query`

  The server is checking whether the user has privileges to access a cached query result.

- `checking query cache for query`

  The server is checking whether the current query is present in the query cache.

- `invalidating query cache entries`

  Query cache entries are being marked invalid because the underlying tables have changed.

- `sending cached result to client`

  The server is taking the result of a query from the query cache and sending it to the client.

- `storing result in query cache`

  The server is storing the result of a query in the query cache.

8.14.5 Replication Master Thread States

The following list shows the most common states you may see in the `State` column for the master's `Binlog Dump` thread. If you see no `Binlog Dump` threads on a master server, this means that replication is not running—that is, that no slaves are currently connected.

- `Sending binlog event to slave`

  Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Finished reading one binlog; switching to next binlog`

  The thread has finished reading a binary log file and is opening the next one to send to the slave.

- `Has sent all binlog to slave; waiting for binlog to be updated`

  The thread has read all outstanding updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- `Waiting to finalize termination`

  A very brief state that occurs as the thread is stopping.

8.14.6 Replication Slave I/O Thread States

The following list shows the most common states you see in the `State` column for a slave server I/O thread. This state also appears in the `Slave_IO_State` column displayed by `SHOW SLAVE STATUS`, so you can get a good view of what is happening by using that statement.

- `Waiting for master update`

  The initial state before `Connecting to master`.

- `Connecting to master`

  The thread is attempting to connect to the master.

- `Checking master version`

  A state that occurs very briefly, after the connection to the master is established.

- `Registering slave on master`

  A state that occurs very briefly after the connection to the master is established.

- `Requesting binlog dump`

  A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log file name and position.

- `Waiting to reconnect after a failed binlog dump request`

  If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `CHANGE MASTER TO` statement or the `--master-connect-retry` option.

- `Reconnecting after a failed binlog dump request`

  The thread is trying to reconnect to the master.

- `Waiting for master to send event`

  The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_net_timeout` seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- `Queueing master event to the relay log`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Waiting to reconnect after a failed master event read`

  An error occurred while reading (due to disconnection). The thread is sleeping for the number of seconds set by the `CHANGE MASTER TO` statement or `--master-connect-retry` option (default 60) before attempting to reconnect.

- `Reconnecting after a failed master event read`

  The thread is trying to reconnect to the master. When connection is established again, the state becomes `Waiting for master to send event`.

- `Waiting for the slave SQL thread to free enough relay log space`

  You are using a nonzero `relay_log_space_limit` value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting for slave mutex on exit`

  A state that occurs briefly as the thread is stopping.

8.14.7 Replication Slave SQL Thread States

The following list shows the most common states you may see in the `State` column for a slave server SQL thread:

- `Waiting for the next event in relay log`

  The initial state before `Reading event from the relay log`.

- `Reading event from the relay log`

  The thread has read an event from the relay log so that the event can be processed.

- `Has read all relay log; waiting for the slave I/O thread to update it`

  The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- `Making temp file`

  The thread is executing a `LOAD DATA INFILE` statement and is creating a temporary file containing the data from which the slave will read rows.

- `Waiting for slave mutex on exit`

  A very brief state that occurs as the thread is stopping.

The `State` column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

8.14.8 Replication Slave Connection Thread States

These thread states occur on a replication slave but are associated with connection threads, not with the I/O or SQL threads.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Changing master`

  The thread is processing a `CHANGE MASTER TO` statement.

- `Creating table from master dump`

  The slave is creating a table using the `CREATE TABLE` statement contained in the dump from the master. Used for `LOAD TABLE FROM MASTER` and `LOAD DATA FROM MASTER`.

- `Killing slave`

  The thread is processing a `STOP SLAVE` statement.

- `Opening master dump table`

  This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

  This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

  This state occurs after `Reading master dump table data`.

- `starting slave`

  The thread is starting the slave threads after processing a successful `LOAD DATA FROM MASTER` load operation.

8.14.9 MySQL Cluster Thread States

- `Committing events to binlog`
- `Opening mysql.ndb_apply_status`
- `Processing events`

  The thread is processing events for binary logging.

- `Processing events from schema table`

  The thread is doing the work of schema replication.

- `Shutting down`
- `Syncing ndb table schema operation and binlog`

  This is used to have a correct binary log of schema operations for NDB.

- `Waiting for event from ndbcluster`

  The server is acting as an SQL node in a MySQL Cluster, and is connected to a cluster management node.

- `Waiting for first event from ndbcluster`
- `Waiting for ndbcluster binlog update to reach current position`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Waiting for ndbcluster to start`
- `Waiting for schema epoch`

  The thread is waiting for a schema epoch (that is, a global checkpoint).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 9 Language Structure

Table of Contents

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Keywords and reserved words
- User-defined and system variables
- Comments

9.1 Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and `NULL`. The section also covers the various nuances and “gotchas” that you may run into when dealing with these basic types in MySQL.

9.1.1 String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (“`'`”) or double quote (“`"`”) characters. Examples:

```
'a string'
"another string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'
'a' ' ' 'string'
```


This documentation is for an older version. If you're

This documentation is for an older version. If you're

If the ANSI_QUOTES SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

A *binary string* is a string of bytes that has no character set or collation. A *nonbinary string* is a string of characters that has a character set and collation. For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte. For nonbinary strings the unit is the character and some character sets support multibyte characters. Character value ordering is a function of the string collation.

String literals may have an optional character set introducer and COLLATE clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

You can use N'*literal*' (or n'*literal*') to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For more information about these forms of string syntax, see Section 10.1.3.5, “Character String Literal Character Set and Collation”, and Section 10.1.3.6, “National Character Set”.

Within a string, certain sequences have special meaning unless the NO_BACKSLASH_ESCAPES SQL mode is enabled. Each of these sequences begins with a backslash (“\”), known as the *escape character*. MySQL recognizes the escape sequences shown in Table 9.1, “Special Character Escape Sequences”. For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, “\x” is just “x”. These sequences are case sensitive. For example, “\b” is interpreted as a backspace, but “\B” is interpreted as “B”. Escape processing is done according to the character set indicated by the character_set_connection system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in Section 10.1.3.5, “Character String Literal Character Set and Collation”.

Table 9.1 Special Character Escape Sequences

Escape Sequence	Character Represented by Sequence
\0	An ASCII NUL (X'00') character
\'	A single quote (“'”) character
\"	A double quote (“"”) character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character
\Z	ASCII 26 (Control+Z); see note following the table
\\	A backslash (“\”) character
\%	A “%” character; see note following the table

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Escape Sequence	Character Represented by Sequence
`\_`	A "_" character; see note following the table

The ASCII 26 character can be encoded as "`\Z`" to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

The "`\%`" and "`\_`" sequences are used to search for literal instances of "`%`" and "`_`" in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the `LIKE` operator in Section 12.5.1, "String Comparison Functions". If you use "`\%`" or "`\_`" outside of pattern-matching contexts, they evaluate to the strings "`\%`" and "`\_`", not to "`%`" and "`_`".

There are several ways to include quote characters within a string:

- A "`'`" inside a string quoted with "`'`" may be written as "`''`".
- A "`"`" inside a string quoted with "`"`" may be written as "`""`".
- Precede the quote character by an escape character ("`\`").
- A "`'`" inside a string quoted with "`"`" needs no special treatment and need not be doubled or escaped. In the same way, "`"`" inside a string quoted with "`'`" needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
+------------------------+
```

If you want to insert binary data into a string column (such as a `BLOB` column), you should represent certain characters by escape sequences. Backslash ("`\`") and the quote character used to quote the string must be escaped. In certain client environments, it may also be necessary to escape `NUL` or Control+Z. The `mysql` client truncates quoted strings containing `NUL` characters if they are not escaped, and Control+Z may be taken for END-OF-FILE on Windows if not escaped. For the escape sequences that represent each of these characters, see Table 9.1, "Special Character Escape Sequences".

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See Section 20.6.7.53, “mysql_real_escape_string()”. Within SQL statements that construct other SQL statements, you can use the `QUOTE()` function. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 20.8, “MySQL Perl API”. Other language interfaces may provide a similar capability.
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

9.1.2 Number Literals

Number literals include exact-value (integer and `DECIMAL`) literals and approximate-value (floating-point) literals.

Integers are represented as a sequence of digits. Numbers may include “`.`” as a decimal separator. Numbers may be preceded by “`-`” or “`+`” to indicate a negative or positive value, respectively. Numbers represented in scientific notation with a mantissa and exponent are approximate-value numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types. For more information about exact-value calculations, see Section 12.17, “Precision Math”.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3 Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of `'2015-07-21'`, `'20150721'`, and `20150721` as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, consult these sections:

- Section 11.1.2, “Date and Time Type Overview”
- Section 11.3, “Date and Time Types”

Standard SQL and ODBC Date and Time Literals. Standard SQL permits temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

```
DATE 'str'
TIME 'str'
TIMESTAMP 'str'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL recognizes those constructions and also the corresponding ODBC syntax:

```
{ d 'str' }
{ t 'str' }
{ ts 'str' }
```

However, MySQL ignores the type keyword and each of the preceding constructions produces the string value `'str'`, with a type of `VARCHAR`.

String and Numeric Literals in Date and Time Context. MySQL recognizes `DATE` values in these formats:

- As a string in either `'YYYY-MM-DD'` or `'YY-MM-DD'` format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, `'2012-12-31'`, `'2012/12/31'`, `'2012^12^31'`, and `'2012@12@31'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDD'` or `'YYMMDD'` format, provided that the string makes sense as a date. For example, `'20070523'` and `'070523'` are interpreted as `'2007-05-23'`, but `'071332'` is illegal (it has nonsensical month and day parts) and becomes `'0000-00-00'`.
- As a number in either `YYYYMMDD` or `YYMMDD` format, provided that the number makes sense as a date. For example, `19830905` and `830905` are interpreted as `'1983-09-05'`.

MySQL recognizes `DATETIME` and `TIMESTAMP` values in these formats:

- As a string in either `'YYYY-MM-DD HH:MM:SS'` or `'YY-MM-DD HH:MM:SS'` format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, `'2012-12-31 11:30:45'`, `'2012^12^31 11+30+45'`, `'2012/12/31 11*30*45'`, and `'2012@12@31 11^30^45'` are equivalent.

  The date and time parts can be separated by `T` rather than a space. For example, `'2012-12-31 11:30:45'` `'2012-12-31T11:30:45'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDDHHMMSS'` or `'YYMMDDHHMMSS'` format, provided that the string makes sense as a date. For example, `'20070523091528'` and `'070523091528'` are interpreted as `'2007-05-23 09:15:28'`, but `'071122129015'` is illegal (it has a nonsensical minute part) and becomes `'0000-00-00 00:00:00'`.
- As a number in either `YYYYMMDDHHMMSS` or `YYMMDDHHMMSS` format, provided that the number makes sense as a date. For example, `19830905132800` and `830905132800` are interpreted as `'1983-09-05 13:28:00'`.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `DATETIME` or `TIMESTAMP` columns. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”.

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

- Year values in the range `70-99` are converted to `1970-1999`.
- Year values in the range `00-69` are converted to `2000-2069`.

See also Section 11.3.8, “Two-Digit Years in Dates”.

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than `10`. `'2015-6-9'` is the same as `'2015-06-09'`. Similarly, for

This documentation is for an older version. If you're
This documentation is for an older version. If you're

values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than `10`. `'2015-10-30 1:2:3'` is the same as `'2015-10-30 01:02:03'`.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in `YYYYMMDD` or `YYYYMMDDHHMMSS` format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in `YYMMDD` or `YYMMDDHHMMSS` format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify `'9903'`, thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are `99` and `03`, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value `'1999-03-00'`, use `'990300'`.

MySQL recognizes `TIME` values in these formats:

- As a string in `'D HH:MM:SS'` format. You can also use one of the following “relaxed” syntaxes: `'HH:MM:SS'`, `'HH:MM'`, `'D HH:MM'`, `'D HH'`, or `'SS'`. Here `D` represents days and can have a value from 0 to 34.
- As a string with no delimiters in `'HHMMSS'` format, provided that it makes sense as a time. For example, `'101112'` is understood as `'10:11:12'`, but `'109712'` is illegal (it has a nonsensical minute part) and becomes `'00:00:00'`.
- As a number in `HHMMSS` format, provided that it makes sense as a time. For example, `101112` is understood as `'10:11:12'`. The following alternative formats are also understood: `SS`, `MMSS`, or `HHMMSS`.

A trailing fractional seconds part is recognized in the `'D HH:MM:SS.fraction'`, `'HH:MM:SS.fraction'`, `'HHMMSS.fraction'`, and `HHMMSS.fraction` time formats, where `fraction` is the fractional part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `TIME` columns. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”.

For `TIME` values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than `10`. `'8:3:2'` is the same as `'08:03:02'`.

9.1.4 Hexadecimal Literals

MySQL supports hexadecimal values, written using `X'val'`, `x'val'`, or `0xval` format, where *`val`* contains hexadecimal digits (`0..9`, `A..F`). Lettercase of the digits does not matter. For values written using `X'val'` or `x'val'` format, *`val`* must contain an even number of digits. For values written using `0xval` syntax, values that contain an odd number of digits are treated as having an extra leading `0`. For example, `0x0a` and `0xaaa` are interpreted as `0x0a` and `0x0aaa`.

In numeric contexts, hexadecimal values act like integers (64-bit precision). In string contexts, they act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT X'4D7953514C';
        -> 'MySQL'
mysql> SELECT x'0a'+0;
        -> 10
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT 0x5061756c;
       -> 'Paul'
```

The default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST(... AS UNSIGNED)`:

```
mysql> SELECT X'41', CAST(X'41' AS UNSIGNED);
       -> 'A', 65
```

The `X'hexstring'` and `x'val'` syntaxes are based on standard SQL. The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for `BLOB` columns.

To convert a string or a number to a string in hexadecimal format, use the `HEX()` function:

```
mysql> SELECT HEX('cat');
       -> '636174'
mysql> SELECT X'636174';
       -> 'cat'
```

9.1.5 Boolean Literals

The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
       -> 1, 1, 0, 0
```

9.1.6 Bit-Field Literals

Beginning with MySQL 5.0.3, bit-field values can be written using `b'value'` or `0bvalue` notation. *`value`* is a binary value written using zeros and ones.

Bit-field notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

Bit values are returned as binary values. To display them in printable form, add 0 or use a conversion function such as `BIN()`. High-order 0 bits are not displayed in the converted value.

```
mysql> SELECT b+0, BIN(b+0), OCT(b+0), HEX(b+0) FROM t;
+------+----------+----------+----------+
| b+0  | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+------+----------+----------+----------+
|  255 | 11111111 | 377      | FF       |
|   10 | 1010     | 12       | A        |
|    5 | 101      | 5        | 5        |
+------+----------+----------+----------+
```

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()` or `+0`:

```
mysql> SET @v1 = 0b1000001;
mysql> SET @v2 = CAST(0b1000001 AS UNSIGNED), @v3 = 0b1000001+0;
mysql> SELECT @v1, @v2, @v3;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

9.1.7 NULL Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase. A synonym is `\N` (case sensitive).

For text file import or export operations performed with `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See Section 13.2.6, “LOAD DATA INFILE Syntax”.

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. For more information, see Section B.5.4.3, “Problems with NULL Values”.

9.2 Schema Object Names

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, partition, and other object names are known as identifiers. This section describes the permissible syntax for identifiers in MySQL. Section 9.2.2, “Identifier Case Sensitivity”, describes which types of identifiers are case sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you *must* quote it whenever you refer to it. (Exception: A reserved word that follows a period in a qualified name must be an identifier, so it need not be quoted.) Reserved words are listed at Section 9.3, “Keywords and Reserved Words”.

Identifiers are converted to Unicode internally. They may contain these characters:

- Permitted characters in unquoted identifiers:
  - ASCII: [0-9,a-z,A-Z$_] (basic Latin letters, digits 0-9, dollar, underscore)
  - Extended: U+0080 .. U+FFFF
- Permitted characters in quoted identifiers include the full Unicode Basic Multilingual Plane (BMP), except U+0000:
  - ASCII: U+0001 .. U+007F
  - Extended: U+0080 .. U+FFFF
- ASCII NUL (U+0000) and supplementary characters (U+10000 and higher) are not permitted in quoted or unquoted identifiers.
- Identifiers may begin with a digit but unless quoted may not consist solely of digits.
- Database, table, and column names cannot end with space characters.
- Database and table names cannot contain “`/`”, “`\`”, “`.`”, or characters that are not permitted in file names.

The identifier quote character is the backtick (“`` ` ``”):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the `ANSI_QUOTES` SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

The `ANSI_QUOTES` mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in Section 5.1.7, “Server SQL Modes”.

Identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named `a``b` that contains a column named `c"d`:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
mysql> SELECT 1 AS `one`, 2 AS 'two';
+-----+-----+
| one | two |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

It is recommended that you do not use names that begin with *`Me`* or *`MeN`*, where *`M`* and *`N`* are integers. For example, avoid using `1e` as an identifier, because an expression such as `1e+3` is ambiguous. Depending on context, it might be interpreted as the expression `1e + 3` or as the number `1e+3`.

Be careful when using `MD5()` to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See Section 9.4, “User-Defined Variables”, for more information and examples of workarounds.

The following table describes the maximum length for each type of identifier.

Identifier	Maximum Length (characters)
Database	64 (`NDB` storage engine: 63)
Table	64 (`NDB` storage engine: 63)
Column	64
Index	64
Constraint	64
Stored Program	64
View	64
Alias	256 (see exception following table)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Identifier	Maximum Length (characters)
Compound Statement Label	16

As of MySQL 5.0.52, aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in `.frm` files and to identifiers stored in the grant tables in the `mysql` database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multibyte characters without reducing the number of characters permitted for values stored in these columns. As indicated earlier, the permissible Unicode characters are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted.

For tables using the `NDB` storage engine, there is an additional requirement that the combined length of a table name and the name of the database in which it is found must not exceed 122 characters. See Section 17.1.5.5, “Limits Associated with Database Objects in MySQL Cluster”.

9.2.1 Identifier Qualifiers

MySQL permits names that consist of a single identifier or multiple identifiers. The components of a multiple-part name must be separated by period (“`.`”) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL, you can refer to a table column using any of the following forms.

Column Reference	Meaning
`col_name`	The column *`col_name`* from whichever table used in the statement contains a column of that name.
`tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the default database.
`db_name.tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the database *`db_name`*.

The qualifier character is a separate token and need not be contiguous with the associated identifiers. For example, *`tbl_name.col_name`* and *`tbl_name . col_name`* are equivalent.

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write `` `my-table`.`my-column` ``, not `` `my-table.my-column` ``.

A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.

You need not specify a *`tbl_name`* or *`db_name.tbl_name`* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables `t1` and `t2` each contain a column `c`, and you retrieve `c` in a `SELECT` statement that uses both `t1` and `t2`. In this case, `c` is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as `t1.c` or `t2.c` to indicate which table you mean. Similarly, to retrieve from a table `t` in database `db1` and from a table `t` in database `db2` in the same statement, you must refer to columns in those tables as `db1.t.`*`col_name`* and `db2.t.`*`col_name`*.

The syntax *`.tbl_name`* means the table *`tbl_name`* in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a “`.`” character.

9.2.2 Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage

This documentation is for an older version. If you're
This documentation is for an older version. If you're

engine). Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database and table names. This means database and table names are not case sensitive in Windows, and case sensitive in most varieties of Unix. One notable exception is OS X, which is Unix-based but uses a default file system type (HFS+) that is not case sensitive. However, OS X also supports UFS volumes, which are case sensitive just as on any Unix. See Section 1.8.1, "MySQL Extensions to Standard SQL". The `lower_case_table_names` system variable also affects how the server handles identifier case sensitivity, as described later in this section.

Note

Although database and table names are not case sensitive on some platforms, you should not refer to a given database or table using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index, and stored routine names are not case sensitive on any platform, nor are column aliases. Trigger names are case sensitive, which differs from standard SQL.

By default, table aliases are case sensitive on Unix, but not so on Windows or OS X. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
    -> WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` system variable, which you can set when starting `mysqld`. `lower_case_table_names` can take the values shown in the following table. On Unix, the default value of `lower_case_table_names` is 0. On Windows, the default value is 1. On OS X, the default value is 2.

Value	Meaning
`0`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement. Name comparisons are case sensitive. You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or OS X). If you force this variable to 0 with `--lower-case-table-names=0` on a case-insensitive file system and access `MyISAM` tablenames using different lettercases, index corruption may result.
`1`	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
`2`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. This works *only* on file systems that are not case sensitive! `InnoDB` table names are stored in lowercase, as for `lower_case_table_names=1`.

If you are using MySQL on only one platform, you do not normally have to change the `lower_case_table_names` variable from its default value. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

you can have two different tables named `my_table` and `MY_TABLE`, but on Windows these two names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you do not see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

  Exception: If you are using `InnoDB` tables and you are trying to avoid these data transfer problems, you should set `lower_case_table_names` to 1 on all platforms to force names to be converted to lowercase.

If you plan to set the `lower_case_table_names` system variable to 1 on Unix, you must first convert your old database and table names to lowercase before stopping `mysqld` and restarting it with the new variable setting. To do this for an individual table, use `RENAME TABLE`:

```
RENAME TABLE T1 TO t1;
```

To convert one or more entire databases, dump them before setting `lower_case_table_names`, then drop the databases, and reload them after setting `lower_case_table_names`:

1. Use `mysqldump` to dump each database:

   ```
   mysqldump --databases db1 > db1.sql
   mysqldump --databases db2 > db2.sql
   ...
   ```

   Do this for each database that must be recreated.

2. Use `DROP DATABASE` to drop each database.
3. Stop the server, set `lower_case_table_names`, and restart the server.
4. Reload the dump file for each database. Because `lower_case_table_names` is set, each database and table name will be converted to lowercase as it is recreated:

   ```
   mysql < db1.sql
   mysql < db2.sql
   ...
   ```

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, procedures, functions, savepoints, stored routine parameters and stored program local variables. It is not true for names of names of columns, constraints, databases, statements prepared with `PREPARE`, tables, triggers, users, and user-defined variables.

9.2.3 Function Name Parsing and Resolution

MySQL 5.0 supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `count` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following "`(`" parenthesis character.
- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The `sql_functions[]` array in the `sql/lex.h` source file lists the names of these special functions for which following whitespace determines their interpretation. Before MySQL 5.1, these are rather numerous (about 200). In MySQL 5.1, parser improvements reduce to about 30 the number of affected function names. You may find it easiest to treat the no-whitespace requirement as applying to all function calls.

For functions not listed as special in `sql/lex.h`, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. `ASCII` is one such name. However, for these nonaffected function names, interpretation may vary in expression context: *`func_name`* `()` is interpreted as a built-in function if there is one with the given name; if not, *`func_name`* `()` is interpreted as a user-defined function or stored function if one exists with that name.

The `IGNORE_SPACE` SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With `IGNORE_SPACE` disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

  ```
  mysql> CREATE TABLE count(i INT);
  ERROR 1064 (42000): You have an error in your SQL syntax ...
  near 'count(i INT)'
  ```

  To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

  ```
  CREATE TABLE count (i INT);
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- With `IGNORE_SPACE` enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

  However, enabling `IGNORE_SPACE` also has the side effect that the parser treats the affected function names as reserved words (see Section 9.3, “Keywords and Reserved Words”). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with `IGNORE_SPACE` enabled, both of the following statements fail with a syntax error because the parser interprets `count` as a reserved word:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

  To use the function name in nonexpression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

To enable the `IGNORE_SPACE` SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

`IGNORE_SPACE` is also enabled by certain other composite modes such as `ANSI` that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check Section 5.1.7, “Server SQL Modes”, to see which composite modes enable `IGNORE_SPACE`.

To minimize the dependency of SQL code on the `IGNORE_SPACE` setting, use these guidelines:

- Avoid creating UDFs or stored functions that have the same name as a built-in function.
- Avoid using function names in nonexpression context. For example, these statements use `count` (one of the affected function names affected by `IGNORE_SPACE`), so they fail with or without whitespace following the name if `IGNORE_SPACE` is enabled:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

  If you must use a function name in nonexpression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

- Built-in functions and user-defined functions

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A UDF can be created with the same name as a built-in function but the UDF cannot be invoked because the parser resolves invocations of the function to refer to the built-in function. For example, if you create a UDF named `ABS`, references to `ABS()` invoke the built-in function.

- Built-in functions and stored functions

  It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a database name. For example, if you create a stored function named `PI` in the `test` database, you invoke it as `test.PI()` because the server resolves `PI()` as a reference to the built-in function.

- User-defined functions and stored functions

  User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

- If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF and `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. Then modify any affected code to use the new name.
- If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a schema qualifier (that is, use *`schema_name.func_name()`* syntax). In either case, modify any affected code accordingly.

9.3 Keywords and Reserved Words

Keywords are words that have significance in SQL. Certain keywords, such as `SELECT`, `DELETE`, or `BIGINT`, are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Nonreserved keywords are permitted as identifiers without quoting. Reserved words are permitted as identifiers if you quote them as described in Section 9.2, “Schema Object Names”:

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'
```

`BEGIN` and `END` are keywords but not reserved, so their use as identifiers does not require quoting. `INTERVAL` is a reserved keyword and must be quoted to be used as an identifier:

```
mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, `COUNT` is acceptable as a column name. However, by default, no whitespace is permitted in

This documentation is for an older version. If you're
This documentation is for an older version. If you're

function invocations between the function name and the following "(" character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further details on recognition of function names, see Section 9.2.3, "Function Name Parsing and Resolution".

The following table shows the keywords and reserved words in MySQL 5.0. Reserved keywords are marked with (R).

At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the reserved words in the table are forbidden by standard SQL as column or table names (for example, GROUP). A few are reserved because MySQL needs them and uses a yacc parser.

Table 9.2 Keywords and Reserved Words in MySQL 5.0

ACTION	ADD (R)	AFTER
AGAINST	AGGREGATE	ALGORITHM
ALL (R)	ALTER (R)	ANALYZE (R)
AND (R)	ANY	AS (R)
ASC (R)	ASCII	ASENSITIVE (R)
AUTO_INCREMENT	AVG	AVG_ROW_LENGTH
BACKUP	BDB	BEFORE (R)
BEGIN	BERKELEYDB	BETWEEN (R)
BIGINT (R)	BINARY (R)	BINLOG
BIT	BLOB (R)	BLOCK
BOOL	BOOLEAN	BOTH (R)
BTREE	BY (R)	BYTE
CACHE	CALL (R)	CASCADE (R)
CASCADED	CASE (R)	CHAIN
CHANGE (R)	CHANGED	CHAR (R)
CHARACTER (R)	CHARSET	CHECK (R)
CHECKSUM	CIPHER	CLIENT
CLOSE	CODE	COLLATE (R)
COLLATION	COLUMN (R)	COLUMNS
COMMENT	COMMIT	COMMITTED
COMPACT	COMPRESSED	CONCURRENT
CONDITION (R)	CONNECTION	CONSISTENT
CONSTRAINT (R)	CONTAINS	CONTEXT
CONTINUE (R)	CONVERT (R)	CPU
CREATE (R)	CROSS (R)	CUBE
CURRENT_DATE (R)	CURRENT_TIME (R)	CURRENT_TIMESTAMP (R)
CURRENT_USER (R)	CURSOR (R)	DATA
DATABASE (R)	DATABASES (R)	DATE
DATETIME	DAY	DAY_HOUR (R)
DAY_MICROSECOND (R)	DAY_MINUTE (R)	DAY_SECOND (R)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

DEALLOCATE	DEC (R)	DECIMAL (R)
DECLARE (R)	DEFAULT (R)	DEFINER
DELAYED (R)	DELAY_KEY_WRITE	DELETE (R)
DESC (R)	DESCRIBE (R)	DES_KEY_FILE
DETERMINISTIC (R)	DIRECTORY	DISABLE
DISCARD	DISTINCT (R)	DISTINCTROW (R)
DIV (R)	DO	DOUBLE (R)
DROP (R)	DUAL (R)	DUMPFILE
DUPLICATE	DYNAMIC	EACH (R)
ELSE (R)	ELSEIF (R)	ENABLE
ENCLOSED (R)	END	ENGINE
ENGINES	ENUM	ERRORS
ESCAPE	ESCAPED (R)	EVENTS
EXECUTE	EXISTS (R)	EXIT (R)
EXPANSION	EXPLAIN (R)	EXTENDED
FALSE (R)	FAST	FAULTS
FETCH (R)	FIELDS	FILE
FIRST	FIXED	FLOAT (R)
FLOAT4 (R)	FLOAT8 (R)	FLUSH
FOR (R)	FORCE (R)	FOREIGN (R)
FOUND	FRAC_SECOND	FROM (R)
FULL	FULLTEXT (R)	FUNCTION
GEOMETRY	GEOMETRYCOLLECTION	GET_FORMAT
GLOBAL	GRANT (R)	GRANTS
GROUP (R)	HANDLER	HASH
HAVING (R)	HELP	HIGH_PRIORITY (R)
HOSTS	HOUR	HOUR_MICROSECOND (R)
HOUR_MINUTE (R)	HOUR_SECOND (R)	IDENTIFIED
IF (R)	IGNORE (R)	IMPORT
IN (R)	INDEX (R)	INDEXES
INFILE (R)	INNER (R)	INNOBASE
INNODB	INOUT (R)	INSENSITIVE (R)
INSERT (R)	INSERT_METHOD	INT (R)
INT1 (R)	INT2 (R)	INT3 (R)
INT4 (R)	INT8 (R)	INTEGER (R)
INTERVAL (R)	INTO (R)	INVOKER
IO	IO_THREAD	IPC
IS (R)	ISOLATION	ISSUER
ITERATE (R)	JOIN (R)	KEY (R)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

KEYS (R)	KILL (R)	LANGUAGE
LAST	LEADING (R)	LEAVE (R)
LEAVES	LEFT (R)	LEVEL
LIKE (R)	LIMIT (R)	LINES (R)
LINESTRING	LOAD (R)	LOCAL
LOCALTIME (R)	LOCALTIMESTAMP (R)	LOCK (R)
LOCKS	LOGS	LONG (R)
LONGBLOB (R)	LONGTEXT (R)	LOOP (R)
LOW_PRIORITY (R)	MASTER	MASTER_CONNECT_RETRY
MASTER_HOST	MASTER_LOG_FILE	MASTER_LOG_POS
MASTER_PASSWORD	MASTER_PORT	MASTER_SERVER_ID
MASTER_SSL	MASTER_SSL_CA	MASTER_SSL_CAPATH
MASTER_SSL_CERT	MASTER_SSL_CIPHER	MASTER_SSL_KEY
MASTER_USER	MATCH (R)	MAX_CONNECTIONS_PER_HOUR
MAX_QUERIES_PER_HOUR	MAX_ROWS	MAX_UPDATES_PER_HOUR
MAX_USER_CONNECTIONS	MEDIUM	MEDIUMBLOB (R)
MEDIUMINT (R)	MEDIUMTEXT (R)	MEMORY
MERGE	MICROSECOND	MIDDLEINT (R)
MIGRATE	MINUTE	MINUTE_MICROSECOND (R)
MINUTE_SECOND (R)	MIN_ROWS	MOD (R)
MODE	MODIFIES (R)	MODIFY
MONTH	MULTILINESTRING	MULTIPOINT
MULTIPOLYGON	MUTEX	NAME
NAMES	NATIONAL	NATURAL (R)
NCHAR	NDB	NDBCLUSTER
NEW	NEXT	NO
NONE	NOT (R)	NO_WRITE_TO_BINLOG (R)
NULL (R)	NUMERIC (R)	NVARCHAR
OFFSET	OLD_PASSWORD	ON (R)
ONE	ONE_SHOT	OPEN
OPTIMIZE (R)	OPTION (R)	OPTIONALLY (R)
OR (R)	ORDER (R)	OUT (R)
OUTER (R)	OUTFILE (R)	PACK_KEYS
PAGE	PARTIAL	PASSWORD
PHASE	POINT	POLYGON
PRECISION (R)	PREPARE	PREV
PRIMARY (R)	PRIVILEGES	PROCEDURE (R)
PROCESSLIST	PROFILE	PROFILES
PURGE (R)	QUARTER	QUERY

This documentation is for an older version. If you're
This documentation is for an older version. If you're

QUICK	RAID0	RAID_CHUNKS
RAID_CHUNKSIZE	RAID_TYPE	READ (R)
READS (R)	REAL (R)	RECOVER
REDUNDANT	REFERENCES (R)	REGEXP (R)
RELAY_LOG_FILE	RELAY_LOG_POS	RELAY_THREAD
RELEASE (R)	RELOAD	RENAME (R)
REPAIR	REPEAT (R)	REPEATABLE
REPLACE (R)	REPLICATION	REQUIRE (R)
RESET	RESTORE	RESTRICT (R)
RESUME	RETURN (R)	RETURNS
REVOKE (R)	RIGHT (R)	RLIKE (R)
ROLLBACK	ROLLUP	ROUTINE
ROW	ROWS	ROW_FORMAT
RTREE	SAVEPOINT	SCHEMA (R)
SCHEMAS (R)	SECOND	SECOND_MICROSECOND (R)
SECURITY	SELECT (R)	SENSITIVE (R)
SEPARATOR (R)	SERIAL	SERIALIZABLE
SESSION	SET (R)	SHARE
SHOW (R)	SHUTDOWN	SIGNED
SIMPLE	SLAVE	SMALLINT (R)
SNAPSHOT	SOME	SONAME (R)
SOUNDS	SOURCE	SPATIAL (R)
SPECIFIC (R)	SQL (R)	SQLEXCEPTION (R)
SQLSTATE (R)	SQLWARNING (R)	SQL_BIG_RESULT (R)
SQL_BUFFER_RESULT	SQL_CACHE	SQL_CALC_FOUND_ROWS (R)
SQL_NO_CACHE	SQL_SMALL_RESULT (R)	SQL_THREAD
SQL_TSI_DAY	SQL_TSI_FRAC_SECOND	SQL_TSI_HOUR
SQL_TSI_MINUTE	SQL_TSI_MONTH	SQL_TSI_QUARTER
SQL_TSI_SECOND	SQL_TSI_WEEK	SQL_TSI_YEAR
SSL (R)	START	STARTING (R)
STATUS	STOP	STORAGE
STRAIGHT_JOIN (R)	STRING	STRIPED
SUBJECT	SUPER	SUSPEND
SWAPS	SWITCHES	TABLE (R)
TABLES	TABLESPACE	TEMPORARY
TEMPTABLE	TERMINATED (R)	TEXT
THEN (R)	TIME	TIMESTAMP
TIMESTAMPADD	TIMESTAMPDIFF	TINYBLOB (R)
TINYINT (R)	TINYTEXT (R)	TO (R)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

TRAILING (R)	TRANSACTION	TRIGGER (R)
TRIGGERS	TRUE (R)	TRUNCATE
TYPE	TYPES	UNCOMMITTED
UNDEFINED	UNDO (R)	UNICODE
UNION (R)	UNIQUE (R)	UNKNOWN
UNLOCK (R)	UNSIGNED (R)	UNTIL
UPDATE (R)	UPGRADE	USAGE (R)
USE (R)	USER	USER_RESOURCES
USE_FRM	USING (R)	UTC_DATE (R)
UTC_TIME (R)	UTC_TIMESTAMP (R)	VALUE
VALUES (R)	VARBINARY (R)	VARCHAR (R)
VARCHARACTER (R)	VARIABLES	VARYING (R)
VIEW	WARNINGS	WEEK
WHEN (R)	WHERE (R)	WHILE (R)
WITH (R)	WORK	WRITE (R)
X509	XA	XOR (R)
YEAR	YEAR_MONTH (R)	ZEROFILL (R)

The following table shows the keywords and reserved words that are new in MySQL 5.0. Reserved keywords are marked with (R).

Table 9.3 New Keywords and Reserved Words in MySQL 5.0 compared to MySQL 4.1

ALGORITHM	ASENSITIVE (R)	BLOCK
CALL (R)	CASCADED	CHAIN
CODE	COMPACT	CONDITION (R)
CONNECTION	CONTAINS	CONTEXT
CONTINUE (R)	CPU	CURSOR (R)
DECLARE (R)	DEFINER	DETERMINISTIC (R)
EACH (R)	ELSEIF (R)	EXIT (R)
FAULTS	FETCH (R)	FOUND
FRAC_SECOND	INOUT (R)	INSENSITIVE (R)
INVOKER	IO	IPC
ITERATE (R)	LANGUAGE	LEAVE (R)
LOOP (R)	MAX_USER_CONNECTIONS	MEMORY
MERGE	MIGRATE	MODIFIES (R)
MUTEX	NAME	ONE
OUT (R)	PAGE	PHASE
PROFILE	PROFILES	QUARTER
READS (R)	RECOVER	REDUNDANT
RELEASE (R)	REPEAT (R)	RESUME

This documentation is for an older version. If you're
This documentation is for an older version. If you're

RETURN (R)	ROUTINE	SCHEMA (R)
SCHEMAS (R)	SECURITY	SENSITIVE (R)
SOURCE	SPECIFIC (R)	SQL (R)
SQLEXCEPTION (R)	SQLSTATE (R)	SQLWARNING (R)
SQL_TSI_DAY	SQL_TSI_FRAC_SECOND	SQL_TSI_HOUR
SQL_TSI_MINUTE	SQL_TSI_MONTH	SQL_TSI_QUARTER
SQL_TSI_SECOND	SQL_TSI_WEEK	SQL_TSI_YEAR
SUSPEND	SWAPS	SWITCHES
TEMPTABLE	TIMESTAMPADD	TIMESTAMPDIFF
TRIGGER (R)	TRIGGERS	UNDEFINED
UNDO (R)	UNKNOWN	UPGRADE
VIEW	WEEK	WHILE (R)
XA		

9.4 User-Defined Variables

You can store a value in a user-defined variable in one statement and then refer to it later in another statement. This enables you to pass values from one statement to another.

User variables are written as *@var_name*, where the variable name *var_name* consists of alphanumeric characters, “.”, “_”, and “$”. A user variable name can contain other characters if you quote it as a string or identifier (for example, @'my-var', @"my-var", or @`my-var`).

User-defined variables are session-specific. That is, a user variable defined by one client cannot be seen or used by other clients. All variables for a given client session are automatically freed when that client exits.

User variable names are not case sensitive in MySQL 5.0 and up, but are case sensitive before MySQL 5.0.

One way to set a user-defined variable is by issuing a SET statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For SET, either = or := can be used as the assignment operator.

You can also assign a value to a user variable in statements other than SET. In this case, the assignment operator must be := and not = because the latter is treated as the comparison operator = in non-SET statements:

```
mysql> SET @t1=1, @t2=2, @t3:=4;
mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
+------+------+------+--------------------+
| @t1  | @t2  | @t3  | @t4 := @t1+@t2+@t3 |
+------+------+------+--------------------+
|    1 |    2 |    4 |                  7 |
+------+------+------+--------------------+
```

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or NULL value. Assignment of decimal and real values does not preserve the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string.

If a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is implicit as of MySQL 5.0.3. (This is the same coercibility as for table column values.)

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()` or `+0`:

```
mysql> SET @v1 = b'1000001';
mysql> SET @v2 = CAST(b'1000001' AS UNSIGNED), @v3 = b'1000001'+0;
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

As a general rule, other than in `SET` statements, you should never assign a value to a user variable and read the value within the same statement. For example, to increment a variable, this is okay:

```
SET @a = @a + 1;
```

For other statements, such as `SELECT`, you might get the results you expect, but this is not guaranteed. In the following statement, you might think that MySQL will evaluate `@a` first and then do an assignment second:

```
SELECT @a, @a:=@a+1, ...;
```

However, the order of evaluation for expressions involving user variables is undefined.

Another issue with assigning a value to a variable and reading the value within the same non-`SET` statement is that the default result type of a variable is based on its type at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this `SELECT` statement, MySQL reports to the client that column one is a string and converts all accesses of `@a` to strings, even though @a is set to a number for the second row. After the `SELECT` statement executes, `@a` is regarded as a number for the next statement.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to `0`, `0.0`, or `''` to define its type before you use it.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In a `SELECT` statement, each select expression is evaluated only when sent to the client. This means that in a `HAVING`, `GROUP BY`, or `ORDER BY` clause, referring to a variable that is assigned a value in the select expression list does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to `b` in the `HAVING` clause refers to an alias for an expression in the select list that uses `@aa`. This does not work as expected: `@aa` contains the value of `id` from the previous selected row, not from the current row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as `SELECT`. This is true even if the variable is quoted, as shown in the following example:

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| c1   |
+------+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| `c1` |
+------+
1 row in set (0.00 sec)
```

An exception to this principle that user variables cannot be used to provide identifiers is that if you are constructing a string for use as a prepared statement to be executed later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

See Section 13.5, “SQL Syntax for Prepared Statements”, for more information.

A similar technique can be used in application programs to construct SQL statements using program variables, as shown here using PHP 5:

```
<?php
  $mysqli = new mysqli("localhost", "user", "pass", "test");

  if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

  $col = "c1";

  $query = "SELECT $col FROM t";

  $result = $mysqli->query($query);

  while($row = $result->fetch_assoc())
  {
    echo "<p>" . $row["$col"] . "</p>\n";
  }

  $result->close();

  $mysqli->close();
?>
```

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

9.5 Expression Syntax

The following rules define expression syntax in MySQL. The grammar shown here is based on that given in the `sql/sql_yacc.yy` file of MySQL source distributions. See the notes after the grammar for additional information about some of the terms.

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
    bit_expr [NOT] IN (subquery)
  | bit_expr [NOT] IN (expr [, expr] ...)
  | bit_expr [NOT] BETWEEN bit_expr AND predicate
  | bit_expr SOUNDS LIKE bit_expr
  | bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
  | bit_expr [NOT] REGEXP bit_expr
  | bit_expr

bit_expr:
    bit_expr | bit_expr
  | bit_expr & bit_expr
  | bit_expr << bit_expr
  | bit_expr >> bit_expr
  | bit_expr + bit_expr
  | bit_expr - bit_expr
  | bit_expr * bit_expr
  | bit_expr / bit_expr
  | bit_expr DIV bit_expr
  | bit_expr MOD bit_expr
  | bit_expr % bit_expr
  | bit_expr ^ bit_expr
  | bit_expr + interval_expr
  | bit_expr - interval_expr
  | simple_expr

simple_expr:
    literal
  | identifier
  | function_call
  | simple_expr COLLATE collation_name
  | param_marker
  | variable
  | simple_expr || simple_expr
  | + simple_expr
  | - simple_expr
  | ~ simple_expr
  | ! simple_expr
  | BINARY simple_expr
  | (expr [, expr] ...)
  | ROW (expr, expr [, expr] ...)
  | (subquery)
  | EXISTS (subquery)
  | {identifier expr}
  | match_expr
  | case_expr
  | interval_expr
```

Notes:

For operator precedence, see in Section 12.3.1, “Operator Precedence”.

For literal value syntax, see Section 9.1, “Literal Values”.

For identifier syntax, see Section 9.2, “Schema Object Names”.

Variables can be user variables, system variables, or stored program local variables or parameters:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- User variables: Section 9.4, “User-Defined Variables”
- System variables: Section 5.1.5, “Using System Variables”
- Local variables: Section 13.6.4.1, “Local Variable DECLARE Syntax”
- Parameters: Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”

param_marker is ? as used in prepared statements for placeholders. See Section 13.5.1, “PREPARE Syntax”.

(*subquery*) indicates a subquery that returns a single value; that is, a scalar subquery. See Section 13.2.9.1, “The Subquery as Scalar Operand”.

{*identifier expr*} is ODBC escape syntax and is accepted for ODBC compatibility. The value is *expr*. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

match_expr indicates a MATCH expression. See Section 12.9, “Full-Text Search Functions”.

case_expr indicates a CASE expression. See Section 12.4, “Control Flow Functions”.

interval_expr represents a time interval. The syntax is INTERVAL *expr unit*, where *unit* is a specifier such as HOUR, DAY, or WEEK. For the full list of *unit* specifiers, see the description of the DATE_ADD() function in Section 12.7, “Date and Time Functions”.

The meaning of some operators depends on the SQL mode:

- By default, || is a logical OR operator. With PIPES_AS_CONCAT enabled, || is string concatenation, with a precedence between ^ and the unary operators.
- By default, ! has a higher precedence than NOT as of MySQL 5.0.2. For earlier versions, or from 5.0.2 on with HIGH_NOT_PRECEDENCE enabled, ! and NOT have the same precedence.

See Section 5.1.7, “Server SQL Modes”.

9.6 Comment Syntax

MySQL Server supports three comment styles:

- From a “#” character to the end of the line.
- From a “-- ” sequence to the end of the line. In MySQL, the “-- ” (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.8.2.5, “'--' as the Start of a Comment”.
- From a /* sequence to the following */ sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
/*
this is a
multiple-line comment
*/
1;
```

Nested comments are not supported.

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the "`!`" character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the `mysqld` server parses SQL statements. The `mysql` client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.)

Comments in this format, `/*!12345 ... */`, are not stored on the server. If this format is used to comment stored routines, the comments will not be retained on the server.

The use of short-form `mysql` commands such as `\C` within multiple-line `/* ... */` comments is not supported.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 10 Globalization

Table of Contents

This chapter covers issues of globalization, which includes internationalization (MySQL's capabilities for adapting to local use) and localization (selecting particular local conventions):

- MySQL support for character sets in SQL statements.
- How to configure the server to support different character sets.
- Selecting the language for error messages.
- How to set the server's time zone and enable per-connection time zone support.
- Selecting the locale for day and month names.

10.1 Character Set Support

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server,

This documentation is for an older version. If you're

This documentation is for an older version. If you're

database, table, and column level. MySQL supports the use of character sets for the `MyISAM`, `MEMORY`, `NDBCLUSTER`, and `InnoDB` storage engines.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment
- Syntax for specifying character sets and collations
- Affected functions and operations
- Unicode support
- The character sets and collations that are available, with notes

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see Section 10.1.5, “Configuring the Character Set and Collation for Applications”, and Section 10.1.4, “Connection Character Sets and Collations”.

10.1.1 Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: “A”, “B”, “a”, “b”. We give each letter a number: “A” = 0, “B” = 1, “a” = 2, “b” = 3. The letter “A” is a symbol, the number 0 is the **encoding** for “A”, and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, “A” and “B”. The simplest way to do this is to look at the encodings: 0 for “A” and 1 for “B”. Because 0 is less than 1, we say “A” is less than “B”. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters “a” and “b” as equivalent to “A” and “B”; (2) then compare the encodings. We call this a *case-insensitive* collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just “A” and “B” but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German “Ö”), and for multiple-character mappings (such as the rule that “Ö” = “OE” in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets
- Compare strings using a variety of collations

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Mix strings with different character sets or collations in the same server, the same database, or even the same table
- Enable specification of character set and collation at any level

In these respects, MySQL is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

10.1.2 Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the `SHOW CHARACTER SET` statement. A partial listing follows. For more complete information, see Section 10.1.13, "Character Sets and Collations That MySQL Supports".

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
...
```

Any given character set always has at least one collation. It may have several collations. To list the collations for a character set, use the `SHOW COLLATION` statement. For example, to see the collations for the `latin1` (cp1252 West European) character set, use this statement to find those collation names that begin with `latin1`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+--------------------+---------+----+---------+----------+---------+
| Collation          | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+----+---------+----------+---------+
| latin1_german1_ci  | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci  | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci   | latin1  | 15 |         |          |       0 |
| latin1_german2_ci  | latin1  | 31 |         | Yes      |       2 |
| latin1_bin         | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci  | latin1  | 48 |         |          |       0 |
| latin1_general_cs  | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci  | latin1  | 94 |         |          |       0 |
+--------------------+---------+----+---------+----------+---------+
```

The `latin1` collations have the following meanings.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Collation	Meaning
latin1_german1_ci	German DIN-1
latin1_swedish_ci	Swedish/Finnish
latin1_danish_ci	Danish/Norwegian
latin1_german2_ci	German DIN-2
latin1_bin	Binary according to latin1 encoding
latin1_general_ci	Multilingual (Western European)
latin1_general_cs	Multilingual (ISO Western European), case sensitive
latin1_spanish_ci	Modern Spanish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for latin1 is latin1_swedish_ci. The output for SHOW CHARACTER SET indicates which collation is the default for each displayed character set.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with _ci (case insensitive), _cs (case sensitive), or _bin (binary).

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

10.1.3 Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

CHARACTER SET is used in clauses that specify a character set. CHARSET can be used as a synonym for CHARACTER SET.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the utf8 Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see Section 10.1.4, “Connection Character Sets and Collations”.

10.1.3.1 Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup on the command line or in an option file and changed at runtime.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Initially, the server character set and collation depend on the options that you use when you start `mysqld`. You can use `--character-set-server` for the character set. Along with it, you can add `--collation-server` for the collation. If you don't specify a character set, that is the same as saying `--character-set-server=latin1`. If you specify only a character set (for example, `latin1`) but not a collation, that is the same as saying `--character-set-server=latin1 --collation-server=latin1_swedish_ci` because `latin1_swedish_ci` is the default collation for `latin1`. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
          --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. If you want to change the default server character set and collation when building from sources, use: `--with-charset` and `--with-collation` as arguments for `configure`. For example:

```
shell> ./configure --with-charset=latin1
```

Or:

```
shell> ./configure --with-charset=latin1 \
          --with-collation=latin1_german1_ci
```

Both `mysqld` and `configure` verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in `CREATE DATABASE` statements. They have no other purpose.

The current server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables. These variables can be changed at runtime.

10.1.3.2 Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

The keyword `SCHEMA` can be used instead of `DATABASE`.

All database options are stored in a text file named `db.opt` that can be found in the database directory.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.
- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *Y* and collation *Y* are used.
- Otherwise, the server character set and server collation are used.

The character set and collation for the default database can be determined from the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` and `collation_server`.

To see the default character set and collation for a given database, use these statements:

```
USE db_name;
SELECT @@character_set_database, @@collation_database;
```

Alternatively, to display the values without changing the default database:

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'db_name';
```

The database character set and collation affect these aspects of server operation:

- For `CREATE TABLE` statements, the database character set and collation are used as default values for table definitions if the table character set and collation are not specified. To override this, provide explicit `CHARACTER SET` and `COLLATE` table options.
- For `LOAD DATA` statements that include no `CHARACTER SET` clause, the server uses the character set indicated by the `character_set_database` system variable to interpret the information in the file. To override this, provide an explicit `CHARACTER SET` clause.
- For stored routines (procedures and functions), the database character set and collation in effect at routine creation time are used as the character set and collation of character data parameters for which the declaration includes no `CHARACTER SET` attribute. To override this, provide an explicit `CHARACTER SET` attribute.

10.1.3.3 Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]]

ALTER TABLE tbl_name
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.
- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *Y* and collation *Y* are used.
- Otherwise, the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.1.3.4 Column Character Set and Collation

Every “character” column (that is, a column of type `CHAR`, `VARCHAR`, or `TEXT`) has a column character set and a column collation. Column definition syntax for `CREATE TABLE` and `ALTER TABLE` has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

These clauses can also be used for `ENUM` and `SET` columns:

```
col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Examples:

```
CREATE TABLE t1
(
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE t1
(
    col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set and collation are specified for the column, so they are used. The column has character set `utf8` and collation `utf8_unicode_ci`.

- If `CHARACTER SET X` is specified without `COLLATE`, character set *`X`* and its default collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10) CHARACTER SET utf8
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set is specified for the column, but the collation is not. The column has character set `utf8` and the default collation for `utf8`, which is `utf8_general_ci`. To see the default collation for each character set, use the `SHOW COLLATION` statement.

- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *`Y`* and collation *`Y`* are used.

```
CREATE TABLE t1
(
    col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The collation is specified for the column, but the character set is not. The column has collation `utf8_polish_ci` and the character set is the one associated with the collation, which is `utf8`.

- Otherwise, the table character set and collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_bin;
```

Neither the character set nor collation are specified for the column, so the table defaults are used. The column has character set `latin1` and collation `latin1_bin`.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

If you use `ALTER TABLE` to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

10.1.3.5 Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

A character string literal may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement `SELECT 'string'`, the string has the character set and collation defined by the `character_set_connection` and `collation_connection` system variables.

The *`_charset_name`* expression is formally called an *introducer*. It tells the parser, “the string that is about to follow uses character set *`X`*.” Because this has confused people in the past, we emphasize that an introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string's value, although padding may occur. The introducer is just a signal. An introducer is also legal before standard hex literal and numeric hex literal notation (`x'literal'` and `0xnnnn`), or before bit-field literal notation (`b'literal'` and `0bnnnn`).

Examples:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 b'1100011';
SELECT _latin1 0b1100011;
```

MySQL determines a literal's character set and collation in the following manner:

- If both *`_X`* and `COLLATE` *`Y`* are specified, character set *`X`* and collation *`Y`* are used.
- If *`_X`* is specified but `COLLATE` is not specified, character set *`X`* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- Otherwise, the character set and collation given by the `character_set_connection` and `collation_connection` system variables are used.

Examples:

- A string with `latin1` character set and `latin1_german1_ci` collation:

  ```
  SELECT _latin1'Müller' COLLATE latin1_german1_ci;
  ```

- A string with `latin1` character set and its default collation (that is, `latin1_swedish_ci`):

  ```
  SELECT _latin1'Müller';
  ```

- A string with the connection default character set and collation:

  ```
  SELECT 'Müller';
  ```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

An introducer indicates the character set for the following string, but does not change now how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`.

The following examples show that escape processing occurs using `character_set_connection` even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection`, as discussed in Section 10.1.4, “Connection Character Sets and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Collations"), and display the resulting strings using the `HEX()` function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+------------+-----------------+
| HEX('à\n') | HEX(_sjis'à\n') |
+------------+-----------------+
| E00A       | E00A            |
+------------+-----------------+
1 row in set (0.00 sec)
```

Here, "`à`" (hex value `E0`) is followed by "`\n`", the escape sequence for newline. The escape sequence is interpreted using the `character_set_connection` value of `latin1` to produce a literal newline (hex value `0A`). This happens even for the second string. That is, the introducer of `_sjis` does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+------------+-------------------+
| HEX('à\n') | HEX(_latin1'à\n') |
+------------+-------------------+
| E05C6E     | E05C6E            |
+------------+-------------------+
1 row in set (0.04 sec)
```

Here, `character_set_connection` is `sjis`, a character set in which the sequence of "`à`" followed by "`\`" (hex values `05` and `5C`) is a valid multibyte character. Hence, the first two bytes of the string are interpreted as a single `sjis` character, and the "`\`" is not interpreted as an escape character. The following "`n`" (hex value `6E`) is not interpreted as part of an escape sequence. This is true even for the second string; the introducer of `_latin1` does not affect escape processing.

10.1.3.6 National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NVARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For information on upgrading character sets to MySQL 5.0 from versions prior to 4.1, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.

10.1.3.7 Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

10.1.3.8 Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.1.4 Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation are the values of the `character_set_server` and `collation_server` system variables.
- The character set and collation of the default database are the values of the `character_set_database` and `collation_database` system variables.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation system variables.

A “connection” is what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets or error messages, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the statement in when it leaves the client?

  The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

- What character set should the server translate a statement to after receiving it?

  For this, the server uses the `character_set_connection` and `collation_connection` system variables. It converts statements sent by the client from `character_set_client` to `character_set_connection` (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence.

- What character set should the server translate to before shipping result sets back to the client?

  The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings *for each connection to the server.*

Two statements affect the connection-related character set variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, "future incoming messages from this client are in character set `cp1251`." It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement.)

  A `SET NAMES 'charset_name'` statement is equivalent to these three statements:

```
SET character_set_client = charset_name;
SET character_set_results = charset_name;
SET character_set_connection = charset_name;
```

  Setting `character_set_connection` to *`charset_name`* also implicitly sets `collation_connection` to the default collation for *`charset_name`*. It is unnecessary to set that collation explicitly. To specify a particular collation, use the optional `COLLATE` clause:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

- `SET CHARACTER SET charset_name`

  `SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` and `collation_connection` to `character_set_database` and `collation_database`. A `SET CHARACTER SET charset_name` statement is equivalent to these three statements:

```
SET character_set_client = charset_name;
SET character_set_results = charset_name;
SET collation_connection = @@collation_database;
```

  Setting `collation_connection` also implicitly sets `character_set_connection` to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is unnecessary to set `character_set_connection` explicitly.

Note

`ucs2` cannot be used as a client character set, which means that it does not work for `SET NAMES` or `SET CHARACTER SET`.

The MySQL client programs `mysql`, `mysqladmin`, `mysqlcheck`, `mysqlimport`, and `mysqlshow` determine the default character set to use as follows:

- In the absence of other information, the programs use the compiled-in default character set, usually `latin1`.
- The programs support a `--default-character-set` option, which enables users to specify the character set explicitly to override whatever default the client otherwise determines.

When a client connects to the server, it sends the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

With the `mysql` client, to use a character set different from the default, you could explicitly execute `SET NAMES` every time you start up. To accomplish the same result more easily, add the `--default-character-set` option setting to your `mysql` command line or in your option file. For example, the following option file setting changes the three connection-related character set variables set to `koi8r` each time you invoke `mysql`:

```
[mysql]
default-character-set=koi8r
```

If you are using the `mysql` client with auto-reconnect enabled (which is not recommended), it is preferable to use the `charset` command rather than `SET NAMES`. For example:

```
mysql> charset utf8
Charset changed
```

The `charset` command issues a `SET NAMES` statement, and also changes the default character set that `mysql` uses when it reconnects after the connection has dropped.

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET latin1` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy if there are characters that are not in both character sets.

If you want the server to perform no conversion of result sets or error messages, set `character_set_results` to `NULL` or `binary`:

```
SET character_set_results = NULL;
```

To see the values of the character set and collation system variables that apply to your connection, use these statements:

```
SHOW VARIABLES LIKE 'character_set%';
SHOW VARIABLES LIKE 'collation%';
```

You must also consider the environment within which your MySQL applications execute. See Section 10.1.5, “Configuring the Character Set and Collation for Applications”.

For more information about character sets and error messages, see Section 10.1.6, “Character Set for Error Messages”.

10.1.5 Configuring the Character Set and Collation for Applications

For applications that store data using the default MySQL character set and collation (`latin1`, `latin1_swedish_ci`), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might require `utf8`, whereas applications that use another database might require `sjis`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.
- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the `utf8` character set and `utf8_general_ci` collation.

Specify character settings per database. To create a database such that its tables will use a given default character set and collation for data storage, use a `CREATE DATABASE` statement like this:

```
CREATE DATABASE mydb
  DEFAULT CHARACTER SET utf8
  DEFAULT COLLATE utf8_general_ci;
```

Tables created in the database will use `utf8` and `utf8_general_ci` by default for any character columns.

Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a `SET NAMES 'utf8'` statement after connecting. The statement can be used regardless of connection method: The `mysql` client, PHP scripts, and so forth.

In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, for connections made using `mysql`, you can specify the `--default-character-set=utf8` command-line option to achieve the same effect as `SET NAMES 'utf8'`.

For more information about configuring client connections, see Section 10.1.4, “Connection Character Sets and Collations”.

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)

Specify character settings at server startup. To select a character set and collation at server startup, use the `--character-set-server` and `--collation-server` options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
```

These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

It is still necessary for applications to configure their connection using `SET NAMES` or equivalent after they connect, as described previously. You might be tempted to start the server with the `--`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`init_connect="SET NAMES 'utf8'"` option to cause `SET NAMES` to be executed automatically for each client that connects. However, this will yield inconsistent results because the `init_connect` value is not executed for users who have the `SUPER` privilege.

Specify character settings at MySQL configuration time. To select a character set and collation when you configure and build MySQL from source, use the `--with-charset` and `--with-collation` options:

```
shell> ./configure --with-charset=utf8 --with-collation=utf8_general_ci
```

The resulting server uses `utf8` and `utf8_general_ci` as the default for databases and tables and for client connections. It is unnecessary to use `--character-set-server` and `--collation-server` to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using `SET NAMES` or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. If you will send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

10.1.6 Character Set for Error Messages

This section describes how the server uses character sets for constructing error messages and returning them to clients. For information about the language of error messages (rather than the character set), see Section 10.2, “Setting the Error Message Language”.

In MySQL, the server constructs error messages and returns them to clients as follows:

- The message template has the character set associated with the error message language. For example, English, Korean, and Russian messages use `latin1`, `euckr`, and `koi8r`, respectively.
- Parameters in the message template are replaced with values that apply to a specific error occurrence. These parameters use their own character set. Identifiers such as table or column names use UTF-8. Data values retain their character set. For example, in the following duplicate-key message, `'xxx'` has the character set of the table column associated with key 1:

  ```
  Duplicate entry 'xxx' for key1
  ```

The preceding method of error-message construction can result in messages that contain a mix of character sets unless all items involved contain only ASCII characters. This issue is resolved in MySQL 5.5, in which error messages are constructed internally within the server using UTF-8 and returned to the client in the character set specified by the `character_set_results` system variable.

10.1.7 Collation Issues

The following sections discuss various aspects of character set collations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

10.1.7.1 Collation Naming Conventions

MySQL collation names follow these conventions:

- A name ending in `_ci` indicates a case-insensitive collation.
- A name ending in `_cs` indicates a case-sensitive collation.
- A name ending in `_bin` indicates a binary collation. Character comparisons are based on character binary code values.

10.1.7.2 Using COLLATE in SQL Statements

With the `COLLATE` clause, you can override whatever the default collation is for a comparison. `COLLATE` may be used in various parts of SQL statements. Here are some examples:

- With `ORDER BY`:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With `AS`:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- With `GROUP BY`:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With `DISTINCT`:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With `WHERE`:

```
    SELECT *
    FROM t1
    WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
    SELECT *
    FROM t1
    WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With `HAVING`:

This documentation is for an older version. If you're

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.1.7.3 COLLATE Clause Precedence

The `COLLATE` clause has high precedence (higher than `||`), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```

10.1.7.4 Collations Must Be for the Right Character Set

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.1.7.5 Collation of Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column `charset_name`:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should the comparison use the collation of the column `x`, or of the string literal `'Y'`? Both `x` and `'Y'` have collations, so which collation takes precedence?

Standard SQL resolves such questions using what used to be called “coercibility” rules. MySQL assigns coercibility values as follows:

- An explicit `COLLATE` clause has a coercibility of 0. (Not coercible at all.)
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` or `VERSION()`) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `NULL` or an expression that is derived from `NULL` has a coercibility of 5.

The preceding coercibility values are current as of MySQL 5.0.3. Prior to 5.0.3, there is no system constant or `NULL` coercibility. Functions such as `USER()` have a coercibility of 2 rather than 3, and literals have a coercibility of 3 rather than 4.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:
  - If both sides are Unicode, or both sides are not Unicode, it is an error.
  - If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

    ```
    SELECT CONCAT(utf8_column, latin1_column) FROM t1;
    ```

    It returns a result that has a character set of `utf8` and the same collation as `utf8_column`. Values of `latin1_column` are automatically converted to `utf8` before concatenating.
  - For an operation with operands from the same character set but that mix a `_bin` collation and a `_ci` or `_cs` collation, the `_bin` collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, except that it is for collations rather than data types.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

Examples:

Comparison	Collation Used
`column1 = 'A'`	Use collation of `column1`
`column1 = 'A' COLLATE x`	Use collation of `'A' COLLATE x`
`column1 COLLATE x = 'A' COLLATE y`	Error

The `COERCIBILITY()` function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
mysql> SELECT COERCIBILITY('A');
        -> 4
```

See Section 12.13, “Information Functions”.

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument `1` in the expression `CONCAT(1, 'abc')`, the result is a binary string for which the character set and collation are `binary`. See Section 12.2, “Type Conversion in Expression Evaluation”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

10.1.7.6 The _bin and binary Collations

This section describes how `_bin` collations for nonbinary strings differ from the `binary` “collation” for binary strings.

Nonbinary strings (as stored in the `CHAR`, `VARCHAR`, and `TEXT` data types) have a character set and collation. A given character set can have several collations, each of which defines a particular sorting and comparison order for the characters in the set. One of these is the binary collation for the character set, indicated by a `_bin` suffix in the collation name. For example, `latin1` and `utf8` have binary collations named `latin1_bin` and `utf8_bin`.

Binary strings (as stored in the `BINARY`, `VARBINARY`, and `BLOB` data types) have no character set or collation in the sense that nonbinary strings do. (Applied to a binary string, the `CHARSET()` and `COLLATION()` functions both return a value of `binary`.) Binary strings are sequences of bytes and the numeric values of those bytes determine sort order.

The `_bin` collations differ from the `binary` collation in several respects.

The unit for sorting and comparison. Binary strings are sequences of bytes. Sorting and comparison is always based on numeric byte values. Nonbinary strings are sequences of characters, which might be multibyte. Collations for nonbinary strings define an ordering of the character values for sorting and comparison. For the `_bin` collation, this ordering is based solely on binary code values of the characters (which is similar to ordering for binary strings except that a `_bin` collation must take into account that a character might contain multiple bytes). For other collations, character ordering might take additional factors such as lettercase into account.

Character set conversion. A nonbinary string has a character set and is converted to another character set in many cases, even when the string has a `_bin` collation:

- When assigning column values from another column that has a different character set:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- When assigning column values for `INSERT` or `UPDATE` using a string literal:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- When sending results from the server to a client:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

For binary string columns, no conversion occurs. For the preceding cases, the string value is copied byte-wise.

Lettercase conversion. Collations provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for `_bin` collations that ignore lettercase for ordering:

```
mysql> SET NAMES latin1 COLLATE latin1_bin;
Query OK, 0 rows affected (0.02 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT LOWER('aA'), UPPER('zZ');
+-------------+-------------+
| LOWER('aA') | UPPER('zZ') |
+-------------+-------------+
| aa          | ZZ          |
+-------------+-------------+
1 row in set (0.13 sec)
```

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must be converted to a nonbinary string:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING latin1));
+-------------+----------------------------------+
| LOWER('aA') | LOWER(CONVERT('aA' USING latin1)) |
+-------------+----------------------------------+
| aA          | aa                               |
+-------------+----------------------------------+
1 row in set (0.00 sec)
```

Trailing space handling in comparisons. Nonbinary strings have `PADSPACE` behavior for all collations, including `_bin` collations. Trailing spaces are insignificant in comparisons:

```
mysql> SET NAMES utf8 COLLATE utf8_bin;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          1 |
+------------+
1 row in set (0.00 sec)
```

For binary strings, all characters are significant in comparisons, including trailing spaces:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
1 row in set (0.00 sec)
```

Trailing space handling for inserts and retrievals. `CHAR(N)` columns store nonbinary strings. Values shorter than *N* characters are extended with spaces on insertion. For retrieval, trailing spaces are removed.

`BINARY(N)` columns store binary strings. Values shorter than *N* bytes are extended with `0x00` bytes on insertion. For retrieval, nothing is removed; a value of the declared length is always returned.

```
mysql> CREATE TABLE t1 (
    ->   a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
    ->   b BINARY(10)
    -> );
Query OK, 0 rows affected (0.09 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> INSERT INTO t1 VALUES ('a','a');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(a), HEX(b) FROM t1;
+--------+----------------------+
| HEX(a) | HEX(b)               |
+--------+----------------------+
| 61     | 6100000000000000000 |
+--------+----------------------+
1 row in set (0.04 sec)
```

10.1.7.7 The BINARY Operator

The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a comparison to be done byte by byte rather than character by character. `BINARY` also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

`BINARY` *`str`* is shorthand for `CAST(`*`str`* `AS BINARY)`.

The `BINARY` attribute in character column definitions has a different effect. A character column defined with the `BINARY` attribute is assigned the binary collation of the column character set. Every character set has a binary collation. For example, the binary collation for the `latin1` character set is `latin1_bin`, so if the table default character set is `latin1`, these two column definitions are equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

The use of `CHARACTER SET binary` in the definition of a `CHAR`, `VARCHAR`, or `TEXT` column causes the column to be treated as a binary data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

10.1.7.8 Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column `X` in table `T` has these `latin1` column values:

```
Muffler
Müller
MX Systems
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The following table shows the resulting order of the values if we use `ORDER BY` with different collations.

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
    ->   c CHAR(10)
    -> ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
    ->   c CHAR(10)
    -> ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
    ->   c CHAR(10)
    -> ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
```


This documentation is for an older version. If you're

```
| Bär  |
+------+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bär  |
+------+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
```

This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

10.1.8 String Repertoire

The *repertoire* of a character set is the collection of characters in the set.

As of MySQL 5.0.48, string expressions have a repertoire attribute, which can have two values:

- `ASCII`: The expression can contain only characters in the Unicode range `U+0000` to `U+007F`.
- `UNICODE`: The expression can contain characters in the Unicode range `U+0000` to `U+FFFF`.

The `ASCII` range is a subset of `UNICODE` range, so a string with `ASCII` repertoire can be converted safely without loss of information to the character set of any string with `UNICODE` repertoire or to a character set that is a superset of `ASCII`. (All MySQL character sets are supersets of `ASCII` with the exception of `swe7`, which reuses some punctuation characters for Swedish accented characters.) The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an “illegal mix of collations” error.

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for string constants depends on string content:

  ```
  SET NAMES utf8; SELECT 'abc';
  SELECT _utf8'def';
  SELECT N'MySQL';
  ```

  Although the character set is `utf8` in each of the preceding cases, the strings do not actually contain any characters outside the ASCII range, so their repertoire is `ASCII` rather than `UNICODE`.

- Columns having the `ascii` character set have `ASCII` repertoire because of their character set. In the following table, `c1` has `ASCII` repertoire:

  ```
  CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
  ```

  The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

  ```
  CREATE TABLE t1 (
    c1 CHAR(1) CHARACTER SET latin1,
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  c2 CHAR(1) CHARACTER SET ascii
);
INSERT INTO t1 VALUES ('a','b');
SELECT CONCAT(c1,c2) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (ascii_general_ci,IMPLICIT) for operation 'concat'
```

Using repertoire, subset to superset (`ascii` to `latin1`) conversion can occur and a result is returned:

```
+---------------+
| CONCAT(c1,c2) |
+---------------+
| ab            |
+---------------+
```

- Functions with one string argument inherit the repertoire of their argument. The result of `UPPER(_utf8'abc')` has `ASCII` repertoire, because its argument has `ASCII` repertoire.
- For functions that return a string but do not have string arguments and use `character_set_connection` as the result character set, the result repertoire is `ASCII` if `character_set_connection` is `ascii`, and `UNICODE` otherwise:

```
FORMAT(numeric_column, 4);
```

Use of repertoire changes how MySQL evaluates the following example:

```
SET NAMES ascii;
CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
INSERT INTO t1 VALUES (1,'b');
SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-------------------------+
| CONCAT(FORMAT(a, 4), b) |
+-------------------------+
| 1.0000b                 |
+-------------------------+
```

- Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire (`UNICODE` is wider than `ASCII`). Consider the following `CONCAT()` calls:

```
CONCAT(_ucs2 X'0041', _ucs2 X'0042')
CONCAT(_ucs2 X'0041', _ucs2 X'00C2')
```

For the first call, the repertoire is `ASCII` because both arguments are within the range of the `ascii` character set. For the second call, the repertoire is `UNICODE` because the second argument is outside the `ascii` character set range.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The repertoire for function return values is determined based only on the repertoire of the arguments that affect the result's character set and collation.

```
IF(column1 < column2, 'smaller', 'greater')
```

  The result repertoire is `ASCII` because the two string arguments (the second argument and the third argument) both have `ASCII` repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

10.1.9 Operations Affected by Character Set Support

This section describes operations that take character set information into account.

10.1.9.1 Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` returns a string whose character string and collation are the same as that of *X*. The same applies for `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, and `UPPER()`.

Note: The `REPLACE()` function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has no character set or collation. This can be checked by using the `CHARSET()` and `COLLATION()` functions, both of which return `binary` to indicate that their argument is a binary string:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+--------------------+----------------------+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+--------------------+----------------------+
| binary              | binary               |
+--------------------+----------------------+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE X` occurs, use *X*.
- If explicit `COLLATE X` and `COLLATE Y` occur, raise an error.
- Otherwise, if all collations are *X*, use *X*.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is *X*. The same applies for `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables. This applies only to `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, `SPACE()`. Before MySQL 5.0.15, it also applies to `CHAR()`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you are uncertain about the character set or collation of the result returned by a string function, you can use the `CHARSET()` or `COLLATION()` function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+---------------+-----------------+-------------------+
| USER()        | CHARSET(USER()) | COLLATION(USER()) |
+---------------+-----------------+-------------------+
| test@localhost | utf8            | utf8_general_ci   |
+---------------+-----------------+-------------------+
```

10.1.9.2 CONVERT() and CAST()

`CONVERT()` provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` is implemented according to the standard SQL specification.

You may also use `CAST()` to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use `CAST()` without specifying `CHARACTER SET`, the resulting character set and collation are defined by the `character_set_connection` and `collation_connection` system variables. If you use `CAST()` with `CHARACTER SET X`, the resulting character set and collation are `X` and the default collation of `X`.

You may not use a `COLLATE` clause inside a `CONVERT()` or `CAST()` call, but you may use it outside. For example, `CAST(... COLLATE ...)` is illegal, but `CAST(...) COLLATE ...` is legal:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

10.1.9.3 SHOW Statements and INFORMATION_SCHEMA

Several `SHOW` statements provide additional character set information. These include `SHOW CHARACTER SET`, `SHOW COLLATION`, `SHOW CREATE DATABASE`, `SHOW CREATE TABLE` and `SHOW COLUMNS`. These statements are described here briefly. For more information, see Section 13.7.5, “SHOW Syntax”.

`INFORMATION_SCHEMA` has several tables that contain information similar to that displayed by the `SHOW` statements. For example, the `CHARACTER_SETS` and `COLLATIONS` tables contain the information displayed by `SHOW CHARACTER SET` and `SHOW COLLATION`. See Chapter 19, *INFORMATION_SCHEMA Tables*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `SHOW CHARACTER SET` statement shows all available character sets. It takes an optional `LIKE` clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

`SHOW CREATE DATABASE` displays the `CREATE DATABASE` statement that creates a given database:

```
mysql> SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

If no `COLLATE` clause is shown, the default collation for the character set applies.

`SHOW CREATE TABLE` is similar, but displays the `CREATE TABLE` statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

The `SHOW COLUMNS` statement displays the collations of a table's columns when invoked as `SHOW FULL COLUMNS`. Columns with `CHAR`, `VARCHAR`, or `TEXT` data types have collations. Numeric and other noncharacter types have no collation (indicated by `NULL` as the `Collation` value). For example:

```
mysql> SHOW FULL COLUMNS FROM person\G
*************************** 1. row ***************************
     Field: id
      Type: smallint(5) unsigned
 Collation: NULL
      Null: NO
       Key: PRI
   Default: NULL
     Extra: auto_increment
Privileges: select,insert,update,references
   Comment:
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
*************************** 2. row ***************************
     Field: name
      Type: char(60)
 Collation: latin1_swedish_ci
      Null: NO
       Key:
   Default:
     Extra:
Privileges: select,insert,update,references
   Comment:
```

The character set is not part of the display but is implied by the collation name.

10.1.10 Unicode Support

MySQL supports two character sets for storing Unicode data:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character.

These two character sets support the characters from the Basic Multilingual Plane (BMP) of Unicode Version 3.0. BMP characters have these characteristics:

- Their code values are between 0 and 65535 (or `U+0000` .. `U+FFFF`).
- They can be encoded with a fixed 16-bit word, as in `ucs2`.
- They can be encoded with 8, 16, or 24 bits, as in `utf8`.
- They are sufficient for almost all characters in major languages.

The `ucs2` and `utf8` character sets do not support supplementary characters that lie outside the BMP. Characters outside the BMP compare as REPLACEMENT CHARACTER and convert to `'?'` when converted to a Unicode character set.

A similar set of collations is available for each Unicode character set. For example, each has a Danish collation, the names of which are `ucs2_danish_ci` and `utf8_danish_ci`. All Unicode collations are listed at Section 10.1.13.1, “Unicode Character Sets”.

The MySQL implementation of UCS-2 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of values will need to be performed when transferring data between those systems and MySQL.

MySQL uses no BOM for UTF-8 values.

Client applications that need to communicate with the server using Unicode should set the client character set accordingly; for example, by issuing a `SET NAMES 'utf8'` statement. `ucs2` cannot be used as a client character set, which means that it does not work for `SET NAMES` or `SET CHARACTER SET`. (See Section 10.1.4, “Connection Character Sets and Collations”.)

The following sections provide additional detail on the Unicode character sets in MySQL.

10.1.10.1 The ucs2 Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a 2-byte Unicode code with the most significant byte first. For example: `LATIN CAPITAL LETTER A` has the code `0x0041` and it is stored as a 2-byte sequence: `0x00`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`0x41`. `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) is stored as a 2-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the Unicode Home Page.

In MySQL, the `ucs2` character set is a fixed-length 16-bit encoding for Unicode BMP characters.

10.1.10.2 The utf8 Character Set (3-Byte UTF-8 Unicode Encoding)

UTF-8 (Unicode Transformation Format with 8-bit units) is an alternative way to store Unicode data. It is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. MySQL support for UTF-8 does not include 4-byte sequences. (An older standard for UTF-8 encoding, RFC 2279, describes UTF-8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete; for this reason, sequences with five and six bytes are no longer used.)

The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use 3-byte sequences.

Tip: To save space with UTF-8, use `VARCHAR` instead of `CHAR`. Otherwise, MySQL must reserve three bytes for each character in a `CHAR CHARACTER SET utf8` column because that is the maximum possible length. For example, MySQL must reserve 30 bytes for a `CHAR(10) CHARACTER SET utf8` column.

For additional information about data type storage, see Section 11.7, “Data Type Storage Requirements”. For information about `InnoDB` physical row storage, including how `InnoDB` tables that use `COMPACT` row format handle UTF-8 `CHAR(N)` columns internally, see Section 14.2.10.5, “Physical Row Structure”.

10.1.11 UTF-8 for Metadata

Metadata is “the data about the data.” Anything that *describes* the database—as opposed to being the *contents* of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from `SHOW` are metadata. This is also true of the contents of tables in `INFORMATION_SCHEMA` because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the `SHOW` statements nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()`, `CURRENT_USER()`, `SESSION_USER()`, `SYSTEM_USER()`, `DATABASE()`, and `VERSION()` functions have the UTF-8 character set by default.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` system variable, which has a default value of `latin1`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` and other related system variables. (See Section 10.1.4, “Connection Character Sets and Collations”.) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client to perform the conversion, but this option is not always available for all clients.

If `character_set_results` is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see Section 10.1.7.5, “Collation of Expressions”.

10.1.12 Column Character Set Conversion

To convert a binary or nonbinary string column to use a particular character set, use `ALTER TABLE`. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (`BINARY`, `VARBINARY`, `BLOB`), all the values that it contains must be encoded using a single character set (the character set you're converting the column to). If you use a binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If the column has a nonbinary data type (`CHAR`, `VARCHAR`, `TEXT`), its contents should be encoded in the column character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a nonbinary column with the desired character set.

Suppose that a table `t` has a binary column named `col1` defined as `VARBINARY(50)`. Assuming that the information in the column is encoded using a single character set, you can convert it to a nonbinary column that has that character set. For example, if `col1` contains binary data representing characters in the `greek` character set, you can convert it as follows:

```
ALTER TABLE t MODIFY col1 VARCHAR(50) CHARACTER SET greek;
```

If your original column has a type of `BINARY(50)`, you could convert it to `CHAR(50)`, but the resulting values will be padded with `0x00` bytes at the end, which may be undesirable. To remove these bytes, use the `TRIM()` function:

```
UPDATE t SET col1 = TRIM(TRAILING 0x00 FROM col1);
```

Suppose that table `t` has a nonbinary column named `col1` defined as `CHAR(50) CHARACTER SET latin1` but you want to convert it to use `utf8` so that you can store values from many languages. The following statement accomplishes this:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from before MySQL 4.1 where a nonbinary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored `sjis` values in a column, even though MySQL's default character set was `latin1`. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was `latin1` and `col1` is defined as `CHAR(50)` but its contents are `sjis` values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```
ALTER TABLE t MODIFY col1 BLOB;
```

The next step is to convert the column to a nonbinary data type with the proper character set:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as `INSERT` or `UPDATE` after an upgrade to MySQL 4.1 or later. In that case, MySQL would store new values in the column using `latin1`, and the column will contain a mix of `sjis` and `latin1` values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with `ALTER TABLE`. For example, if you specified `NOT NULL` and an explicit `DEFAULT` value, you should also provide them in the `ALTER TABLE` statement. Otherwise, the resulting column definition will not include those attributes.

To convert all character columns in a table, the `ALTER TABLE ... CONVERT TO CHARACTER SET` *`charset`* statement may be useful. See Section 13.1.4, “ALTER TABLE Syntax”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

10.1.13 Character Sets and Collations That MySQL Supports

MySQL supports 70+ collations for 30+ character sets. This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.

You can always list the available character sets and their default collations with the `SHOW CHARACTER SET` statement:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+
| Charset  | Description                 | Default collation   |
+----------+-----------------------------+---------------------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |
| dec8     | DEC West European           | dec8_swedish_ci     |
| cp850    | DOS West European           | cp850_general_ci    |
| hp8      | HP West European            | hp8_english_ci      |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |
| latin1   | cp1252 West European        | latin1_swedish_ci   |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |
| ascii    | US ASCII                    | ascii_general_ci    |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |
| cp1250   | Windows Central European    | cp1250_general_ci   |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |
| cp866    | DOS Russian                 | cp866_general_ci    |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |
| macce    | Mac Central European        | macce_general_ci    |
| macroman | Mac West European           | macroman_general_ci |
| cp852    | DOS Central European        | cp852_general_ci    |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |
| cp1256   | Windows Arabic              | cp1256_general_ci   |
| cp1257   | Windows Baltic              | cp1257_general_ci   |
| binary   | Binary pseudo charset       | binary              |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |
+----------+-----------------------------+---------------------+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

10.1.13.1 Unicode Character Sets

MySQL has two Unicode character sets:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character.

You can store text in about 650 languages using these character sets. This section lists the collations available for each Unicode character set and describes their differentiating properties. For general information about the character sets, see Section 10.1.10, “Unicode Support”.

A similar set of collations is available for each Unicode character set. These are shown in the following list, where *`xxx`* represents the character set name. For example, *`xxx`*`_danish_ci` represents the Danish collations, the specific names of which are `ucs2_danish_ci` and `utf8_danish_ci`.

- *`xxx`*`_bin`
- *`xxx`*`_czech_ci`
- *`xxx`*`_danish_ci`
- *`xxx`*`_esperanto_ci`
- *`xxx`*`_estonian_ci`
- *`xxx`*`_general_ci` (default)
- *`xxx`*`_hungarian_ci`
- *`xxx`*`_icelandic_ci`
- *`xxx`*`_latvian_ci`
- *`xxx`*`_lithuanian_ci`
- *`xxx`*`_persian_ci`
- *`xxx`*`_polish_ci`
- *`xxx`*`_roman_ci`
- *`xxx`*`_romanian_ci`
- *`xxx`*`_slovak_ci`
- *`xxx`*`_slovenian_ci`
- *`xxx`*`_spanish_ci`
- *`xxx`*`_spanish2_ci`
- *`xxx`*`_swedish_ci`
- *`xxx`*`_turkish_ci`
- *`xxx`*`_unicode_ci`

The *`xxx`*`_esperanto_ci` collations were added in MySQL 5.0.13. The *`xxx`*`_hungarian_ci` collations were added in MySQL 5.0.19.

MySQL implements the *`xxx`*`_unicode_ci` collations according to the Unicode Collation Algorithm (UCA) described at http://www.unicode.org/reports/tr10/. The collation uses the version-4.0.0 UCA weight keys: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. The *`xxx`*`_unicode_ci` collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo. A combined character will be considered different from the same character written with a single unicode character in string comparisons, and the two characters are considered to have a different length (for example, as returned by the `CHAR_LENGTH()` function or in result set metadata).

MySQL implements language-specific Unicode collations only if the ordering with *`xxx`*`_unicode_ci` does not work well for a language. Language-specific collations are UCA-based. They are derived from *`xxx`*`_unicode_ci` with additional language tailoring rules.

For any Unicode character set, operations performed using the *`xxx`*`_general_ci` collation are faster than those for the *`xxx`*`_unicode_ci` collation. For example, comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`. The reason for this is that `utf8_unicode_ci` supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, in German and some other languages “ß” is equal to “`ss`”. `utf8_unicode_ci` also supports contractions and ignorable characters. `utf8_general_ci` is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci` (for the effect this has in comparisons or when doing searches, see Section 10.1.7.8, “Examples of the Effect of Collation”):

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for `utf8_general_ci`:

```
ß = s
```

Whereas this is true for `utf8_unicode_ci`, which supports the German DIN-1 ordering (also known as dictionary order):

```
ß = ss
```

MySQL implements language-specific collations for the `utf8` character set only if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German dictionary order and French, so there is no need to create special `utf8` collations.

`utf8_general_ci` also is satisfactory for both German and French, except that “ß” is equal to “`s`”, and not to “`ss`”. If this is acceptable for your application, you should use `utf8_general_ci` because it is faster. Otherwise, use `utf8_unicode_ci` because it is more accurate.

`xxx``_swedish_ci` includes Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

The *`xxx`*`_spanish_ci` and *`xxx`*`_spanish2_ci` collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, “ñ” (n-tilde) is a separate letter between “`n`” and “`o`”. In addition, for traditional Spanish, “`ch`” is a separate letter between “`c`” and “`d`”, and “`ll`” is a separate letter between “`l`” and “`m`”

The *`xxx`*`_spanish2_ci` collations may also be used for Asturian and Galician.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The *xxx*_danich_ci collations may also be used for Norwegian.

In the *xxx*_roman_ci collations, I and J compare as equal, and U and V compare as equal.

For additional information about Unicode collations in MySQL, see Collation-Charts.Org (utf8).

10.1.13.2 West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- ascii (US ASCII) collations:
  - ascii_bin
  - ascii_general_ci (default)
- cp850 (DOS West European) collations:
  - cp850_bin
  - cp850_general_ci (default)
- dec8 (DEC Western European) collations:
  - dec8_bin
  - dec8_swedish_ci (default)
- hp8 (HP Western European) collations:
  - hp8_bin
  - hp8_english_ci (default)
- latin1 (cp1252 West European) collations:
  - latin1_bin
  - latin1_danish_ci
  - latin1_general_ci
  - latin1_general_cs
  - latin1_german1_ci
  - latin1_german2_ci
  - latin1_spanish_ci
  - latin1_swedish_ci (default)

  latin1 is the default character set. MySQL's latin1 is the same as the Windows cp1252 character set. This means it is the same as the official ISO 8859-1 or IANA (Internet Assigned Numbers Authority) latin1, except that IANA latin1 treats the code points between 0x80 and 0x9f as "undefined," whereas cp1252, and therefore MySQL's latin1, assign characters for those positions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For example, `0x80` is the Euro sign. For the “undefined” entries in `cp1252`, MySQL translates `0x81` to Unicode `0x0081`, `0x8d` to `0x008d`, `0x8f` to `0x008f`, `0x90` to `0x0090`, and `0x9d` to `0x009d`.

The `latin1_swedish_ci` collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.” For an example of the effect this has in comparisons or when doing searches, see Section 10.1.7.8, “Examples of the Effect of Collation”.

- `latin1_german1_ci` (dictionary) rules:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- `latin1_german2_ci` (phone-book) rules:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

In the `latin1_spanish_ci` collation, “ñ” (n-tilde) is a separate letter between “n” and “o”.

- `macroman` (Mac West European) collations:
  - `macroman_bin`
  - `macroman_general_ci` (default)
- `swe7` (7bit Swedish) collations:
  - `swe7_bin`
  - `swe7_swedish_ci` (default)

For additional information about Western European collations in MySQL, see Collation-Charts.Org (ascii, cp850, dec8, hp8, latin1, macroman, swe7).

10.1.13.3 Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- `cp1250` (Windows Central European) collations:
  - `cp1250_bin`
  - `cp1250_croatian_ci`
  - `cp1250_czech_cs`
  - `cp1250_general_ci` (default)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `cp852` (DOS Central European) collations:
  - `cp852_bin`
  - `cp852_general_ci` (default)
- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
  - `keybcs2_bin`
  - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_cs`
  - `latin2_general_ci` (default)
  - `latin2_hungarian_ci`
- `macce` (Mac Central European) collations:
  - `macce_bin`
  - `macce_general_ci` (default)

For additional information about Central European collations in MySQL, see Collation-Charts.Org (cp1250, cp852, keybcs2, latin2, macce).

10.1.13.4 South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- `armscii8` (ARMSCII-8 Armenian) collations:
  - `armscii8_bin`
  - `armscii8_general_ci` (default)
- `cp1256` (Windows Arabic) collations:
  - `cp1256_bin`
  - `cp1256_general_ci` (default)
- `geostd8` (GEOSTD8 Georgian) collations:
  - `geostd8_bin`
  - `geostd8_general_ci` (default)
- `greek` (ISO 8859-7 Greek) collations:
  - `greek_bin`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - `greek_general_ci` (default)
- `hebrew` (ISO 8859-8 Hebrew) collations:
  - `hebrew_bin`
  - `hebrew_general_ci` (default)
- `latin5` (ISO 8859-9 Turkish) collations:
  - `latin5_bin`
  - `latin5_turkish_ci` (default)

For additional information about South European and Middle Eastern collations in MySQL, see Collation-Charts.Org (armscii8, cp1256, geostd8, greek, hebrew, latin5).

10.1.13.5 Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- `cp1257` (Windows Baltic) collations:
  - `cp1257_bin`
  - `cp1257_general_ci` (default)
  - `cp1257_lithuanian_ci`
- `latin7` (ISO 8859-13 Baltic) collations:
  - `latin7_bin`
  - `latin7_estonian_cs`
  - `latin7_general_ci` (default)
  - `latin7_general_cs`

For additional information about Baltic collations in MySQL, see Collation-Charts.Org (cp1257, latin7).

10.1.13.6 Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- `cp1251` (Windows Cyrillic) collations:
  - `cp1251_bin`
  - `cp1251_bulgarian_ci`
  - `cp1251_general_ci` (default)
  - `cp1251_general_cs`
  - `cp1251_ukrainian_ci`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `cp866` (DOS Russian) collations:
  - `cp866_bin`
  - `cp866_general_ci` (default)
- `koi8r` (KOI8-R Relcom Russian) collations:
  - `koi8r_bin`
  - `koi8r_general_ci` (default)
- `koi8u` (KOI8-U Ukrainian) collations:
  - `koi8u_bin`
  - `koi8u_general_ci` (default)

For additional information about Cyrillic collations in MySQL, see Collation-Charts.Org (cp1251, cp866, koi8r, koi8u).).

10.1.13.7 Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See The cp932 Character Set, for additional information about the `cp932` and `sjis` character sets.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”.

- `big5` (Big5 Traditional Chinese) collations:
  - `big5_bin`
  - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
  - `cp932_bin`
  - `cp932_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
  - `eucjpms_bin`
  - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
  - `euckr_bin`
  - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
  - `gb2312_bin`
  - `gb2312_chinese_ci` (default)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `gbk` (GBK Simplified Chinese) collations:
  - `gbk_bin`
  - `gbk_chinese_ci` (default)
- `sjis` (Shift-JIS Japanese) collations:
  - `sjis_bin`
  - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
  - `tis620_bin`
  - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
  - `ujis_bin`
  - `ujis_japanese_ci` (default)

The `big5_chinese_ci` collation sorts on number of strokes.

For additional information about Asian collations in MySQL, see Collation-Charts.Org (big5, cp932, eucjpms, euckr, gb2312, gbk, sjis, tis620, ujis).

The cp932 Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See http://www.iana.org/assignments/character-sets.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems. It is available as of MySQL 5.0.3.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Because MySQL supports character set conversion, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does `cp932` differ from `sjis`?

The `cp932` character set differs from `sjis` in the following ways:

- `cp932` supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some `cp932` characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to `cp932`, one of the code points must be selected. For this "round trip conversion," the rule recommended by Microsoft is used. (See http://support.microsoft.com/kb/170559/EN-US/.)

  The conversion rule works like this:

  - If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
  - If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
  - If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

  The table shown at https://msdn.microsoft.com/en-us/goglobal/cc305152.aspx provides information about the Unicode values of `cp932` characters. For `cp932` table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of `cp932` character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the `cp932` characters that begin with those digits.

  The following links are of special interest. They correspond to the encodings for the following sets of characters:

  - NEC special characters (lead byte `0x87`):

    ```
    https://msdn.microsoft.com/en-us/goglobal/gg674964
    ```

  - NEC selected—IBM extended characters (lead byte `0xED` and `0xEE`):

    ```
    https://msdn.microsoft.com/en-us/goglobal/gg671837
    https://msdn.microsoft.com/en-us/goglobal/gg671838
    ```

  - IBM selected characters (lead byte `0xFA`, `0xFB`, `0xFC`):

    ```
    https://msdn.microsoft.com/en-us/goglobal/gg671839
    https://msdn.microsoft.com/en-us/goglobal/gg671840
    https://msdn.microsoft.com/en-us/goglobal/gg671841
    ```

- Starting from version 5.0.3, `cp932` supports conversion of user-defined characters in combination with `eucjpms`, and solves the problems with `sjis`/`ujis` conversion. For details, please refer to http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html.

For some characters, conversion to and from `ucs2` is different for `sjis` and `cp932`. The following tables illustrate these differences.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Conversion to ucs2:

sjis/cp932 Value	sjis -> ucs2 Conversion	cp932 -> ucs2 Conversion
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from ucs2:

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` (or `--skip-character-set-client-handshake`) has an important effect. See Section 5.1.3, “Server Command Options”.

10.2 Setting the Error Message Language

By default, mysqld produces error messages in English, but they can also be displayed in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can select which language the server uses for error messages using the instructions in this section.

To start `mysqld` with a particular language for error messages, use the `--language` or `-L` option. The option value can be a language name or the full path to the error message file. For example:

```
shell> mysqld --language=swedish
```

Or:

```
shell> mysqld --language=/usr/local/share/swedish
```

The language name should be specified in lowercase.

By default, the language files are located in the `share/mysql/LANGUAGE` directory under the MySQL base directory.

For information about changing the character set for error messages (rather than the language), see Section 10.1.6, “Character Set for Error Messages”.

You can change the content of the error messages produced by the server using the instructions in the MySQL Internals manual, available at MySQL Internals: Error Messages. If you do change the content of error messages, remember to repeat your changes after each upgrade to a newer version of MySQL.

10.3 Adding a Character Set

This section discusses the procedure for adding a character set to MySQL. The proper procedure depends on whether the character set is simple or complex:

- If the character set does not need special string collating routines for sorting and does not need multibyte character support, it is simple.
- If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.

To use the following instructions, you must have a MySQL source distribution. In the instructions, *`MYSET`* represents the name of the character set that you want to add.

1. Add a `<charset>` element for *`MYSET`* to the `sql/share/charsets/Index.xml` file. Use the existing contents in the file as a guide to adding new contents. A partial listing for the `latin1` `<charset>` element follows:

```
<charset name="latin1">
  <family>Western</family>
  <description>cp1252 West European</description>
  ...
  <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
    <flag>primary</flag>
    <flag>compiled</flag>
  </collation>
  <collation name="latin1_danish_ci" id="15" order="Danish"/>
  ...
  <collation name="latin1_bin" id="47" order="Binary">
    <flag>binary</flag>
    <flag>compiled</flag>
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  </collation>
  ...
</charset>
```

The `<charset>` element must list all the collations for the character set. These must include at least a binary collation and a default (primary) collation. The default collation is often named using a suffix of `general_ci` (general, case insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a `primary` flag. The binary collation should have a `binary` flag.

You must assign a unique ID number to each collation, chosen from the range 1 to 254. To find the maximum of the currently used collation IDs, use this query:

```
SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multibyte functions, or both.

   For a simple character set, create a configuration file, *`MYSET`*`.xml`, that describes the character set properties. Create this file in the `sql/share/charsets` directory. You can use a copy of `latin1.xml` as the basis for this file. The syntax for the file is very simple:

   - Comments are written as ordinary XML comments (`<!-- `*`text`*` -->`).
   - Words within `<map>` array elements are separated by arbitrary amounts of whitespace.
   - Each word within `<map>` array elements must be a number in hexadecimal format.
   - The `<map>` array element for the `<ctype>` element has 257 words. The other `<map>` array elements after that have 256 words. See Section 10.3.1, “Character Definition Arrays”.
   - For each collation listed in the `<charset>` element for the character set in `Index.xml`, *`MYSET`*`.xml` must contain a `<collation>` element that defines the character ordering.

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   - Create the file `ctype-`*`MYSET`*`.c` in the `strings` directory. Look at one of the existing `ctype-*.c` files (such as `ctype-big5.c`) to see what needs to be defined. The arrays in your file must have names like `ctype_`*`MYSET`*, `to_lower_`*`MYSET`*, and so on. These correspond to the arrays for a simple character set. See Section 10.3.1, “Character Definition Arrays”.
   - For each `<collation>` element listed in the `<charset>` element for the character set in `Index.xml`, the `ctype-`*`MYSET`*`.c` file must provide an implementation of the collation.
   - If the character set requires string collating functions, see Section 10.3.2, “String Collating Support for Complex Character Sets”.
   - If the character set requires multibyte character support, see Section 10.3.3, “Multi-Byte Character Support for Complex Character Sets”.

3. Modify the configuration information. Use the existing configuration information as a guide to adding information for *`MYSYS`*. The example here assumes that the character set has default and binary collations, but more lines are needed if *`MYSET`* has additional collations.

   a. Edit `mysys/charset-def.c`, and “register” the collations for the new character set.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Add these lines to the “declaration” section:

```
#ifdef HAVE_CHARSET_MYSET
extern CHARSET_INFO my_charset_MYSET_general_ci;
extern CHARSET_INFO my_charset_MYSET_bin;
#endif
```

Add these lines to the “registration” section:

```
#ifdef HAVE_CHARSET_MYSET
  add_compiled_collation(&my_charset_MYSET_general_ci);
  add_compiled_collation(&my_charset_MYSET_bin);
#endif
```

b. If the character set uses `ctype-MYSET.c`, edit `strings/Makefile.am` and add `ctype-MYSET.c` to each definition of the `CSRCS` variable, and to the `EXTRA_DIST` variable.

c. If the character set uses `ctype-MYSET.c`, edit `libmysql/Makefile.shared` and add `ctype-MYSET.lo` to the `mystringsobjects` definition.

d. Edit `config/ac-macros/character_sets.m4`:

i. Add *`MYSET`* to one of the `define(CHARSETS_AVAILABLE...)` lines in alphabetic order.

ii. Add *`MYSET`* to `CHARSETS_COMPLEX`. This is needed even for simple character sets, or `configure` will not recognize `--with-charset=`*`MYSET`*.

iii. Add *`MYSET`* to the first `case` control structure. Omit the `USE_MB` and `USE_MB_IDENT` lines for 8-bit character sets.

```
MYSET)
  AC_DEFINE(HAVE_CHARSET_MYSET, 1, [Define to enable charset MYSET])
  AC_DEFINE([USE_MB], 1, [Use multi-byte character routines])
  AC_DEFINE(USE_MB_IDENT, 1)
  ;;
```

iv. Add *`MYSET`* to the second `case` control structure:

```
MYSET)
  default_charset_default_collation="MYSET_general_ci"
  default_charset_collations="MYSET_general_ci MYSET_bin"
  ;;
```

4. Reconfigure, recompile, and test.

10.3.1 Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. For a character set named *`MYSYS`*, the file is named *`MYSET`*`.xml`. It uses `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character.
- `<lower>` and `<upper>` list the lowercase and uppercase characters.
- `<unicode>` maps 8-bit character values to Unicode values.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `<collation>` elements indicate character ordering for comparisons and sorts, one element per collation. Binary collations need no `<map>` element because the character codes themselves provide the ordering.

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See also the existing `ctype-*.c` files for examples. See the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

Most of the arrays are indexed by character value and have 256 elements. The `<ctype>` array is indexed by character value + 1 and has 257 elements. This is a legacy convention for handling `EOF`.

`<ctype>` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m_ctype.h`:

```
#define _MY_U   01      /* Upper case */
#define _MY_L   02      /* Lower case */
#define _MY_NMR 04      /* Numeral (digit) */
#define _MY_SPC 010     /* Spacing character */
#define _MY_PNT 020     /* Punctuation */
#define _MY_CTR 040     /* Control character */
#define _MY_B   0100    /* Blank */
#define _MY_X   0200    /* heXadecimal digit */
```

The `<ctype>` value for a given character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_MY_U`) as well as a hexadecimal digit (`_MY_X`), so its `ctype` value should be defined like this:

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m_ctype.h` are octal values, but the elements of the `<ctype>` array in `MYSET.xml` should be written as hexadecimal values.

The `<lower>` and `<upper>` arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each `<collation>` array indicates how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the `<upper>` array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in Section 10.3.2, “String Collating Support for Complex Character Sets”.

10.3.2 String Collating Support for Complex Character Sets

For a simple character set named `MYSET`, sorting rules are specified in the `MYSET.xml` configuration file using `<map>` array elements within `<collation>` elements. If the sorting rules for your language are too complex to be handled with simple arrays, you must define string collating functions in the `ctype-MYSET.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets. Take a look at the `MY_COLLATION_HANDLER` structures

This documentation is for an older version. If you're
This documentation is for an older version. If you're

to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.3.3 Multi-Byte Character Support for Complex Character Sets

If you want to add support for a new character set named *`MYSET`* that includes multibyte characters, you must use multibyte character functions in the `ctype-`*`MYSET`*`.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets. Take a look at the `MY_CHARSET_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.4 Adding a Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

MySQL supports several collation implementations, as discussed in Section 10.4.1, “Collation Implementation Types”. Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets
- UCA-based collations for Unicode character sets
- Binary (*`xxx`*`_bin`) collations

The following sections describe how to add collations of the first two types to existing character sets. All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new collation:

1. Choose a collation ID
2. Add configuration information that names the collation and describes the character-ordering rules
3. Restart the server
4. Verify that the collation is present

The instructions here cover only collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source file), use the instructions in Section 10.3, “Adding a Character Set”. However, instead of adding all the information required for a complete character set, just modify the appropriate files for an existing character set. That is, based on what is already present for the character set's current collations, add data structures, functions, and configuration information for the new collation.

Note

If you modify an existing collation, that may affect the ordering of rows for indexes on columns that use the collation. In this case, rebuild any such indexes to

This documentation is for an older version. If you're
This documentation is for an older version. If you're

avoid problems such as incorrect query results. For further information, see Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.

Additional Resources

- The Unicode Collation Algorithm (UCA) specification: http://www.unicode.org/reports/tr10/
- The Locale Data Markup Language (LDML) specification: http://www.unicode.org/reports/tr35/

10.4.1 Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from character codes to weights. `latin1_swedish_ci` is an example. It is a case-insensitive collation, so the uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

For implementation instructions, see Section 10.4.3, “Adding a Simple Collation to an 8-Bit Character Set”.

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters, as described in Section 10.3, “Adding a Character Set”.

Collations for non-Unicode multibyte character sets

For this type of collation, 8-bit (single-byte) and multibyte characters are handled differently. For 8-bit characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte characters `'a'` and `'A'` both have a weight of `0x41`.) For multibyte characters, there are two types of relationship between character codes and weights:

- Weights equal character codes. `sjis_japanese_ci` is an example of this kind of collation. The multibyte character `'ぢ'` has a character code of `0x82C0`, and the weight is also `0x82C0`.
- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. `gbk_chinese_ci` is an example of this kind of collation. The multibyte character `'膰'` has a character code of `0x81B0` but a weight of `0xC286`.

For implementation instructions, see Section 10.3, “Adding a Character Set”.

Collations for Unicode multibyte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case insensitive and accent insensitive. `utf8_general_ci` is an example: `'a'`, `'A'`, `'À'`, and `'á'` each have different character codes but all have a weight of `0x0041` and compare as equal.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a' = 'A', 'a' = 'À', 'a' = 'á';
+-----------+-----------+-----------+
| 'a' = 'A' | 'a' = 'À' | 'a' = 'á' |
+-----------+-----------+-----------+
|         1 |         1 |         1 |
+-----------+-----------+-----------+
1 row in set (0.06 sec)
```

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits)
- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: `'a'` has a weight of `0x0E33`.
- A character may have many weights. This is an expansion. Example: The German letter `'ß'` (SZ ligature, or SHARP S) has a weight of `0x0FEA0FEA`.
- Many characters may have one weight. This is a contraction. Example: `'ch'` is a single letter in Czech and has a weight of `0x0EE2`.

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

For implementation instructions, for a non-UCA collation, see Section 10.3, “Adding a Character Set”. For a UCA collation, see Section 10.4.4, “Adding a UCA Collation to a Unicode Character Set”.

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

10.4.2 Choosing a Collation ID

Each collation must have a unique ID. To add a collation, you must choose an ID value that is not currently used. The value must be in the range from 1 to 254. The collation ID that you choose will appear in these contexts:

- The `ID` column of the `INFORMATION_SCHEMA.COLLATIONS` table
- The `Id` column of `SHOW COLLATION` output
- The `charsetnr` member of the `MYSQL_FIELD` C API data structure
- The `number` member of the `MY_CHARSET_INFO` data structure returned by the `mysql_get_character_set_info()` C API function

To determine the largest currently used ID, issue the following statement:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+---------+
| MAX(ID) |
+---------+
|     210 |
+---------+
```

For the output just shown, you could choose an ID higher than 210 for the new collation.

To display a list of all currently used IDs, issue this statement:

```
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+-----+
| ID  |
+-----+
|   1 |
|   2 |
| ... |
|  52 |
|  53 |
|  57 |
|  58 |
| ... |
|  98 |
|  99 |
| 128 |
| 129 |
| ... |
| 210 |
+-----+
```

In this case, you can either choose an unused ID from within the current range of IDs, or choose an ID that is higher than the current maximum ID. For example, in the output just shown, there are unused IDs between 53 and 57, and between 99 and 128. Or you could choose an ID higher than 210.

Warning

If you upgrade MySQL, you may find that the collation ID you choose has been assigned to a collation included in the new MySQL distribution. In this case, you will need to choose a new value for your own collation.

In addition, before upgrading, you should save the configuration files that you change. If you upgrade in place, the process will replace the your modified files.

10.4.3 Adding a Simple Collation to an 8-Bit Character Set

This section describes how to add a simple collation for an 8-bit character set by writing the `<collation>` elements associated with a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. The example adds a collation named `latin1_test_ci` to the `latin1` character set.

1. Choose a collation ID, as shown in Section 10.4.2, “Choosing a Collation ID”. The following steps use an ID of 56.

2. Modify the `Index.xml` and `latin1.xml` configuration files. These files will be located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

   ```
   mysql> SHOW VARIABLES LIKE 'character_sets_dir';
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+--------------------+----------------------------------------+
| Variable_name      | Value                                  |
+--------------------+----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+----------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. For example:

```
<charset name="latin1">
  ...
  <collation name="latin1_test_ci" id="56"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table for character codes 0 to 255. Each value within the `<map>` element must be a number in hexadecimal format.

```
<collation name="latin1_test_ci">
<map>
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
 60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>
```

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION LIKE 'latin1_test_ci';
+----------------+---------+----+---------+----------+---------+
| Collation      | Charset | Id | Default | Compiled | Sortlen |
+----------------+---------+----+---------+----------+---------+
| latin1_test_ci | latin1  | 56 |         |          |       1 |
+----------------+---------+----+---------+----------+---------+
```

10.4.4 Adding a UCA Collation to a Unicode Character Set

This section describes how to add a UCA collation for a Unicode character set by writing the `<collation>` element within a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. It uses a subset of the Locale Data Markup Language (LDML) specification, which is available at http://www.unicode.org/reports/tr35/. In 5.0, this method of adding collations is supported as of MySQL 5.0.46. With this method, you need not define the entire collation. Instead, you begin with an existing “base” collation and describe the new collation in terms of how it differs from the base collation. The following table lists the base collations of the Unicode character sets for which UCA collations can be defined.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 10.1 MySQL Character Sets Available for User-Defined UCA Collations

Character Set	Base Collation
utf8	utf8_unicode_ci
ucs2	ucs2_unicode_ci

The following sections show how to add a collation that is defined using LDML syntax, and provide a summary of LDML rules supported in MySQL.

10.4.4.1 Defining a UCA Collation Using LDML Syntax

To add a UCA collation for a Unicode character set without recompiling MySQL, use the following procedure. If you are unfamiliar with the LDML rules used to describe the collation's sort characteristics, see Section 10.4.4.2, “LDML Syntax Supported in MySQL”.

The example adds a collation named utf8_phone_ci to the utf8 character set. The collation is designed for a scenario involving a Web application for which users post their names and phone numbers. Phone numbers can be given in very different formats:

```
+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567
```

The problem raised by dealing with these kinds of values is that the varying permissible formats make searching for a specific phone number very difficult. The solution is to define a new collation that reorders punctuation characters, making them ignorable.

1. Choose a collation ID, as shown in Section 10.4.2, “Choosing a Collation ID”. The following steps use an ID of 252.

2. To modify the Index.xml configuration file. This file will be located in the directory named by the character_sets_dir system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+--------------------+----------------------------------------+
| Variable_name      | Value                                  |
+--------------------+----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+----------------------------------------+
```

3. Choose a name for the collation and list it in the Index.xml file. In addition, you'll need to provide the collation ordering rules. Find the <charset> element for the character set to which the collation is being added, and add a <collation> element that indicates the collation name and ID, to associate the name with the ID. Within the <collation> element, provide a <rules> element containing the ordering rules:

```
<charset name="utf8">
  ...
  <collation name="utf8_phone_ci" id="252">
    <rules>
      <reset>\u0000</reset>
      <s>\u0020</s> <!-- space -->
      <s>\u0028</s> <!-- left parenthesis -->
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
      <s>\u0029</s> <!-- right parenthesis -->
      <s>\u002B</s> <!-- plus -->
      <s>\u002D</s> <!-- hyphen -->
    </rules>
  </collation>
  ...
</charset>
```

4. If you want a similar collation for other Unicode character sets, add other `<collation>` elements. For example, to define `ucs2_phone_ci`, add a `<collation>` element to the `<charset name="ucs2">` element. Remember that each collation must have its own unique ID.

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION LIKE 'utf8_phone_ci';
+---------------+---------+-----+---------+----------+---------+
| Collation     | Charset | Id  | Default | Compiled | Sortlen |
+---------------+---------+-----+---------+----------+---------+
| utf8_phone_ci | utf8    | 252 |         |          |       8 |
+---------------+---------+-----+---------+----------+---------+
```

Now test the collation to make sure that it has the desired properties.

Create a table containing some sample phone numbers using the new collation:

```
mysql> CREATE TABLE phonebook (
    ->   name VARCHAR(64),
    ->   phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005');
Query OK, 1 row affected (0.00 sec)
```

Run some queries to see whether the ignored punctuation characters are in fact ignored for sorting and comparisons:

```
mysql> SELECT * FROM phonebook ORDER BY phone;
+-------+--------------------+
| name  | phone              |
+-------+--------------------+
| Sanja | +380 (912) 8008005 |
| Bar   | +7-912-800-80-01   |
| Svoj  | +7 912 800 80 02   |
| Ramil | (7912) 800 80 03   |
| Hf    | +7 (912) 800 80 04 |
+-------+--------------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='7 9 1 2 8 0 0 8 0 0 1';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)
```

10.4.4.2 LDML Syntax Supported in MySQL

This section describes the LDML syntax that MySQL recognizes. This is a subset of the syntax described in the LDML specification available at http://www.unicode.org/reports/tr35/, which should be consulted for further information. The rules described here are all supported except that character sorting occurs only at the primary level. Rules that specify differences at secondary or higher sort levels are recognized (and thus can be included in collation definitions) but are treated as equality at the primary level.

Character Representation

Characters named in LDML rules can be written in `\u`*nnnn* format, where *nnnn* is the hexadecimal Unicode code point value. Within hexadecimal values, the digits `A` through `F` are not case sensitive; `\u00E1` and `\u00e1` are equivalent. Basic Latin letters `A-Z` and `a-z` can also be written literally (this is a MySQL limitation; the LDML specification permits literal non-Latin1 characters in the rules). Only characters in the Basic Multilingual Plane can be specified. This notation does not apply to characters outside the BMP range of `0000` to `FFFF`.

The `Index.xml` file itself should be written using ASCII encoding.

Syntax Rules

LDML has reset rules and shift rules to specify character ordering. Orderings are given as a set of rules that begin with a reset rule that establishes an anchor point, followed by shift rules that indicate how characters sort relative to the anchor point.

- A `<reset>` rule does not specify any ordering in and of itself. Instead, it “resets” the ordering for subsequent shift rules to cause them to be taken in relation to a given character. Either of the following rules resets subsequent shift rules to be taken in relation to the letter `'A'`:

  ```
  <reset>A</reset>

  <reset>\u0041</reset>
  ```

- The `<p>`, `<s>`, and `<t>` shift rules define primary, secondary, and tertiary differences of a character from another character:

  - Use primary differences to distinguish separate letters.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Use secondary differences to distinguish accent variations.
- Use tertiary differences to distinguish lettercase variations.

Either of these rules specifies a primary shift rule for the `'G'` character:

```
<p>G</p>

<p>\u0047</p>
```

10.5 Character Set Configuration

You can change the default server character set and collation with the `--character-set-server` and `--collation-server` options when you start the server. The collation must be a legal collation for the default character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.) See Section 5.1.3, “Server Command Options”.

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program uses an incorrect path to determine where the character sets are stored (which is typically the `share/mysql/charsets` or `share/charsets` directory under the MySQL installation directory). This can be fixed by using the `--character-sets-dir` option when you run the program in question. For example, to specify a directory to be used by MySQL client programs, list it in the `[client]` group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

  ```
  [client]
  character-sets-dir=/usr/local/mysql/share/mysql/charsets

  [client]
  character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.0/share/charsets"
  ```

- The character set is a complex character set that cannot be loaded dynamically. In this case, you must recompile the program with support for the character set.

  For Unicode character sets, you can define collations without recompiling by using LDML notation. See Section 10.4.4, “Adding a UCA Collation to a Unicode Character Set”.

- The character set is a dynamic character set, but you do not have a configuration file for it. In this case, you should install the configuration file for the character set from a new MySQL distribution.

- If your character set index file does not contain the name for the character set, your program displays an error message. The file is named `Index.xml` and the message is:

  ```
  Character set 'charset_name' is not a compiled character set and is not
  specified in the '/usr/share/mysql/charsets/Index.xml' file
  ```

  To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This is normally unnecessary. However, when `character_set_system` differs from `character_set_server` or `character_set_client`, and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=system_character_set`—that is, setting the client character set to match the system character set—should fix the problem.

For `MyISAM` tables, you can check the character set name and number for a table with `myisamchk -dvv tbl_name`.

10.6 MySQL Server Time Zone Support

The MySQL server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable. The value does not change thereafter.

  You can set the system time zone for MySQL Server at startup with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`. The permissible values for `--timezone` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

- The server's current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial value for `time_zone` is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

  The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone=timezone` option on the command line, or you can use the following line in an option file:

  ```
  default-time-zone='timezone'
  ```

  If you have the `SUPER` privilege, you can set the global server time zone value at runtime with this statement:

  ```
  mysql> SET GLOBAL time_zone = timezone;
  ```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

  ```
  mysql> SET time_zone = timezone;
  ```

The current session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the current time zone to UTC for storage, and from UTC to the current time zone for retrieval.

The current time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current values of the global and client-specific time zones can be retrieved like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

timezone values can be given in several formats, none of which are case sensitive:

- The value `'SYSTEM'` indicates that the time zone should be the same as the system time zone.
- The value can be given as a string indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`.
- The value can be given as a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated.

Populating the Time Zone Tables

Several tables in the `mysql` system database exist to maintain time zone information (see Section 5.3, "The mysql System Database"). The MySQL installation procedure creates the time zone tables, but does not load them. You must do so manually using the following instructions.

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See the notes at the end of this section.

If your system has its own *zoneinfo* database (the set of files describing time zones), you should use the `mysql_tzinfo_to_sql` program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Solaris, and OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a zoneinfo database, you can use the downloadable package described later in this section.

The `mysql_tzinfo_to_sql` program is used to load the time zone tables. On the command line, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or to generate leap second information:

- To load a single time zone file *tz_file* that corresponds to a time zone name *tz_name*, invoke `mysql_tzinfo_to_sql` like this:

  ```
  shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone needs to account for leap seconds, initialize the leap second information like this, where *tz_file* is the name of your time zone file:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

- After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

If your system is one that has no zoneinfo database (for example, Windows or HP-UX), you can use a package that is available for download at the MySQL Developer Zone:

```
http://dev.mysql.com/downloads/timezones.html
```

You can use either a package that contains SQL statements to populate your existing time zone tables, or a package that contains pre-built `MyISAM` time zone tables to replace your existing tables:

- To use a time zone package that contains SQL statements, download and unpack it, then load the package file contents into your existing time zone tables:

  ```
  shell> mysql -u root mysql < file_name
  ```

  Then restart the server.

- To use a time zone package that contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables, download and unpack it. These table files are part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. Stop the server before doing this and restart it afterward.

Warning

Do not use a downloadable package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup, please see Section 16.4.1, “Replication Features and Issues”.

10.6.1 Staying Current with Time Zone Changes

When time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are two factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the Web site for your operating system vendor for an update that addresses the time changes.

- If you replace the system's `/etc/localtime` timezone file with a version that uses rules differing from those in effect at `mysqld` startup, you should restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.

- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date. If your system has its own zoneinfo database, you should reload the MySQL time zone tables whenever the zoneinfo database is updated. For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace the content of your current time zone tables. For instructions for both methods,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

see Populating the Time Zone Tables. `mysqld` caches time zone information that it looks up, so after updating the time zone tables, you should restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

A count of zero indicates that the table is empty. In this case, no one can be using named time zones, and you don't need to update the tables. A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, you should be sure to reload your time zone tables so that anyone who uses named time zones will get correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses these two queries:

```
SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you would see an incorrect result like this:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 01:00:00                                       |
+-----------------------------------------------------------+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 02:00:00                                       |
+-----------------------------------------------------------+
```

After updating the tables, you should see the correct result:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 01:00:00                                       |
+-----------------------------------------------------------+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 01:00:00                                       |
+-----------------------------------------------------------+
```

10.6.2 Time Zone Leap Second Support

Before MySQL 5.0.74, if the operating system is configured to return leap seconds from OS time calls or if the MySQL server uses a time zone definition that has leap seconds, functions such as `NOW()` could return a value having a time part that ends with `:59:60` or `:59:61`. If such values are inserted into a table, they would be dumped as is by `mysqldump` but considered invalid when reloaded, leading to backup/restore problems.

As of MySQL 5.0.74, leap second values are returned with a time part that ends with `:59:59`. This means that a function such as `NOW()` can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with `:59:60` or `:59:61` are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with `'YYYY-MM-DD hh:mm:ss'` values. The following example demonstrates this. It changes the local time zone to UTC so there is no difference between internal values (which are in UTC) and displayed values (which have time zone correction applied).

```
mysql> CREATE TABLE t1 (
    ->   a INT,
    ->   ts TIMESTAMP DEFAULT NOW(),
    ->   PRIMARY KEY (ts)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+------+---------------------+--------------------+
| a    | ts                  | UNIX_TIMESTAMP(ts) |
+------+---------------------+--------------------+
|    1 | 2008-12-31 23:59:59 |         1230767999 |
|    2 | 2008-12-31 23:59:59 |         1230768000 |
+------+---------------------+--------------------+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+------+---------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| a    | ts                  |
+------+---------------------+
|    1 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in column, which has the leap second correction applied:

```
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    2 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)
```

10.7 MySQL Server Locale Support

Beginning with MySQL 5.0.25, the locale indicated by the `lc_time_names` system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()`, and `MONTHNAME()` functions.

`lc_time_names` does not affect the `STR_TO_DATE()` or `GET_FORMAT()` function.

Locale names have language and region subtags listed by IANA (http://www.iana.org/assignments/language-subtag-registry) such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but you can set the value at server startup or set the `GLOBAL` value if you have the `SUPER` privilege. Any client can examine the value of `lc_time_names` or set its `SESSION` value to affect the locale for its own connection.

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
| Friday                | January                 |
+-----------------------+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----------------------------------------+
| Friday Fri January Jan                  |
+-----------------------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| es_MX           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
| viernes               | enero                   |
+-----------------------+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----------------------------------------+
| viernes vie enero ene                   |
+-----------------------------------------+
1 row in set (0.00 sec)
```

The day or month name for each of the affected functions is converted from `utf8` to the character set indicated by the `character_set_connection` system variable.

`lc_time_names` may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

`ar_AE`: Arabic - United Arab Emirates	`ar_BH`: Arabic - Bahrain
`ar_DZ`: Arabic - Algeria	`ar_EG`: Arabic - Egypt
`ar_IN`: Arabic - India	`ar_IQ`: Arabic - Iraq
`ar_JO`: Arabic - Jordan	`ar_KW`: Arabic - Kuwait
`ar_LB`: Arabic - Lebanon	`ar_LY`: Arabic - Libya
`ar_MA`: Arabic - Morocco	`ar_OM`: Arabic - Oman
`ar_QA`: Arabic - Qatar	`ar_SA`: Arabic - Saudi Arabia
`ar_SD`: Arabic - Sudan	`ar_SY`: Arabic - Syria
`ar_TN`: Arabic - Tunisia	`ar_YE`: Arabic - Yemen
`be_BY`: Belarusian - Belarus	`bg_BG`: Bulgarian - Bulgaria
`ca_ES`: Catalan - Spain	`cs_CZ`: Czech - Czech Republic
`da_DK`: Danish - Denmark	`de_AT`: German - Austria
`de_BE`: German - Belgium	`de_CH`: German - Switzerland
`de_DE`: German - Germany	`de_LU`: German - Luxembourg
`en_AU`: English - Australia	`en_CA`: English - Canada
`en_GB`: English - United Kingdom	`en_IN`: English - India
`en_NZ`: English - New Zealand	`en_PH`: English - Philippines
`en_US`: English - United States	`en_ZA`: English - South Africa

This documentation is for an older version. If you're
This documentation is for an older version. If you're

en_ZW: English - Zimbabwe	es_AR: Spanish - Argentina
es_BO: Spanish - Bolivia	es_CL: Spanish - Chile
es_CO: Spanish - Columbia	es_CR: Spanish - Costa Rica
es_DO: Spanish - Dominican Republic	es_EC: Spanish - Ecuador
es_ES: Spanish - Spain	es_GT: Spanish - Guatemala
es_HN: Spanish - Honduras	es_MX: Spanish - Mexico
es_NI: Spanish - Nicaragua	es_PA: Spanish - Panama
es_PE: Spanish - Peru	es_PR: Spanish - Puerto Rico
es_PY: Spanish - Paraguay	es_SV: Spanish - El Salvador
es_US: Spanish - United States	es_UY: Spanish - Uruguay
es_VE: Spanish - Venezuela	et_EE: Estonian - Estonia
eu_ES: Basque - Basque	fi_FI: Finnish - Finland
fo_FO: Faroese - Faroe Islands	fr_BE: French - Belgium
fr_CA: French - Canada	fr_CH: French - Switzerland
fr_FR: French - France	fr_LU: French - Luxembourg
gl_ES: Galician - Spain	gu_IN: Gujarati - India
he_IL: Hebrew - Israel	hi_IN: Hindi - India
hr_HR: Croatian - Croatia	hu_HU: Hungarian - Hungary
id_ID: Indonesian - Indonesia	is_IS: Icelandic - Iceland
it_CH: Italian - Switzerland	it_IT: Italian - Italy
ja_JP: Japanese - Japan	ko_KR: Korean - Republic of Korea
lt_LT: Lithuanian - Lithuania	lv_LV: Latvian - Latvia
mk_MK: Macedonian - FYROM	mn_MN: Mongolia - Mongolian
ms_MY: Malay - Malaysia	nb_NO: Norwegian(Bokmål) - Norway
nl_BE: Dutch - Belgium	nl_NL: Dutch - The Netherlands
no_NO: Norwegian - Norway	pl_PL: Polish - Poland
pt_BR: Portugese - Brazil	pt_PT: Portugese - Portugal
ro_RO: Romanian - Romania	ru_RU: Russian - Russia
ru_UA: Russian - Ukraine	sk_SK: Slovak - Slovakia
sl_SI: Slovenian - Slovenia	sq_AL: Albanian - Albania
sr_YU: Serbian - Yugoslavia	sv_FI: Swedish - Finland
sv_SE: Swedish - Sweden	ta_IN: Tamil - India
te_IN: Telugu - India	th_TH: Thai - Thailand
tr_TR: Turkish - Turkey	uk_UA: Ukrainian - Ukraine
ur_PK: Urdu - Pakistan	vi_VN: Vietnamese - Viet Nam
zh_CN: Chinese - China	zh_HK: Chinese - Hong Kong
zh_TW: Chinese - Taiwan Province of China	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 11 Data Types

Table of Contents

MySQL supports a number of data types in several categories: numeric types, date and time types, string (character and byte) types, and spatial types. This chapter provides an overview of these data types, a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the permissible formats in which you can specify values.

Data type descriptions use these conventions:

- *M* indicates the maximum display width for integer types. For floating-point and fixed-point types, *M* is the total number of digits that can be stored (the precision). For string types, *M* is the maximum length. The maximum permissible value of *M* depends on the data type.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- *D* applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than *M*−2.
- Square brackets (“[” and “]”) indicate optional parts of type definitions.

11.1 Data Type Overview

11.1.1 Numeric Type Overview

A summary of the numeric data types follows. For additional information about properties and storage requirements of the numeric types, see Section 11.2, “Numeric Types”, and Section 11.7, “Data Type Storage Requirements”.

M indicates the maximum display width for integer types. The maximum legal display width is 255. Display width is unrelated to the range of values a type can contain, as described in Section 11.2, “Numeric Types”. For floating-point and fixed-point types, *M* is the total number of digits that can be stored.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Numeric data types that permit the `UNSIGNED` attribute also permit `SIGNED`. However, these data types are signed by default, so the `SIGNED` attribute has no effect.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

Warning

When you use subtraction between integer values where one is of type `UNSIGNED`, the result is unsigned unless the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled. See Section 12.10, “Cast Functions and Operators”.

- `BIT[(M)]`

  A bit-field type. *M* indicates the number of bits per value, from 1 to 64. The default is 1 if *M* is omitted.

  This data type was added in MySQL 5.0.3 for `MyISAM`, and extended in 5.0.5 to `MEMORY`, `InnoDB`, `BDB`, and `NDBCLUSTER`. Before 5.0.3, `BIT` is a synonym for `TINYINT(1)`.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

- `BOOL`, `BOOLEAN`

  These types are synonyms for `TINYINT(1)`. A value of zero is considered false. Nonzero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----------------------+
| IF(0, 'true', 'false') |
+-----------------------+
| false                 |
+-----------------------+

mysql> SELECT IF(1, 'true', 'false');
+-----------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| IF(1, 'true', 'false') |
+------------------------+
| true                   |
+------------------------+

mysql> SELECT IF(2, 'true', 'false');
+------------------------+
| IF(2, 'true', 'false') |
+------------------------+
| true                   |
+------------------------+
```

However, the values `TRUE` and `FALSE` are merely aliases for `1` and `0`, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+--------------------------------+
| IF(0 = FALSE, 'true', 'false') |
+--------------------------------+
| true                           |
+--------------------------------+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-------------------------------+
| IF(1 = TRUE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-------------------------------+
| IF(2 = TRUE, 'true', 'false') |
+-------------------------------+
| false                         |
+-------------------------------+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+--------------------------------+
| IF(2 = FALSE, 'true', 'false') |
+--------------------------------+
| false                          |
+--------------------------------+
```

The last two statements display the results shown because `2` is equal to neither `1` nor `0`.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

  A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

  A medium-sized integer. The signed range is `-8388608` to `8388607`. The unsigned range is `0` to `16777215`.

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

  A normal-size integer. The signed range is `-2147483648` to `2147483647`. The unsigned range is `0` to `4294967295`.

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

  This type is a synonym for `INT`.

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

Some things you should be aware of with respect to `BIGINT` columns:

- All arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you should not use unsigned big integers larger than `9223372036854775807` (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

  MySQL can handle `BIGINT` in the following cases:

  - When using integers to store large unsigned values in a `BIGINT` column.
  - In `MIN(col_name)` or `MAX(col_name)`, where `col_name` refers to a `BIGINT` column.
  - When using operators (`+`, `-`, `*`, and so on) where both operands are integers.

- You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
- The `-`, `+`, and `*` operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than `9223372036854775807`.

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

  For MySQL 5.0.3 and above:

  A packed “exact” fixed-point number. *`M`* is the total number of digits (the precision) and *`D`* is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the “`-`” sign are not counted in *`M`*. If *`D`* is 0, values have no decimal point or fractional part. The maximum number of digits (*`M`*) for `DECIMAL` is 65 (64 from 5.0.3 to 5.0.5). The maximum number of supported decimals (*`D`*) is 30. If *`D`* is omitted, the default is 0. If *`M`* is omitted, the default is 10.

  `UNSIGNED`, if specified, disallows negative values.

  All basic calculations (`+, -, *, /`) with `DECIMAL` columns are done with a precision of 65 digits.

  Before MySQL 5.0.3:

  An unpacked fixed-point number. Behaves like a `CHAR` column; “unpacked” means the number is stored as a string, using one character for each digit of the value. *`M`* is the total number of digits and *`D`* is the number of digits after the decimal point. The decimal point and (for negative numbers) the “`-`” sign are not counted in *`M`*, although space for them is reserved. If *`D`* is 0, values have no decimal point or fractional part. The maximum range of `DECIMAL` values is the same as for `DOUBLE`, but the actual range for a given `DECIMAL` column may be constrained by the choice of *`M`* and *`D`*. If *`D`* is omitted, the default is 0. If *`M`* is omitted, the default is 10.

  `UNSIGNED`, if specified, disallows negative values.

  The behavior used by the server for `DECIMAL` columns in a table depends on the version of MySQL used to create the table. If your server is from MySQL 5.0.3 or higher, but you have `DECIMAL` columns

This documentation is for an older version. If you're
This documentation is for an older version. If you're

in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer `DECIMAL` format, dump them with `mysqldump` and reload them.

- `DEC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DECIMAL`. The `FIXED` synonym is available for compatibility with other database systems.

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

  A small (single-precision) floating-point number. Permissible values are `-3.402823466E+38` to `-1.175494351E-38`, `0`, and `1.175494351E-38` to `3.402823466E+38`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

  `UNSIGNED`, if specified, disallows negative values.

  Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section B.5.4.7, “Solving Problems with No Matching Rows”.

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

  A normal-size (double-precision) floating-point number. Permissible values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

  `UNSIGNED`, if specified, disallows negative values.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DOUBLE`. Exception: If the `REAL_AS_FLOAT` SQL mode is enabled, `REAL` is a synonym for `FLOAT` rather than `DOUBLE`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

  A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If *p* is from 0 to 24, the data type becomes `FLOAT` with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes `DOUBLE` with no *M* or *D* values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data types described earlier in this section.

  `FLOAT(p)` syntax is provided for ODBC compatibility.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

11.1.2 Date and Time Type Overview

A summary of the temporal data types follows. For additional information about properties and storage requirements of the temporal types, see Section 11.3, “Date and Time Types”, and Section 11.7, “Data Type Storage Requirements”. For descriptions of functions that operate on temporal values, see Section 12.7, “Date and Time Functions”.

For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

- `DATE`

  A date. The supported range is `'1000-01-01'` to `'9999-12-31'`. MySQL displays `DATE` values in `'YYYY-MM-DD'` format, but permits assignment of values to `DATE` columns using either strings or numbers.

- `DATETIME`

  A date and time combination. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`. MySQL displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS'` format, but permits assignment of values to `DATETIME` columns using either strings or numbers.

- `TIMESTAMP`

  A timestamp. The range is `'1970-01-01 00:00:01'` UTC to `'2038-01-19 03:14:07'` UTC. `TIMESTAMP` values are stored as the number of seconds since the epoch (`'1970-01-01 00:00:00'` UTC). A `TIMESTAMP` cannot represent the value `'1970-01-01 00:00:00'` because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing `'0000-00-00 00:00:00'`, the “zero” `TIMESTAMP` value.

  Unless specified otherwise, the first `TIMESTAMP` column in a table is defined to be automatically set to the date and time of the most recent modification if not explicitly assigned a value. This makes `TIMESTAMP` useful for recording the timestamp of an `INSERT` or `UPDATE` operation. You can also set any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values. The automatic initialization and updating to the current date and time can be specified using `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses, as described in Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”.

  Note

  The `TIMESTAMP` format that was used prior to MySQL 4.1 is not supported in MySQL 5.0; see *MySQL 3.23, 4.0, 4.1 Reference Manual* for information regarding the old format.

- `TIME`

  A time. The range is `'-838:59:59'` to `'838:59:59'`. MySQL displays `TIME` values in `'HH:MM:SS'` format, but permits assignment of values to `TIME` columns using either strings or numbers.

- `YEAR[(2|4)]`

  A year in two-digit or four-digit format. The default is four-digit format. `YEAR(2)` or `YEAR(4)` differ in display format, but have the same range of values. In four-digit format, values display as `1901` to `2155`, and `0000`. In two-digit format, values display as `70` to `69`, representing years from 1970 to 2069. MySQL displays `YEAR` values in `YYYY` or `YY` format, but permits assignment of values to `YEAR` columns using either strings or numbers.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For additional information about YEAR display format and interpretation of input values, see Section 11.3.3, “The YEAR Type”.

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Note

The MySQL server can be run with the MAXDB SQL mode enabled. In this case, TIMESTAMP is identical with DATETIME. If this mode is enabled at the time that a table is created, TIMESTAMP columns are created as DATETIME columns. As a result, such columns use DATETIME display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See Section 5.1.7, “Server SQL Modes”.

11.1.3 String Type Overview

A summary of the string data types follows. For additional information about properties and storage requirements of the string types, see Section 11.4, “String Types”, and Section 11.7, “Data Type Storage Requirements”.

In some cases, MySQL may change a string column to a type different from that given in a CREATE TABLE or ALTER TABLE statement. See Section 13.1.10.4, “Silent Column Specification Changes”.

In MySQL 4.1 and up, string data types include some features that you may not have encountered in working with versions of MySQL prior to 4.1:

- MySQL interprets length specifications in character column definitions in character units. (Before MySQL 4.1, column lengths were interpreted in bytes.) This applies to CHAR, VARCHAR, and the TEXT types.
- Column definitions for many string data types can include attributes that specify the character set or collation of the column. These attributes apply to the CHAR, VARCHAR, the TEXT types, ENUM, and SET data types:
  - The CHARACTER SET attribute specifies the character set, and the COLLATE attribute specifies a collation for the character set. For example:

    ```
    CREATE TABLE t
    (
        c1 VARCHAR(20) CHARACTER SET utf8,
        c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
    );
    ```

    This table definition creates a column named c1 that has a character set of utf8 with the default collation for that character set, and a column named c2 that has a character set of latin1 and a case-sensitive collation.

    The rules for assigning the character set and collation when either or both of the CHARACTER SET and COLLATE attributes are missing are described in Section 10.1.3.4, “Column Character Set and Collation”.

    CHARSET is a synonym for CHARACTER SET.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

  The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- The `ASCII` attribute is shorthand for `CHARACTER SET latin1`.
- The `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.
- The `BINARY` attribute is shorthand for specifying the binary collation of the column character set. In this case, sorting and comparison are based on numeric character values. (Before MySQL 4.1, `BINARY` caused a column to store binary strings and sorting and comparison were based on numeric byte values. This is the same as using character values for single-byte character sets, but not for multibyte character sets.)

- Character column sorting and comparison are based on the character set assigned to the column. (Before MySQL 4.1, sorting and comparison were based on the collation of the server character set.) For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary collation or the `BINARY` attribute to cause sorting and comparison to use the underlying character code values rather than a lexical ordering.

Section 10.1, “Character Set Support”, provides additional information about use of character sets in MySQL.

- `[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

  A fixed-length string that is always right-padded with spaces to the specified length when stored. *`M`* represents the column length in characters. The range of *`M`* is 0 to 255. If *`M`* is omitted, the length is 1.

  Note

  Trailing spaces are removed when `CHAR` values are retrieved.

  Before MySQL 5.0.3, a `CHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `CHAR(500)` is converted to `TEXT`, and `CHAR(200000)` is converted to `MEDIUMTEXT`. However, this conversion causes the column to become a variable-length column, and also affects trailing-space removal.

  In MySQL 5.0.3 and later, a `CHAR` length greater than 255 is illegal and fails with an error:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> CREATE TABLE c1 (col1 INT, col2 CHAR(500));
ERROR 1074 (42000): Column length too big for column 'col' (max = 255);
use BLOB or TEXT instead
```

`CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. Section 10.1.3.6, “National Character Set”.

The `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

MySQL permits you to create a column of type `CHAR(0)`. This is useful primarily when you have to be compliant with old applications that depend on the existence of a column but that do not actually use its value. `CHAR(0)` is also quite nice when you need a column that can take only two values: A column that is defined as `CHAR(0) NULL` occupies only one bit and can take only the values `NULL` and `''` (the empty string).

- `[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

  A variable-length string. *`M`* represents the maximum column length in characters. The range of *`M`* is 0 to 255 before MySQL 5.0.3, and 0 to 65,535 in MySQL 5.0.3 and later. The effective maximum length of a `VARCHAR` in MySQL 5.0.3 and later is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, `utf8` characters can require up to three bytes per character, so a `VARCHAR` column that uses the `utf8` character set can be declared to be a maximum of 21,844 characters. See Section C.7.4, “Limits on Table Column Count and Row Size”.

  MySQL stores `VARCHAR` values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A `VARCHAR` column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

  Note

  Before 5.0.3, trailing spaces were removed when `VARCHAR` values were stored, which differs from the standard SQL specification.

  Prior to MySQL 5.0.3, a `VARCHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `VARCHAR(500)` is converted to `TEXT`, and `VARCHAR(200000)` is converted to `MEDIUMTEXT`. However, this conversion affects trailing-space removal.

  `VARCHAR` is shorthand for `CHARACTER VARYING`. `NATIONAL VARCHAR` is the standard SQL way to define that a `VARCHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. Section 10.1.3.6, “National Character Set”. `NVARCHAR` is shorthand for `NATIONAL VARCHAR`.

- `BINARY(M)`

  The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than nonbinary character strings. *`M`* represents the column length in bytes.

- `VARBINARY(M)`

  The `VARBINARY` type is similar to the `VARCHAR` type, but stores binary byte strings rather than nonbinary character strings. *`M`* represents the maximum column length in bytes.

- `TINYBLOB`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A `BLOB` column with a maximum length of 255 (2^8 − 1) bytes. Each `TINYBLOB` value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- `TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 255 (2^8 − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each `TINYTEXT` value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- `BLOB[(M)]`

  A `BLOB` column with a maximum length of 65,535 (2^{16} − 1) bytes. Each `BLOB` value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *`M`* can be given for this type. If this is done, MySQL creates the column as the smallest `BLOB` type large enough to hold values *`M`* bytes long.

- `TEXT[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 65,535 (2^{16} − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each `TEXT` value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *`M`* can be given for this type. If this is done, MySQL creates the column as the smallest `TEXT` type large enough to hold values *`M`* characters long.

- `MEDIUMBLOB`

  A `BLOB` column with a maximum length of 16,777,215 (2^{24} − 1) bytes. Each `MEDIUMBLOB` value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- `MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 16,777,215 (2^{24} − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each `MEDIUMTEXT` value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- `LONGBLOB`

  A `BLOB` column with a maximum length of 4,294,967,295 or 4GB (2^{32} − 1) bytes. The effective maximum length of `LONGBLOB` columns depends on the configured maximum packet size in the client/server protocol and available memory. Each `LONGBLOB` value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- `LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 4,294,967,295 or 4GB (2^{32} − 1) characters. The effective maximum length is less if the value contains multibyte characters. The effective maximum length of `LONGTEXT` columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each `LONGTEXT` value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- `ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

  An enumeration. A string object that can have only one value, chosen from the list of values `'value1'`, `'value2'`, `...`, `NULL` or the special `''` error value. `ENUM` values are represented internally as integers.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

An ENUM column can have a maximum of 65,535 distinct elements. (The practical limit is less than 3000.) A table can have no more than 255 unique element list definitions among its ENUM and SET columns considered as a group. For more information on these limits, see Section C.7.5, “Limits Imposed by .frm File Structure”.

- SET('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A set. A string object that can have zero or more values, each of which must be chosen from the list of values '*value1*', '*value2*', ... SET values are represented internally as integers.

  A SET column can have a maximum of 64 distinct members. A table can have no more than 255 unique element list definitions among its ENUM and SET columns considered as a group. For more information on this limit, see Section C.7.5, “Limits Imposed by .frm File Structure”.

11.2 Numeric Types

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keywords DEC and FIXED are synonyms for DECIMAL. MySQL treats DOUBLE as a synonym for DOUBLE PRECISION (a nonstandard extension). MySQL also treats REAL as a synonym for DOUBLE PRECISION (a nonstandard variation), unless the REAL_AS_FLOAT SQL mode is enabled.

As of MySQL 5.0.3, a BIT data type is available for storing bit-field values. (Before 5.0.3, MySQL interprets BIT as TINYINT(1).) In MySQL 5.0.3, BIT is supported only for MyISAM. MySQL 5.0.5 extends BIT support to MEMORY, InnoDB, BDB, and NDBCLUSTER.

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see Section 11.2.6, “Out-of-Range and Overflow Handling”.

For information about numeric type storage requirements, see Section 11.7, “Data Type Storage Requirements”.

The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see Section 12.6.1, “Arithmetic Operators”.

11.2.1 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

MySQL supports the SQL standard integer types INTEGER (or INT) and SMALLINT. As an extension to the standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each integer type.

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

11.2.2 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC.

As of MySQL 5.0.3, DECIMAL values are stored in binary format. Previously, they were stored as strings, with one character used for each digit of the value, the decimal point (if the scale is greater than 0), and the "-" sign (for negative numbers). See Section 12.17, "Precision Math".

In a DECIMAL column declaration, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99. MySQL enforces this limit as of MySQL 5.0.3. Before 5.0.3, on the positive end of the range, the column could actually store numbers up to 9999.99. (For positive numbers, MySQL 5.0.2 and earlier used the byte reserved for the sign to extend the upper end of the range.)

In standard SQL, the syntax DECIMAL(*M*) is equivalent to DECIMAL(*M*,0). Similarly, the syntax DECIMAL is equivalent to DECIMAL(*M*,0), where the implementation is permitted to decide the value of *M*. MySQL supports both of these variant forms of DECIMAL syntax. The default value of *M* is 10.

If the scale is 0, DECIMAL values contain no decimal point or fractional part.

The maximum number of digits for DECIMAL is 65 (64 from MySQL 5.0.3 to 5.0.5). Before MySQL 5.0.3, the maximum range of DECIMAL values is the same as for DOUBLE, but the actual range for a given DECIMAL column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

11.2.3 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

The FLOAT and DOUBLE types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For FLOAT, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses. MySQL also supports this optional precision specification, but the precision value is used only to determine storage size. A precision from 0 to 23

This documentation is for an older version. If you're
This documentation is for an older version. If you're

results in a 4-byte single-precision `FLOAT` column. A precision from 24 to 53 results in an 8-byte double-precision `DOUBLE` column.

MySQL permits a nonstandard syntax: `FLOAT(M,D)` or `REAL(M,D)` or `DOUBLE PRECISION(M,D)`. Here, "`(M,D)`" means than values can be stored with up to `M` digits in total, of which `D` digits may be after the decimal point. For example, a column defined as `FLOAT(7,4)` will look like `-999.9999` when displayed. MySQL performs rounding when storing values, so if you insert `999.00009` into a `FLOAT(7,4)` column, the approximate result is `999.0001`.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see Section B.5.4.8, "Problems with Floating-Point Values"

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

11.2.4 Bit-Value Type - BIT

As of MySQL 5.0.3, the `BIT` data type is used to store bit-field values. A type of `BIT(M)` enables storage of `M`-bit values. `M` can range from 1 to 64.

To specify bit values, `b'value'` notation can be used. `value` is a binary value written using zeros and ones. For example, `b'111'` and `b'10000000'` represent 7 and 128, respectively. See Section 9.1.6, "Bit-Field Literals".

If you assign a value to a `BIT(M)` column that is less than `M` bits long, the value is padded on the left with zeros. For example, assigning a value of `b'101'` to a `BIT(6)` column is, in effect, the same as assigning `b'000101'`.

MySQL Cluster. The maximum combined size of all `BIT` columns used in a given `NDB` table must not exceed 4096 bits.

11.2.5 Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, `INT(4)` specifies an `INT` with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used or not is up to the application.)

The display width does *not* constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as `SMALLINT(3)` has the usual `SMALLINT` range of `-32768` to `32767`, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) attribute `ZEROFILL`, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(4) ZEROFILL`, a value of `5` is retrieved as `0005`.

Note

The `ZEROFILL` attribute is ignored when a column is involved in expressions or `UNION` queries.

If you store values larger than the display width in an integer column that has the `ZEROFILL` attribute, you may experience problems when MySQL generates

This documentation is for an older version. If you're
This documentation is for an older version. If you're

temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

All integer types can have an optional (nonstandard) attribute `UNSIGNED`. Unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift from `-2147483648` and `2147483647` up to `0` and `4294967295`.

Floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Integer or floating-point data types can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is *`value`*`+1`, where *`value`* is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`. (Inserting `NULL` to generate `AUTO_INCREMENT` values requires that the column be declared `NOT NULL`. If the column is declared `NULL`, inserting `NULL` stores a `NULL`.) When you insert any other value into an `AUTO_INCREMENT` column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the inserted value.

11.2.6 Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, the result depends on the SQL mode in effect at the time:

- If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.
- If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead.

  When an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range. If you store 256 into a `TINYINT` or `TINYINT UNSIGNED` column, MySQL stores 127 or 255, respectively.

  When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Column-assignment conversions that occur due to clipping when MySQL is not operating in strict mode are reported as warnings for `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, and multiple-row `INSERT` statements. In strict mode, these statements fail, and some or all the values will not be inserted or changed, depending on whether the table is a transactional table and other factors. For details, see Section 5.1.7, “Server SQL Modes”.

Overflow handling during numeric expression evaluation depends on the types of the operands:

- Integer overflow results in silent wraparound.
- `DECIMAL` overflow results in a truncated result and a warning.
- Floating-point overflow produces a `NULL` result. Overflow for some operations can result in `+INF`, `-INF`, or `NaN`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For example, the largest signed `BIGINT` value is 9223372036854775807, so the following expression wraps around to the minimum `BIGINT` value:

```
mysql> SELECT 9223372036854775807 + 1;
+-------------------------+
| 9223372036854775807 + 1 |
+-------------------------+
|    -9223372036854775808 |
+-------------------------+
```

To enable the operation to succeed in this case, convert the value to unsigned;

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-----------------------------------------+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-----------------------------------------+
|                      9223372036854775808 |
+-----------------------------------------+
```

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because `DECIMAL` values have a larger range than integers:

```
mysql> SELECT 9223372036854775807.0 + 1;
+---------------------------+
| 9223372036854775807.0 + 1 |
+---------------------------+
|     9223372036854775808.0 |
+---------------------------+
```

Subtraction between integer values, where one is of type `UNSIGNED`, produces an unsigned result by default. If the result would otherwise have been negative, it becomes the maximum integer value. If the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the result is negative.

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|    18446744073709551615 |
+-------------------------+

mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

If the result of such an operation is used to update an `UNSIGNED` integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if `NO_UNSIGNED_SUBTRACTION` is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

11.3 Date and Time Types

The date and time types for representing temporal values are `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`. Each temporal type has a range of legal values, as well as a “zero” value that may be used when you specify an illegal value that MySQL cannot represent. The `TIMESTAMP` type has special automatic

This documentation is for an older version. If you're
This documentation is for an older version. If you're

updating behavior, described later. For temporal type storage requirements, see Section 11.7, “Data Type Storage Requirements”.

Keep in mind these general considerations when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). For a description of the permitted formats for date and time types, see Section 9.1.3, “Date and Time Literals”. It is expected that you supply legal values. Unpredictable results may occur if you use values in other formats.
- Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, `'98-09-04'`), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, `'09-04-98'`, `'04-09-98'`).
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:
  - Year values in the range `70-99` are converted to `1970-1999`.
  - Year values in the range `00-69` are converted to `2000-2069`.

  See also Section 11.3.8, “Two-Digit Years in Dates”.
- Conversion of values from one temporal type to another occurs according to the rules in Section 11.3.7, “Conversion Between Date and Time Types”.
- MySQL automatically converts a date or time value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise illegal for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range `TIME` values are clipped to the appropriate endpoint of the `TIME` range.
- Starting from MySQL 5.0.2, by setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See Section 5.1.7, “Server SQL Modes”.) You can get MySQL to accept certain dates, such as `'2009-11-31'`, by enabling the `ALLOW_INVALID_DATES` SQL mode. (Before 5.0.2, this mode was the default behavior for MySQL.) This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 1 to 12 and that the day is in the range from 1 to 31.
- MySQL permits you to store dates where the day or month and day are zero in a `DATE` or `DATETIME` column. This is useful for applications that need to store birthdates for which you may not know the exact date. In this case, you simply store the date as `'2009-00-00'` or `'2009-01-00'`. If you store dates such as these, you should not expect to get correct results for functions such as `DATE_SUB()` or `DATE_ADD()` that require complete dates. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` SQL mode.
- MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” This is in some cases more convenient than using `NULL` values, and uses less data and index space. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` SQL mode.
- “Zero” date or time values used through Connector/ODBC are converted automatically to `NULL` because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

values '0' or 0, which are easier to write. For temporal types that include a date part (DATE, DATETIME, and TIMESTAMP), use of these values produces warnings if the NO_ZERO_DATE SQL mode is enabled.

Data Type	"Zero" Value
DATE	'0000-00-00'
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	'0000-00-00 00:00:00'
YEAR	0000

11.3.1 The DATE, DATETIME, and TIMESTAMP Types

The DATE, DATETIME, and TIMESTAMP types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes DATE, DATETIME, and TIMESTAMP values in several formats, described in Section 9.1.3, "Date and Time Literals". For the DATE and DATETIME range descriptions, "supported" means that although earlier values might work, there is no guarantee.

The DATE type is used for values with a date part but no time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

The DATETIME type is used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The TIMESTAMP data type is used for values that contain both date and time parts. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

MySQL converts TIMESTAMP values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as DATETIME.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis. As long as the time zone setting remains constant, you get back the same value you store. If you store a TIMESTAMP value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the time_zone system variable. For more information, see Section 10.6, "MySQL Server Time Zone Support".

The TIMESTAMP data type offers automatic initialization and updating to the current date and time. For more information, see Section 11.3.5, "Automatic Initialization and Updating for TIMESTAMP".

A DATETIME or TIMESTAMP value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into DATETIME or TIMESTAMP columns. For information about fractional seconds support in MySQL, see Section 11.3.6, "Fractional Seconds in Time Values".

Illegal DATE, DATETIME, or TIMESTAMP values are converted to the "zero" value of the appropriate type ('0000-00-00' or '0000-00-00 00:00:00').

Be aware of certain properties of date value interpretation in MySQL:

- MySQL permits a "relaxed" format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as '10:11:12' might look like a time value because of the ":" delimiter, but is interpreted as the year '2010-11-12' if used in a date context. The value '10:45:15' is converted to '0000-00-00' because '45' is not a legal month.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- As of 5.0.2, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`. See Section 5.1.7, “Server SQL Modes”, for more information.

  Before MySQL 5.0.2, the MySQL server performs only basic checking on the validity of a date: The ranges for year, month, and day are 1000 to 9999, 00 to 12, and 00 to 31, respectively. Any date containing parts not within these ranges is subject to conversion to `'0000-00-00'`. Please note that this still permits you to store invalid dates such as `'2002-04-31'`. To ensure that a date is valid, you should perform a check in your application.

- As of MySQL 5.0.2, MySQL does not accept `TIMESTAMP` values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special “zero” value `'0000-00-00 00:00:00'`.

- `CAST()` treats a `TIMESTAMP` value as a string when not selecting from a table. (This is true even if you specify `FROM DUAL`.) See Section 12.10, “Cast Functions and Operators”.

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

  - Year values in the range `00-69` are converted to `2000-2069`.

  - Year values in the range `70-99` are converted to `1970-1999`.

  See also Section 11.3.8, “Two-Digit Years in Dates”.

Note

The MySQL server can be run with the `MAXDB` SQL mode enabled. In this case, `TIMESTAMP` is identical with `DATETIME`. If this mode is enabled at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See Section 5.1.7, “Server SQL Modes”.

11.3.2 The TIME Type

MySQL retrieves and displays `TIME` values in `'HH:MM:SS'` format (or `'HHH:MM:SS'` format for large hours values). `TIME` values may range from `'-838:59:59'` to `'838:59:59'`. The hours part may be so large because the `TIME` type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes `TIME` values in several formats, described in Section 9.1.3, “Date and Time Literals”. Some of these formats can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `TIME` columns. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”.

Be careful about assigning abbreviated values to a `TIME` column. MySQL interprets abbreviated `TIME` values with colons as time of the day. That is, `'11:12'` means `'11:12:00'`, not `'00:11:12'`. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of `'1112'` and `1112` as meaning `'11:12:00'` (12 minutes after 11 o'clock), but MySQL interprets them as `'00:11:12'` (11 minutes, 12 seconds). Similarly, `'12'` and `12` are interpreted as `'00:00:12'`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

By default, values that lie outside the TIME range but are otherwise legal are clipped to the closest endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'. Illegal TIME values are converted to '00:00:00'. Note that because '00:00:00' is itself a legal TIME value, there is no way to tell, from a value of '00:00:00' stored in a table, whether the original value was specified as '00:00:00' or whether it was illegal.

For more restrictive treatment of invalid TIME values, enable strict SQL mode to cause errors to occur. See Section 5.1.7, "Server SQL Modes".

11.3.3 The YEAR Type

The YEAR type is a 1-byte type used to represent year values. It can be declared as YEAR(4) or YEAR(2) to specify a display width of four or two characters. The default is four characters if no width is given.

Note

The YEAR(2) data type has certain issues that you should consider before choosing to use it. For more information, see Section 11.3.4, "YEAR(2) Limitations and Migrating to YEAR(4)".

YEAR(4) and YEAR(2) differ in display format, but have the same range of values. For 4-digit format, MySQL displays YEAR values in YYYY format, with a range of 1901 to 2155, or 0000. For 2-digit format, MySQL displays only the last two (least significant) digits; for example, 70 (1970 or 2070) or 69 (2069).

You can specify input YEAR values in a variety of formats:

- As a 4-digit number in the range 1901 to 2155.
- As a 4-digit string in the range '1901' to '2155'.
- As a 1- or 2-digit number in the range 1 to 99. MySQL converts values in the ranges 1 to 69 and 70 to 99 to YEAR values in the ranges 2001 to 2069 and 1970 to 1999.
- As a 1- or 2-digit string in the range '0' to '99'. MySQL converts values in the ranges '0' to '69' and '70' to '99' to YEAR values in the ranges 2000 to 2069 and 1970 to 1999.
- Inserting a numeric 0 has a different effect for YEAR(2) and YEAR(4). For YEAR(2), the result has a display value of 00 and an internal value of 2000. For YEAR(4), the result has a display value of 0000 and an internal value of 0000. To specify zero for YEAR(4) and have it be interpreted as 2000, specify it as a string '0' or '00'.
- As the result of a function that returns a value that is acceptable in a YEAR context, such as NOW().

MySQL converts invalid YEAR values to 0000.

See also Section 11.3.8, "Two-Digit Years in Dates".

11.3.4 YEAR(2) Limitations and Migrating to YEAR(4)

This section describes problems that can occur when using YEAR(2) and provides information about converting existing YEAR(2) columns to YEAR(4).

Although the internal range of values for YEAR(4) and YEAR(2) is the same (1901 to 2155, and 0000), the display width for YEAR(2) makes that type inherently ambiguous because displayed values indicate only the last two digits of the internal values and omit the century digits. The result can be a loss of information under certain circumstances. For this reason, consider avoiding YEAR(2) throughout your applications and using YEAR(4) wherever you need a YEAR data type. Note that conversion will become

This documentation is for an older version. If you're
This documentation is for an older version. If you're

necessary at some point because support for YEAR data types with display values other than 4, most notably YEAR(2), is reduced as of MySQL 5.6.6 and will be removed entirely in a future release.

YEAR(2) Limitations

Issues with the YEAR(2) data type include ambiguity of displayed values, and possible loss of information when values are dumped and reloaded or converted to strings.

- Displayed YEAR(2) values can be ambiguous. It is possible for up to three YEAR(2) values that have different internal values to have the same displayed value, as the following example demonstrates:

```
mysql> CREATE TABLE t (y2 YEAR(2), y4 YEAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t (y2) VALUES(1912),(2012),(2112);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> UPDATE t SET y4 = y2;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM t;
+------+------+
| y2   | y4   |
+------+------+
|   12 | 1912 |
|   12 | 2012 |
|   12 | 2112 |
+------+------+
3 rows in set (0.00 sec)
```

- If you use mysqldump to dump the table created in the preceding item, the dump file represents all y2 values using the same 2-digit representation (12). If you reload the table from the dump file, all resulting rows have internal value 2012 and display value 12, thus losing the distinctions among them.
- Conversion of a YEAR(2) or YEAR(4) data value to string form uses the display width of the YEAR type. Suppose that YEAR(2) and YEAR(4) columns both contain the value 1970. Assigning each column to a string results in a value of '70' or '1970', respectively. That is, loss of information occurs for conversion from YEAR(2) to string.
- Values outside the range from 1970 to 2069 are stored incorrectly when inserted into a YEAR(2) column in a CSV table. For example, inserting 2111 results in a display value of 11 but an internal value of 2011.

To avoid these problems, use YEAR(4) rather than YEAR(2). Suggestions regarding migration strategies appear later in this section.

Migrating from YEAR(2) to YEAR(4)

To convert YEAR(2) columns to YEAR(4), use ALTER TABLE. Suppose that a table t1 has this definition:

```
CREATE TABLE t1 (ycol YEAR(2) NOT NULL DEFAULT '70');
```

Modify the column using ALTER TABLE as follows. Remember to include any column attributes such as NOT NULL or DEFAULT:

```
ALTER TABLE t1 MODIFY ycol YEAR(4) NOT NULL DEFAULT '1970';
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `ALTER TABLE` statement converts the table without changing `YEAR(2)` values. If the server is a replication master, the `ALTER TABLE` statement replicates to slaves and makes the corresponding table change on each one.

One migration method should be avoided: Do not dump your data with `mysqldump` and reload the dump file after upgrading. This has the potential to change `YEAR(2)` values, as described previously.

A migration from `YEAR(2)` to `YEAR(4)` should also involve examining application code for the possibility of changed behavior under conditions such as these:

- Code that expects selecting a `YEAR` column to produce exactly two digits.
- Code that does not account for different handling for inserts of numeric `0`: Inserting `0` into `YEAR(2)` or `YEAR(4)` results in an internal value of `2000` or `0000`, respectively.

11.3.5 Automatic Initialization and Updating for TIMESTAMP

Note

In older versions of MySQL (prior to 4.1), the properties of the `TIMESTAMP` data type differed significantly in several ways from what is described in this section (see the *MySQL 3.23, 4.0, 4.1 Reference Manual* for details); these include syntax extensions which are deprecated in MySQL 5.1, and no longer supported in MySQL 5.5. This has implications for performing a dump and restore or replicating between MySQL Server versions. If you are using columns that are defined using the old `TIMESTAMP(N)` syntax, see Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”, prior to upgrading to MySQL 5.1 or later.

The `TIMESTAMP` data type offers automatic initialization and updating to the current date and time (that is, the current timestamp). You can choose whether to use these properties and which column should have them:

- One `TIMESTAMP` column in a table can have the current timestamp as the default value for initializing the column, as the auto-update value, or both. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.
- If the column is auto-initialized, it is set to the current timestamp for inserted rows that specify no value for the column.
- If the column is auto-updated, it is automatically updated to the current timestamp when the value of any other column in the row is changed from its current value. The column remains unchanged if all other columns are set to their current values. To prevent the column from updating when other columns change, explicitly set it to its current value. To update the column even when other columns do not change, explicitly set it to the value it should have (for example, set it to `CURRENT_TIMESTAMP`).

In addition, you can initialize or update any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values.

To specify automatic properties, use the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses. The order of the clauses does not matter. If both are present in a column definition, either can occur first. Any of the synonyms for `CURRENT_TIMESTAMP` have the same meaning as `CURRENT_TIMESTAMP`. These are `CURRENT_TIMESTAMP()`, `NOW()`, `LOCALTIME`, `LOCALTIME()`, `LOCALTIMESTAMP`, and `LOCALTIMESTAMP()`.

Use of `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` is specific to `TIMESTAMP`. The `DEFAULT` clause also can be used to specify a constant (nonautomatic) default value; for example, `DEFAULT 0` or `DEFAULT '2000-01-01 00:00:00'`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

The following examples use `DEFAULT 0`, a default that can produce warnings or errors depending on whether strict SQL mode or the `NO_ZERO_DATE` SQL mode is enabled. Be aware that the `TRADITIONAL` SQL mode includes strict mode and `NO_ZERO_DATE`. See Section 5.1.7, “Server SQL Modes”.

The following rules describe the possibilities for defining the first `TIMESTAMP` column in a table with the current timestamp for both the default and auto-update values, for one but not the other, or for neither:

- With both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP`, the column has the current timestamp for its default value and is automatically updated to the current timestamp.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- With neither `DEFAULT CURRENT_TIMESTAMP` nor `ON UPDATE CURRENT_TIMESTAMP`, it is the same as specifying both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP`.

```
CREATE TABLE t1 (
  ts TIMESTAMP
);
```

- With a `DEFAULT` clause but no `ON UPDATE CURRENT_TIMESTAMP` clause, the column has the given default value and is not automatically updated to the current timestamp.

  The default depends on whether the `DEFAULT` clause specifies `CURRENT_TIMESTAMP` or a constant value. With `CURRENT_TIMESTAMP`, the default is the current timestamp.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

  With a constant, the default is the given value. In this case, the column has no automatic properties at all.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0
);
```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause and a constant `DEFAULT` clause, the column is automatically updated to the current timestamp and has the given constant default value.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
);
```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause but no `DEFAULT` clause, the column is automatically updated to the current timestamp. The default is 0 unless the column is defined with the `NULL` attribute, in which case the default is `NULL`.

```
CREATE TABLE t1 (
  ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP      -- default 0
);
CREATE TABLE t2 (
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  ts TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP -- default NULL
);
```

It need not be the first `TIMESTAMP` column in a table that is automatically initialized or updated to the current timestamp. However, to specify automatic initialization or updating for a different `TIMESTAMP` column, you must suppress the automatic properties for the first one. Then, for the other `TIMESTAMP` column, the rules for the `DEFAULT` and `ON UPDATE` clauses are the same as for the first `TIMESTAMP` column, except that if you omit both clauses, no automatic initialization or updating occurs.

To suppress automatic properties for the first `TIMESTAMP` column, do either of the following:

- Define the column with a `DEFAULT` clause that specifies a constant default value.
- Specify the `NULL` attribute. This also causes the column to permit `NULL` values, which means that you cannot assign the current timestamp by setting the column to `NULL`. Assigning `NULL` sets the column to `NULL`.

Consider these table definitions:

```
CREATE TABLE t1 (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t2 (
  ts1 TIMESTAMP NULL,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t3 (
  ts1 TIMESTAMP NULL DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
```

The tables have these properties:

- In each table definition, the first `TIMESTAMP` column has no automatic initialization or updating.
- The tables differ in how the `ts1` column handles `NULL` values. For `t1`, `ts1` is `NOT NULL` and assigning it a value of `NULL` sets it to the current timestamp. For `t2` and `t3`, `ts1` permits `NULL` and assigning it a value of `NULL` sets it to `NULL`.
- `t2` and `t3` differ in the default value for `ts1`. For `t2`, `ts1` is defined to permit `NULL`, so the default is also `NULL` in the absence of an explicit `DEFAULT` clause. For `t3`, `ts1` permits `NULL` but has an explicit default of 0.

TIMESTAMP Initialization and the NULL Attribute

By default, `TIMESTAMP` columns are `NOT NULL`, cannot contain `NULL` values, and assigning `NULL` assigns the current timestamp. To permit a `TIMESTAMP` column to contain `NULL`, explicitly declare it with the `NULL` attribute. In this case, the default value also becomes `NULL` unless overridden with a `DEFAULT` clause that specifies a different default value. `DEFAULT NULL` can be used to explicitly specify `NULL` as the default value. (For a `TIMESTAMP` column not declared with the `NULL` attribute, `DEFAULT NULL` is illegal.) If a `TIMESTAMP` column permits `NULL` values, assigning `NULL` sets it to `NULL`, not to the current timestamp.

The following table contains several `TIMESTAMP` columns that permit `NULL` values:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

A `TIMESTAMP` column that permits `NULL` values does *not* take on the current timestamp at insert time except under one of the following conditions:

- Its default value is defined as `CURRENT_TIMESTAMP` and no value is specified for the column
- `CURRENT_TIMESTAMP` or any of its synonyms such as `NOW()` is explicitly inserted into the column

In other words, a `TIMESTAMP` column defined to permit `NULL` values auto-initializes only if its definition includes `DEFAULT CURRENT_TIMESTAMP`:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

If the `TIMESTAMP` column permits `NULL` values but its definition does not include `DEFAULT CURRENT_TIMESTAMP`, you must explicitly insert a value corresponding to the current date and time. Suppose that tables `t1` and `t2` have these definitions:

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);
```

To set the `TIMESTAMP` column in either table to the current timestamp at insert time, explicitly assign it that value. For example:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

11.3.6 Fractional Seconds in Time Values

A trailing fractional seconds part is permissible for temporal values in contexts such as literal values, and in the arguments to or return values from some temporal functions. Example:

```
mysql> SELECT MICROSECOND('2010-12-10 14:12:09.019473');
+-----------------------------------------+
| MICROSECOND('2010-12-10 14:12:09.019473') |
+-----------------------------------------+
|                                   19473 |
+-----------------------------------------+
```

However, when MySQL stores a value into a column of any temporal data type, it discards any fractional part and does not store it.

11.3.7 Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of legal values for the resulting type. For example, although `DATE`, `DATETIME`, and `TIMESTAMP` values all can be specified using the same set of formats, the types do not all have the same range of values. `TIMESTAMP` values cannot be earlier than `1970` UTC or later than `'2038-01-19 03:14:07'` UTC. This means that a date such as `'1968-01-01'`, while legal as a `DATE` or `DATETIME` value, is not valid as a `TIMESTAMP` value and is converted to `0`.

Conversion of `DATE` values:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Conversion to a `DATETIME` or `TIMESTAMP` value adds a time part of `'00:00:00'` because the `DATE` value contains no time information.
- Conversion to a `TIME` value is not useful; the result is `'00:00:00'`.

Conversion of `DATETIME` and `TIMESTAMP` values:

- Conversion to a `DATE` value discards the time part because the `DATE` type contains no time information.
- Conversion to a `TIME` value discards the date part because the `TIME` type contains no date information.

Conversion of `TIME` values:

MySQL converts a time value to a date or date-and-time value by parsing the string value of the time as a date or date-and-time. This is unlikely to be useful. For example, `'23:12:31'` interpreted as a date becomes `'2023-12-31'`. Time values not valid as dates become `'0000-00-00'` or `NULL`.

Prior to MySQL 5.0.42, when `DATE` values are compared with `DATETIME` values, the time portion of the `DATETIME` value is ignored, or the comparison could be performed as a string compare. Starting from MySQL 5.0.42, a `DATE` value is coerced to the `DATETIME` type by adding the time portion as `'00:00:00'`. To mimic the old behavior, use the `CAST()` function to cause the comparison operands to be treated as previously. For example:

```
date_col = CAST(datetime_col AS DATE)
```

As of MySQL 5.0.8, conversion of `TIME` or `DATETIME` values to numeric form (for example, by adding `+0`) results in a double-precision value with a microseconds part of `.000000`:

```
mysql> SELECT CURTIME(), CURTIME()+0;
+-----------+---------------+
| CURTIME() | CURTIME()+0   |
+-----------+---------------+
| 10:41:36  | 104136.000000 |
+-----------+---------------+
mysql> SELECT NOW(), NOW()+0;
+---------------------+-----------------------+
| NOW()               | NOW()+0               |
+---------------------+-----------------------+
| 2007-11-30 10:41:47 | 20071130104147.000000 |
+---------------------+-----------------------+
```

Before MySQL 5.0.8, the conversion results in an integer value with no microseconds part.

11.3.8 Two-Digit Years in Dates

Date values with two-digit years are ambiguous because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For `DATETIME`, `DATE`, and `TIMESTAMP` types, MySQL interprets dates specified with ambiguous year values using these rules:

- Year values in the range `00-69` are converted to `2000-2069`.
- Year values in the range `70-99` are converted to `1970-1999`.

For `YEAR`, the rules are the same, with this exception: A numeric `00` inserted into `YEAR(4)` results in `0000` rather than `2000`. To specify zero for `YEAR(4)` and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing four-digit year values.

`ORDER BY` properly sorts `YEAR` values that have two-digit years.

Some functions like `MIN()` and `MAX()` convert a `YEAR` to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the `YEAR` to four-digit year format.

11.4 String Types

The string types are `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`. This section describes how these types work and how to use them in your queries. For string type storage requirements, see Section 11.7, “Data Type Storage Requirements”.

11.4.1 The CHAR and VARCHAR Types

The `CHAR` and `VARCHAR` types are similar, but differ in the way they are stored and retrieved. As of MySQL 5.0.3, they also differ in maximum length and in whether trailing spaces are retained.

The `CHAR` and `VARCHAR` types are declared with a length that indicates the maximum number of characters you want to store. For example, `CHAR(30)` can hold up to 30 characters.

The length of a `CHAR` column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When `CHAR` values are stored, they are right-padded with spaces to the specified length. When `CHAR` values are retrieved, trailing spaces are removed.

Values in `VARCHAR` columns are variable-length strings. The length can be specified as a value from 0 to 255 before MySQL 5.0.3, and 0 to 65,535 in 5.0.3 and later versions. The effective maximum length of a `VARCHAR` in MySQL 5.0.3 and later is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. See Section C.7.4, “Limits on Table Column Count and Row Size”.

In contrast to `CHAR`, `VARCHAR` values are stored as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

If strict SQL mode is not enabled and you assign a value to a `CHAR` or `VARCHAR` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, “Server SQL Modes”.

For `VARCHAR` columns, trailing spaces in excess of the column length are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For `CHAR` columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode.

`VARCHAR` values are not padded when they are stored. Handling of trailing spaces is version-dependent. As of MySQL 5.0.3, trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL. Before MySQL 5.0.3, trailing spaces are removed from values when they are stored into a `VARCHAR` column; this means that the spaces also are absent from retrieved values.

Before MySQL 5.0.3, if you need a data type for which trailing spaces are not removed, consider using a `BLOB` or `TEXT` type. Also, if you want to store binary values such as results from an encryption or compression function that might contain arbitrary byte values, use a `BLOB` column rather than a `CHAR` or `VARCHAR` column, to avoid potential problems with trailing space removal that would change data values.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns (assuming that the column uses a single-byte character set such as latin1).

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

The values shown as stored in the last row of the table apply *only when not using strict mode*; if MySQL is running in strict mode, values that exceed the column length are *not stored*, and an error results.

If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(', v, ')'), CONCAT('(', c, ')') FROM vc;
+---------------------+---------------------+
| CONCAT('(', v, ')') | CONCAT('(', c, ')') |
+---------------------+---------------------+
| (ab  )              | (ab)                |
+---------------------+---------------------+
1 row in set (0.06 sec)
```

Values in CHAR and VARCHAR columns are sorted and compared according to the character set collation assigned to the column.

All MySQL collations are of type PADSPACE. This means that all CHAR, VARCHAR, and TEXT values in MySQL are compared without regard to any trailing spaces. “Comparison” in this context does not include the LIKE pattern-matching operator, for which trailing spaces are significant. For example:

```
mysql> CREATE TABLE names (myname CHAR(10));
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO names VALUES ('Monty');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Monty', myname = 'Monty  ' FROM names;
+------------------+--------------------+
| myname = 'Monty' | myname = 'Monty  ' |
+------------------+--------------------+
|                1 |                  1 |
+------------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT myname LIKE 'Monty', myname LIKE 'Monty  ' FROM names;
+---------------------+----------------------+
| myname LIKE 'Monty' | myname LIKE 'Monty  ' |
+---------------------+----------------------+
|                   1 |                    0 |
+---------------------+----------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
1 row in set (0.00 sec)
```

This is true for all MySQL versions, and it makes no difference whether your version trims trailing spaces from VARCHAR values before storing them. Nor does the server SQL mode make any difference in this regard.

Note

For more information about MySQL character sets and collations, see Section 10.1, “Character Set Support”. For additional information about storage requirements, see Section 11.7, “Data Type Storage Requirements”.

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error.

11.4.2 The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than nonbinary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR, except that the length for BINARY and VARBINARY is a length in bytes rather than in characters.

The BINARY and VARBINARY data types are distinct from the CHAR BINARY and VARCHAR BINARY data types. For the latter types, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains nonbinary character strings rather than binary byte strings. For example, CHAR(5) BINARY is treated as CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin, assuming that the default character set is latin1. This differs from BINARY(5), which stores 5-bytes binary strings that have no character set or collation. For information about differences between nonbinary string binary collations and binary strings, see Section 10.1.7.6, “The _bin and binary Collations”.

If strict SQL mode is not enabled and you assign a value to a BINARY or VARBINARY column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, “Server SQL Modes”.

When BINARY values are stored, they are right-padded with the pad value to the specified length. The pad value and how it is handled is version specific:

- As of MySQL 5.0.15, the pad value is 0x00 (the zero byte). Values are right-padded with 0x00 on insert, and no trailing bytes are removed on select. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 bytes and spaces are different in comparisons, with 0x00 < space.

  Example: For a BINARY(3) column, 'a ' becomes 'a \0' when inserted. 'a\0' becomes 'a\0\0' when inserted. Both inserted values remain unchanged when selected.

- Before MySQL 5.0.15, the pad value is space. Values are right-padded with space on insert, and trailing spaces are removed on select. Trailing spaces are ignored in comparisons, including ORDER BY and DISTINCT operations. 0x00 bytes and spaces are different in comparisons, with 0x00 < space.

  Example: For a BINARY(3) column, 'a ' becomes 'a ' when inserted and 'a' when selected. 'a\0' becomes 'a\0 ' when inserted and 'a\0' when selected.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For VARBINARY, there is no padding on insert and no bytes are stripped on select. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 bytes and spaces are different in comparisons, with 0x00 < space. (Exceptions: Before MySQL 5.0.3, trailing spaces are removed when values are stored. Before MySQL 5.0.15, trailing 0x00 bytes are removed for ORDER BY operations.)

Note: The InnoDB storage engine continues to preserve trailing spaces in BINARY and VARBINARY column values through MySQL 5.0.18. Beginning with MySQL 5.0.19, InnoDB uses trailing space characters in making comparisons as do other MySQL storage engines.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a\0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the BINARY data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how 0x00-padding of BINARY values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+--------+---------+-------------+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+--------+---------+-------------+
| 610000 |       0 |           1 |
+--------+---------+-------------+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use VARBINARY or one of the BLOB data types instead.

11.4.3 The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold. The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements. See Section 11.7, “Data Type Storage Requirements”.

BLOB values are treated as binary strings (byte strings). They have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. TEXT values are treated as nonbinary strings (character strings). They have a character set, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a BLOB or TEXT column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, “Server SQL Modes”.

Beginning with MySQL 5.0.60, truncation of excess trailing spaces from values to be inserted into TEXT columns always generates a warning, regardless of the SQL mode.

For TEXT and BLOB columns, there is no padding on insert and no bytes are stripped on select.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If a TEXT column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for BLOB columns.

In most respects, you can regard a BLOB column as a VARBINARY column that can be as large as you like. Similarly, you can regard a TEXT column as a VARCHAR column. BLOB and TEXT differ from VARBINARY and VARCHAR in the following ways:

- There is no trailing-space removal for BLOB and TEXT columns when values are stored or retrieved. Before MySQL 5.0.3, this differs from VARBINARY and VARCHAR, for which trailing spaces are removed when values are stored.

  On comparisons, TEXT is space extended to fit the compared object, exactly like CHAR and VARCHAR.

- For indexes on BLOB and TEXT columns, you must specify an index prefix length. For CHAR and VARCHAR, a prefix length is optional. See Section 8.3.4, “Column Indexes”.

- BLOB and TEXT columns cannot have DEFAULT values.

If you use the BINARY attribute with a TEXT data type, the column is assigned the binary collation of the column character set.

LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values can be extremely long, you might encounter some constraints in using them:

- Only the first max_sort_length bytes of the column are used when sorting. The default value of max_sort_length is 1024. You can make more bytes significant in sorting or grouping by increasing the value of max_sort_length at server startup or runtime. Any client can change the value of its session max_sort_length variable:

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
    -> ORDER BY comment;
```

- Instances of BLOB or TEXT columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the MEMORY storage engine does not support those data types (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Use of disk incurs a performance penalty, so include BLOB or TEXT columns in the query result only if they are really needed. For example, avoid using SELECT *, which selects all columns.

- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the max_allowed_packet variable, but you must do so for both the server and your client program. For example, both mysql and mysqldump enable you to change the client-side max_allowed_packet value. See Section 8.12.2, “Tuning Server Parameters”, Section 4.5.1, “mysql — The MySQL Command-Line Tool”, and Section 4.5.4, “mysqldump — A Database Backup Program”. You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see Section 11.7, “Data Type Storage Requirements”

Each BLOB or TEXT value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In some cases, it may be desirable to store binary data such as media files in BLOB or TEXT columns. You may find MySQL's string handling functions useful for working with such data. See Section 12.5, “String Functions”. For security and other reasons, it is usually preferable to do so using application code rather than giving application users the FILE privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (http://forums.mysql.com/).

11.4.4 The ENUM Type

An ENUM is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time.

An enumeration value must be a quoted string literal; it may not be an expression, even one that evaluates to a string value. For example, you can create a table with an ENUM column like this:

```
CREATE TABLE sizes (
    name ENUM('small', 'medium', 'large')
);
```

However, this version of the previous CREATE TABLE statement does *not* work:

```
CREATE TABLE sizes (
    c1 ENUM('small', CONCAT('med','ium'), 'large')
);
```

You also may not employ a user variable as an enumeration value. This pair of statements do *not* work:

```
SET @mysize = 'medium';

CREATE TABLE sizes (
    name ENUM('small', @mysize, 'large')
);
```

If you wish to use a number as an enumeration value, you must enclose it in quotation marks. If the quotation marks are omitted, the number is regarded as an index. For this and other reasons—as explained later in this section—we strongly recommend that you do *not* use numbers as enumeration values.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

The value may also be the empty string (`''`) or NULL under certain circumstances:

- If you insert an invalid value into an ENUM (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numeric value 0. More about this later.

  If strict SQL mode is enabled, attempts to insert invalid ENUM values result in an error.

- If an ENUM column is declared to permit NULL, the NULL value is a legal value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of permitted values.

Each enumeration value has an index:

- Values from the list of permissible elements in the column specification are numbered beginning with 1.

- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the `NULL` value is `NULL`.
- The term “index” here refers only to position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as `ENUM('one', 'two', 'three')` can have any of the values shown here. The index of each value is also shown.

Value	Index
`NULL`	`NULL`
`''`	0
`'one'`	1
`'two'`	2
`'three'`	3

An `ENUM` column can have a maximum of 65,535 distinct elements. (The practical limit is less than 3000.) A table can have no more than 255 unique element list definitions among its `ENUM` and `SET` columns considered as a group. For more information on these limits, see Section C.7.5, “Limits Imposed by .frm File Structure”.

Trailing spaces are automatically deleted from `ENUM` member values in the table definition when a table is created.

When retrieved, values stored into an `ENUM` column are displayed using the lettercase that was used in the column definition. Note that `ENUM` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you retrieve an `ENUM` value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an `ENUM` column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

If you store a number into an `ENUM` column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with `LOAD DATA`, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an `ENUM` column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of `'0'`, `'1'`, and `'2'`, but numeric index values of `1`, `2`, and `3`:

```
numbers ENUM('0','1','2')
```

If you store `2`, it is interpreted as an index value, and becomes `'1'` (the value with index 2). If you store `'2'`, it matches an enumeration value, so it is stored as `'2'`. If you store `'3'`, it does not match any enumeration value, so it is treated as an index and becomes `'2'` (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
+---------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| numbers |
+---------+
| 1       |
| 2       |
| 2       |
+---------+
```

`ENUM` values are sorted according to the order in which the enumeration members were listed in the column specification. (In other words, `ENUM` values are sorted according to their index numbers.) For example, `'a'` sorts before `'b'` for `ENUM('a', 'b')`, but `'b'` sorts before `'a'` for `ENUM('b', 'a')`. The empty string sorts before nonempty strings, and `NULL` values sort before all other enumeration values. To prevent unexpected results, specify the `ENUM` list in alphabetic order. You can also use `ORDER BY CAST(col AS CHAR)` or `ORDER BY CONCAT(col)` to make sure that the column is sorted lexically rather than by index number.

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `ENUM` values, the cast operation causes the index number to be used.

To determine all possible values for an `ENUM` column, use `SHOW COLUMNS FROM tbl_name LIKE 'enum_col'` and parse the `ENUM` definition in the `Type` column of the output.

In the C API, `ENUM` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see Section 20.6.5, “C API Data Structures”.

11.4.5 The SET Type

A `SET` is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. `SET` column values that consist of multiple set members are specified with members separated by commas (“`,`”). A consequence of this is that `SET` member values should not themselves contain commas.

For example, a column specified as `SET('one', 'two') NOT NULL` can have any of these values:

```
''
'one'
'two'
'one,two'
```

A `SET` column can have a maximum of 64 distinct members. A table can have no more than 255 unique element list definitions among its `ENUM` and `SET` columns considered as a group. For more information on this limit, see Section C.7.5, “Limits Imposed by .frm File Structure”.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

Trailing spaces are automatically deleted from `SET` member values in the table definition when a table is created.

When retrieved, values stored in a `SET` column are displayed using the lettercase that was used in the column definition. Note that `SET` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores `SET` values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a `SET` value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a `SET` column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a','b','c','d'), the members have the following decimal and binary values.

SET Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. For example, suppose that a column is specified as SET('a','b','c','d'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Then all these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
5 rows in set (0.04 sec)
```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+---------+------+-------------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
6 rows in set (0.01 sec)
```

If strict SQL mode is enabled, attempts to insert invalid `SET` values result in an error.

`SET` values are sorted numerically. `NULL` values sort before non-`NULL` `SET` values.

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `SET` values, the cast operation causes the numeric value to be used.

Normally, you search for `SET` values using the `FIND_IN_SET()` function or the `LIKE` operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *`set_col`* contains the *`value`* set member. The second is similar, but not the same: It finds rows where *`set_col`* contains *`value`* anywhere, even as a substring of another set member.

The following statements also are legal:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to `'val1,val2'` returns different results than comparing values to `'val2,val1'`. You should specify the values in the same order they are listed in the column definition.

To determine all possible values for a `SET` column, use `SHOW COLUMNS FROM tbl_name LIKE set_col` and parse the `SET` definition in the `Type` column of the output.

In the C API, `SET` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see Section 20.6.5, “C API Data Structures”.

11.5 Extensions for Spatial Data

The Open Geospatial Consortium (OGC) is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data.

The Open Geospatial Consortium publishes the *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at http://www.opengeospatial.org/standards/sfs.

Following the OGC specification, MySQL implements spatial extensions as a subset of the **SQL with Geometry Types** environment. This term refers to an SQL environment that has been extended with a set

This documentation is for an older version. If you're
This documentation is for an older version. If you're

of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describes a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

MySQL spatial extensions enable the generation, storage, and analysis of geographic features:

- Data types for representing spatial values
- Functions for manipulating spatial values
- Spatial indexing for improved access times to spatial columns

Before MySQL 5.0.16, these features are available for `MyISAM` tables only. As of MySQL 5.0.16, `InnoDB`, `NDB`, `BDB`, and `ARCHIVE` also support spatial features.

For spatial columns, `MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. The other storage engines support non-`SPATIAL` indexes, as described in Section 13.1.8, “CREATE INDEX Syntax”.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

The discussion here considers these terms synonymous: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. The term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

The following material covers these topics:

- The spatial data types implemented in MySQL model
- The basis of the spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

For information about functions that operate on spatial data, see Section 12.14, “Spatial Analysis Functions”.

MySQL GIS Conformance and Compatibility

MySQL does not implement the following GIS features:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Additional Metadata Views

  OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` on `LineString` and `MultiLineString` should be called in MySQL as `GLength()`

  The problem is that there is an existing SQL function `Length()` that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context.

Additional Resources

- The Open Geospatial Consortium publishes the *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. The Open Geospatial Consortium (OGC) maintains a Web site at http://www.opengeospatial.org/. The specification is available there at http://www.opengeospatial.org/standards/sfs. It contains additional information relevant to the material here.

- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: http://forums.mysql.com/list.php?23.

11.5.1 Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINESTRING`, and `POLYGON`) restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, and `GEOMETRYCOLLECTION`) restrict collection members to those having a particular geometry type.

MySQL spatial data types have their basis in the OpenGIS geometry model, described in Section 11.5.2, “The OpenGIS Geometry Model”. For examples showing how to use spatial data types in MySQL, see Section 11.5.3, “Using Spatial Data”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

11.5.2 The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

11.5.2.1 The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (noninstantiable)
  - `Point` (instantiable)
  - `Curve` (noninstantiable)
    - `LineString` (instantiable)
      - `Line`
      - `LinearRing`
  - `Surface` (noninstantiable)
    - `Polygon` (instantiable)
  - `GeometryCollection` (instantiable)
    - `MultiPoint` (instantiable)
    - `MultiCurve` (noninstantiable)
      - `MultiLineString` (instantiable)
    - `MultiSurface` (noninstantiable)
      - `MultiPolygon` (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

11.5.2.2 Geometry Class

`Geometry` is the root class of the hierarchy. It is a noninstantiable class but has a number of properties, described in the following list, that are common to all geometry values created from any of the `Geometry` subclasses. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

  In MySQL, the SRID value is an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry. The maximum usable SRID value is $2^{32}-1$. If a larger value is given, only the lower 32 bits are used.
- Its **coordinates** in its Spatial Reference System, represented as double-precision (8-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

  Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geodetic** system (coordinates on the Earth's surface) are different things.
- Its **interior**, **boundary**, and **exterior**.

  Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.
- Its **MBR** (minimum bounding rectangle), or envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```
- Whether the value is **simple** or **nonsimple**. Geometry values of types (`LineString`, `MultiPoint`, `MultiLineString`) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Whether the value is **closed** or **not closed**. Geometry values of types (`LineString`, `MultiString`) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is **empty** or **nonempty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a `NULL` value). An empty geometry is defined to be always simple and has an area of 0.
- Its **dimension**. A geometry can have a dimension of −1, 0, 1, or 2:
  - −1 for an empty geometry.
  - 0 for a geometry with no length and no area.
  - 1 for a geometry with nonzero length and zero area.
  - 2 for a geometry with nonzero area.

  `Point` objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

11.5.2.3 Point Class

A `Point` is a geometry that represents a single location in coordinate space.

`Point` Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.
- On a city map, a `Point` object could represent a bus stop.

`Point` Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.
- The boundary of a `Point` is the empty set.

11.5.2.4 Curve Class

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a noninstantiable class.

`Curve` Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice.
- A `Curve` is closed if its start point is equal to its endpoint.
- The boundary of a closed `Curve` is empty.
- The boundary of a nonclosed `Curve` consists of its two endpoints.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- A `Curve` that is simple and closed is a `LinearRing`.

11.5.2.5 LineString Class

A `LineString` is a `Curve` with linear interpolation between points.

`LineString` Examples

- On a world map, `LineString` objects could represent rivers.
- In a city map, `LineString` objects could represent streets.

`LineString` Properties

- A `LineString` has coordinates of segments, defined by each consecutive pair of points.
- A `LineString` is a `Line` if it consists of exactly two points.
- A `LineString` is a `LinearRing` if it is both closed and simple.

11.5.2.6 Surface Class

A `Surface` is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is `Polygon`.

`Surface` Properties

- A `Surface` is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple `Surface` as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

11.5.2.7 Polygon Class

A `Polygon` is a planar `Surface` representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the `Polygon`.

`Polygon` Examples

- On a region map, `Polygon` objects could represent forests, districts, and so on.

`Polygon` Assertions

- The boundary of a `Polygon` consists of a set of `LinearRing` objects (that is, `LineString` objects that are both simple and closed) that make up its exterior and interior boundaries.
- A `Polygon` has no rings that cross. The rings in the boundary of a `Polygon` may intersect at a `Point`, but only as a tangent.
- A `Polygon` has no lines, spikes, or punctures.
- A `Polygon` has an interior that is a connected point set.
- A `Polygon` may have holes. The exterior of a `Polygon` with holes is not connected. Each hole defines a connected component of the exterior.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The preceding assertions make a `Polygon` a simple geometry.

11.5.2.8 GeometryCollection Class

A `GeometryCollection` is a geometry that is a collection of one or more geometries of any class.

All the elements in a `GeometryCollection` must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a `GeometryCollection`, although the subclasses of `GeometryCollection` described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a `MultiPoint` may contain only `Point` elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

11.5.2.9 MultiPoint Class

A `MultiPoint` is a geometry collection composed of `Point` elements. The points are not connected or ordered in any way.

`MultiPoint` Examples

- On a world map, a `MultiPoint` could represent a chain of small islands.
- On a city map, a `MultiPoint` could represent the outlets for a ticket office.

`MultiPoint` Properties

- A `MultiPoint` is a zero-dimensional geometry.
- A `MultiPoint` is simple if no two of its `Point` values are equal (have identical coordinate values).
- The boundary of a `MultiPoint` is the empty set.

11.5.2.10 MultiCurve Class

A `MultiCurve` is a geometry collection composed of `Curve` elements. `MultiCurve` is a noninstantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `MultiCurve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

11.5.2.11 MultiLineString Class

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

11.5.2.12 MultiSurface Class

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a noninstantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Two `MultiSurface` surfaces have no interiors that intersect.
- Two `MultiSurface` elements have boundaries that intersect at most at a finite number of points.

11.5.2.13 MultiPolygon Class

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.
- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.
- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each `Curve` in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every `Curve` in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

11.5.3 Using Spatial Data

This section describes how to create tables that include spatial data type columns, and how to manipulate spatial information.

11.5.3.1 Supported Spatial Data Formats

Two standard spatial data formats are used to represent geometry objects in queries:

- Well-Known Text (WKT) format

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

There are functions available to convert between different data formats; see Section 12.14.6, “Geometry Format Conversion Functions”.

Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of geometry values is designed for exchanging geometry data in ASCII form. The OpenGIS specification provides a Backus-Naur grammar that specifies the formal production rules for writing WKT values (see Section 11.5, “Extensions for Spatial Data”).

Examples of WKT representations of geometry objects:

- A `Point`:

```
POINT(15 20)
```

  The point coordinates are specified with no separating comma. This differs from the syntax for the SQL `Point()` function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both extract the X-coordinate from a `Point` object. The first produces the object directly using the `Point()` function. The second uses a WKT representation converted to a `Point` with `GeomFromText()`.

```
mysql> SELECT X(Point(15, 20));
+------------------+
| X(POINT(15, 20)) |
+------------------+
|               15 |
+------------------+

mysql> SELECT X(GeomFromText('POINT(15 20)'));
+---------------------------------+
| X(GeomFromText('POINT(15 20)')) |
+---------------------------------+
|                              15 |
+---------------------------------+
```

- A `LineString` with four points:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

  The point coordinate pairs are separated by commas.

- A `Polygon` with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A `MultiPoint` with three `Point` values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A `MultiLineString` with two `LineString` values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- A `MultiPolygon` with two `Polygon` values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A `GeometryCollection` consisting of two `Point` values and one `LineString`:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation of geometric values is used for exchanging geometry data as binary streams represented by `BLOB` values containing geometric WKB information. This format is defined by the OpenGIS specification (see Section 11.5, “Extensions for Spatial Data”). It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to `POINT(1 1)` consists of this sequence of 21 bytes, each represented by two hex digits:

```
0101000000000000000000F03F000000000000F03F
```

The sequence consists of these components:

```
Byte order:   01
WKB type:     01000000
X coordinate: 000000000000F03F
Y coordinate: 000000000000F03F
```

Component representation is as follows:

- The byte order is either 1 or 0 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.
- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`.
- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values have more complex data structures, as detailed in the OpenGIS specification.

11.5.3.2 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Spatial columns are supported for `MyISAM`, `InnoDB`, `NDB`, `BDB`, and `ARCHIVE` tables. (Support for storage engines other than `MyISAM` was added in MySQL 5.0.16.) See also the notes about spatial indexes under Section 11.5.3.6, “Creating Spatial Indexes”.

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

11.5.3.3 Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values to internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples use `GeomFromText()` to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

A client application program that wants to use WKB representations of geometry values is responsible for sending correctly formed WKB in queries to the server. There are several ways to satisfy this requirement. For example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
mysql> INSERT INTO geom VALUES
    -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

  Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See Section 20.6.7.53, “mysql_real_escape_string()”.

11.5.3.4 Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them to WKT or WKB format.

- Fetching spatial data in internal format:

  Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- Fetching spatial data in WKT format:

  The `AsText()` function converts a geometry from internal format to a WKT string.

```
SELECT AsText(g) FROM geom;
```

- Fetching spatial data in WKB format:

  The `AsBinary()` function converts a geometry from internal format to a `BLOB` containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

11.5.3.5 Optimizing Spatial Analysis

For `MyISAM` tables, search operations in columns containing spatial data can be optimized using `SPATIAL` indexes. The most typical operations are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for `SPATIAL` indexes on spatial columns. A `SPATIAL` index is built using the minimum bounding rectangle (MBR) of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-`SPATIAL` index, you must declare a prefix for any spatial column except for `POINT` columns.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MyISAM supports both SPATIAL and non-SPATIAL indexes. Other storage engines support non-SPATIAL indexes, as described in Section 13.1.8, “CREATE INDEX Syntax”.

11.5.3.6 Creating Spatial Indexes

For MyISAM tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but using the SPATIAL keyword. Columns in spatial indexes must be declared NOT NULL. The following examples demonstrate how to create spatial indexes:

- With CREATE TABLE:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g)) ENGINE=MyISAM;
```

- With ALTER TABLE:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With CREATE INDEX:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

SPATIAL INDEX creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values is useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see Section 13.1.8, “CREATE INDEX Syntax”.

To drop spatial indexes, use ALTER TABLE or DROP INDEX:

- With ALTER TABLE:

```
ALTER TABLE geom DROP INDEX g;
```

- With DROP INDEX:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table geom contains more than 32,000 geometries, which are stored in the column g of type GEOMETRY. The table also has an AUTO_INCREMENT column fid for storing object ID values.

```
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
|    32376 |
+----------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g) ENGINE=MyISAM;
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

11.5.3.7 Using Spatial Indexes

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` or `MBRWithin()` in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.00 sec)
```

Use `EXPLAIN` to check the way this query is executed:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
         type: range
possible_keys: g
          key: g
      key_len: 32
          ref: NULL
         rows: 50
        Extra: Using where
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 32376
        Extra: Using where
1 row in set (0.00 sec)
```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.46 sec)
```

11.6 Data Type Default Values

The `DEFAULT value` clause in a data type specification indicates a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that you can specify `CURRENT_TIMESTAMP` as the default for a `TIMESTAMP` column. See Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”.

Prior to MySQL 5.0.2, if a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.

If the column cannot take `NULL` as the value, MySQL defines the column with an explicit `DEFAULT` clause, using the implicit default value for the column data type. Implicit defaults are defined as follows:

- For numeric types, the default is `0`, with the exception that for integer or floating-point types declared with the `AUTO_INCREMENT` attribute, the default is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate “zero” value for the type. For the first `TIMESTAMP` column in a table, the default value is the current date and time. See Section 11.3, “Date and Time Types”.
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

`BLOB` and `TEXT` columns cannot be assigned a default value.

As of MySQL 5.0.2, if a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause. This is the same as before 5.0.2.

If the column cannot take `NULL` as the value, MySQL defines the column with no explicit `DEFAULT` clause. Exception: If the column is defined as part of a `PRIMARY KEY` but not explicitly as `NOT NULL`, MySQL creates it as a `NOT NULL` column (because `PRIMARY KEY` columns must be `NOT NULL`), but also assigns it a `DEFAULT` clause using the implicit default value. To prevent this, include an explicit `NOT NULL` in the definition of any `PRIMARY KEY` column.

For data entry into a `NOT NULL` column that has no explicit `DEFAULT` clause, if an `INSERT` or `REPLACE` statement includes no value for the column, or an `UPDATE` statement sets the column to `NULL`, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is enabled, an error occurs for transactional tables and the statement is rolled back. For nontransactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.
- If strict mode is not enabled, MySQL sets the column to the implicit default value for the column data type.

Suppose that a table `t` is defined as follows:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, `i` has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because `DEFAULT(i)` cannot produce a value:

```
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

See Section 5.1.7, “Server SQL Modes”.

For a given table, you can use the `SHOW CREATE TABLE` statement to see which columns have an explicit `DEFAULT` clause.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

11.7 Data Type Storage Requirements

The storage requirements for data vary, according to the storage engine being used for the table in question. Different storage engines use different methods for recording the raw data and different data types. In addition, some engines may compress the information in a given row, either on a column or entire row basis, making calculation of the storage requirements for a given table or column structure.

However, all storage engines must communicate and exchange information on a given row within a table using the same structure, and this information is consistent, irrespective of the storage engine used to write the information to disk.

This sections includes some guideliness and information for the storage requirements for each data type supported by MySQL, including details for the internal format and the sizes used by storage engines that used a fixed size representation for different types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes `BLOB` or `TEXT` columns, which contribute only 9 to 12 bytes toward this size. For `BLOB` and `TEXT` data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this data in different ways, according to the method they use for handling the corresponding types. For more information, see Chapter 14, *Storage Engines*, and Section C.7.4, “Limits on Table Column Count and Row Size”.

Storage Requirements for InnoDB Tables

See Section 14.2.10.5, “Physical Row Structure” for information about storage requirements for `InnoDB` tables.

Storage Requirements for NDBCLUSTER Tables

Important

For tables using the `NDBCLUSTER` storage engine, there is the factor of *4-byte alignment* to be taken into account when calculating storage requirements. This means that all `NDB` data storage is done in multiples of 4 bytes. Thus, a column value that would take 15 bytes in a table using a storage engine other than `NDB`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

> requires 16 bytes in an NDB table. This requirement applies in addition to any other considerations that are discussed in this section. For example, in NDBCLUSTER tables, the TINYINT, SMALLINT, MEDIUMINT, and INTEGER (INT) column types each require 4 bytes storage per record due to the alignment factor.
>
> An exception to this rule is the BIT type, which is *not* 4-byte aligned. In MySQL Cluster tables, a BIT(*M*) column takes *M* bits of storage space. However, if a table definition contains 1 or more BIT columns (up to 32 BIT columns), then NDBCLUSTER reserves 4 bytes (32 bits) per row for these. If a table definition contains more than 32 BIT columns (up to 64 such columns), then NDBCLUSTER reserves 8 bytes (that is, 64 bits) per row.
>
> In addition, while a NULL itself does not require any storage space, NDBCLUSTER reserves 4 bytes per row if the table definition contains any columns defined as NULL, up to 32 NULL columns. (If a MySQL Cluster table is defined with more than 32 NULL columns up to 64 NULL columns, then 8 bytes per row is reserved.)

When calculating storage requirements for MySQL Cluster tables, you must also remember that every table using the NDBCLUSTER storage engine requires a primary key; if no primary key is defined by the user, then a “hidden” primary key will be created by NDB. This hidden primary key consumes 31-35 bytes per table record.

You may find the ndb_size.pl utility to be useful for estimating NDB storage requirements. This Perl script connects to a current MySQL (non-Cluster) database and creates a report on how much space that database would require if it used the NDBCLUSTER storage engine. See Section 17.4.18, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”, for more information.

Storage Requirements for Numeric Types

Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(*p*)	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
FLOAT	4 bytes
DOUBLE [PRECISION], REAL	8 bytes
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	Varies; see following discussion
BIT(*M*)	approximately (*M*+7)/8 bytes

The storage requirements for DECIMAL (and NUMERIC) are version-specific:

As of MySQL 5.0.3, values for DECIMAL columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

Leftover Digits	Number of Bytes
0	0

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Leftover Digits	Number of Bytes
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

Before MySQL 5.0.3, `DECIMAL` columns are represented as strings and storage requirements are: *`M`*+2 bytes if *`D`* > 0, *`M`*+1 bytes if *`D`* = 0, *`D`*+2 if *`M < D`*

Storage Requirements for Date and Time Types

Data Type	Storage Required
`DATE`	3 bytes
`TIME`	3 bytes
`DATETIME`	8 bytes
`TIMESTAMP`	4 bytes
`YEAR`	1 byte

For details about internal representation of temporal values, see MySQL Internals: Important Algorithms and Structures.

Storage Requirements for String Types

In the following table, *`M`* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *`L`* represents the actual length in bytes of a given string value.

Data Type	Storage Required
`CHAR(`*`M`*`)`	*`M`* × *`w`* bytes, 0 <= *`M`* <= 255, where *`w`* is the number of bytes required for the maximum-length character in the character set. See Section 14.2.10.5, “Physical Row Structure” for information about `CHAR` data type storage requirements for `InnoDB` tables.
`BINARY(`*`M`*`)`	*`M`* bytes, 0 <= *`M`* <= 255
`VARCHAR(`*`M`*`)`, `VARBINARY(`*`M`*`)`	*`L`* + 1 bytes if column values require 0 − 255 bytes, *`L`* + 2 bytes if values may require more than 255 bytes
`TINYBLOB`, `TINYTEXT`	*`L`* + 1 bytes, where *`L`* < 2^8
`BLOB`, `TEXT`	*`L`* + 2 bytes, where *`L`* < 2^{16}
`MEDIUMBLOB`, `MEDIUMTEXT`	*`L`* + 3 bytes, where *`L`* < 2^{24}
`LONGBLOB`, `LONGTEXT`	*`L`* + 4 bytes, where *`L`* < 2^{32}
`ENUM('`*`value1`*`','`*`value2`*`',...)`	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Data Type	Storage Required
`SET('value1','value2',...)`	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is *L* (the byte length of the string). For example, storage for a `MEDIUMTEXT` value requires *L* bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular `CHAR`, `VARCHAR`, or `TEXT` column value, you must take into account the character set used for that column and whether the value contains multibyte characters. In particular, when using the `utf8` Unicode character set, you must keep in mind that not all characters use the same number of bytes and can require up to three bytes per character. For a breakdown of the storage used for different categories of `utf8` characters, see Section 10.1.10, “Unicode Support”.

`VARCHAR`, `VARBINARY`, and the `BLOB` and `TEXT` types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length
- The character set used for the column, because some character sets contain multibyte characters

For example, a `VARCHAR(255)` column can hold a string with a maximum length of 255 characters. Assuming that the column uses the `latin1` character set (one byte per character), the actual storage required is the length of the string (*L*), plus one byte to record the length of the string. For the string `'abcd'`, *L* is 4 and the storage requirement is five bytes. If the same column is instead declared to use the `ucs2` double-byte character set, the storage requirement is 10 bytes: The length of `'abcd'` is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of *bytes* that can be stored in a `VARCHAR` or `VARBINARY` column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a `VARCHAR` column that stores multibyte characters, the effective maximum number of *characters* is less. For example, `utf8` characters can require up to three bytes per character, so a `VARCHAR` column that uses the `utf8` character set can be declared to be a maximum of 21,844 characters. See Section C.7.4, “Limits on Table Column Count and Row Size”.

As of MySQL 5.0.3, the `NDBCLUSTER` engine supports only fixed-width columns. This means that a `VARCHAR` column from a table in a MySQL Cluster will behave as follows:

- If the size of the column is fewer than 256 characters, the column requires one byte extra storage per row.
- If the size of the column is 256 characters or more, the column requires two bytes extra storage per row.

The number of bytes required per character varies according to the character set used. For example, if a `VARCHAR(100)` column in a Cluster table uses the `utf8` character set, each character requires 3 bytes storage. This means that each record in such a column takes up 100 × 3 + 1 = 301 bytes for storage, regardless of the length of the string actually stored in any given record. For a `VARCHAR(1000)` column in a table using the `NDBCLUSTER` storage engine with the `utf8` character set, each record will use 1000 × 3 + 2 = 3002 bytes storage; that is, the column is 1,000 characters wide, each character requires 3 bytes storage, and each record has a 2-byte overhead because 1,000 >= 256.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`TEXT` and `BLOB` columns are implemented differently in the NDB Cluster storage engine, wherein each row in a `TEXT` column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2,000 bytes long. This means that the size of a `TEXT` column is 256 if *`size`* <= 256 (where *`size`* represents the size of the row); otherwise, the size is 256 + *`size`* + (2000 − (*`size`* − 256) % 2000).

The size of an `ENUM` object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See Section 11.4.4, “The ENUM Type”.

The size of a `SET` object is determined by the number of different set members. If the set size is *`N`*, the object occupies `(`*`N`*`+7)/8` bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A `SET` can have a maximum of 64 members. See Section 11.4.5, “The SET Type”.

11.8 Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from `1` to `99999`, `MEDIUMINT UNSIGNED` is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

Tables created in MySQL 5.0.3 and above use a new storage format for `DECIMAL` columns. All basic calculations (`+`, `-`, `*`, and `/`) with `DECIMAL` columns are done with precision of 65 decimal (base 10) digits. See Section 11.1.1, “Numeric Type Overview”.

Prior to MySQL 5.0.3, calculations on `DECIMAL` values are performed using double-precision operations. If accuracy is not too important or if speed is the highest priority, the `DOUBLE` type may be good enough. For high precision, you can always convert to a fixed-point type stored in a `BIGINT`. This enables you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

`PROCEDURE ANALYSE` can be used to obtain suggestions for optimal column data types. For more information, see Section 8.4.2.4, “Using PROCEDURE ANALYSE”.

11.9 Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

Other Vendor Type	MySQL Type
`BOOL`	`TINYINT`
`BOOLEAN`	`TINYINT`
`CHARACTER VARYING(`*`M`*`)`	`VARCHAR(`*`M`*`)`
`FIXED`	`DECIMAL`
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`
`INT3`	`MEDIUMINT`
`INT4`	`INT`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Other Vendor Type	MySQL Type
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE t;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| a     | tinyint(1)    | YES  |     | NULL    |       |
| b     | double        | YES  |     | NULL    |       |
| c     | mediumtext    | YES  |     | NULL    |       |
| d     | decimal(10,0) | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 12 Functions and Operators

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

Expressions can be used at several points in SQL statements, such as in the `ORDER BY` or `HAVING` clauses of `SELECT` statements, in the `WHERE` clause of a `SELECT`, `DELETE`, or `UPDATE` statement, or in `SET` statements. Expressions can be written using literal values, column values, `NULL`, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the functions and operators that are permitted for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in Section 18.2, “Using Stored Routines (Procedures and Functions)”, and Section 21.2, “Adding New Functions to MySQL”. See Section 9.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for a particular function or operator.

Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` option. (See Section 5.1.7, “Server SQL Modes”.) Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the `mysql` program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
        -> 2
```

12.1 Function and Operator Reference

Table 12.1 Functions/Operators

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ADDDATE()`	Add time values (intervals) to a date value
`ADDTIME()`	Add time
`AES_DECRYPT()`	Decrypt using AES
`AES_ENCRYPT()`	Encrypt using AES
`AND, &&`	Logical AND

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
Area()	Return Polygon or MultiPolygon area
AsBinary(), AsWKB()	Convert from internal geometry format to WKB
ASCII()	Return numeric value of left-most character
ASIN()	Return the arc sine
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:=	Assign a value
AsText(), AsWKT()	Convert from internal geometry format to WKT
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
AVG()	Return the average value of the argument
BENCHMARK()	Repeatedly execute an expression
BETWEEN ... AND ...	Check whether a value is within a range of values
BIN()	Return a string containing binary representation of a number
BINARY	Cast a string to a binary string
BIT_AND()	Return bitwise AND
BIT_COUNT()	Return the number of bits that are set
BIT_LENGTH()	Return length of argument in bits
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
&	Bitwise AND
~	Bitwise inversion
|	Bitwise OR
^	Bitwise XOR
CASE	Case operator
CAST()	Cast a value as a certain type
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
Centroid()	Return centroid as a point
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CHARSET()	Return the character set of the argument
COALESCE()	Return the first non-NULL argument
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
COMPRESS()	Return result as a binary string
CONCAT()	Return concatenated string

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
CONCAT_WS()	Return concatenate with separator
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
Contains()	Whether MBR of one geometry contains MBR of another
CONV()	Convert numbers between different number bases
CONVERT()	Cast a value as a certain type
CONVERT_TZ()	Convert from one timezone to another
COS()	Return the cosine
COT()	Return the cotangent
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
CRC32()	Compute a cyclic redundancy check value
Crosses()	Whether one geometry crosses another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
CURTIME()	Return the current time
DATABASE()	Return the default (current) database name
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
DECODE()	Decodes a string encrypted using ENCODE()
DEFAULT()	Return the default value for a table column
DEGREES()	Convert radians to degrees
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
Dimension()	Dimension of geometry
Disjoint()	Whether MBRs of two geometries are disjoint
DIV	Integer division

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
/	Division operator
ELT()	Return string at index number
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
EndPoint()	End Point of LineString
Envelope()	Return MBR of geometry
=	Equal operator
<=>	NULL-safe equal to operator
Equals()	Whether MBRs of two geometries are equal
EXP()	Raise to the power of
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
ExteriorRing()	Return exterior ring of Polygon
EXTRACT()	Extract part of a date
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FLOOR()	Return the largest integer value not greater than the argument
FORMAT()	Return a number formatted to specified number of decimal places
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format UNIX timestamp as a date
GeomCollFromText(), GeometryCollectionFromText()	Return geometry collection from WKT
GeomCollFromWKB(), GeometryCollectionFromWKB()	Return geometry collection from WKB
GeometryCollection()	Construct geometry collection from geometries
GeometryN()	Return N-th geometry from geometry collection
GeometryType()	Return name of geometry type
GeomFromText(), GeometryFromText()	Return geometry from WKT
GeomFromWKB(), GeometryFromWKB()	Return geometry from WKB
GET_FORMAT()	Return a date format string
GET_LOCK()	Get a named lock
GLength()	Return length of LineString
>	Greater than operator

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
>=	Greater than or equal operator
GREATEST()	Return the largest argument
GROUP_CONCAT()	Return a concatenated string
HEX()	Return a hexadecimal representation of a decimal or string value
HOUR()	Extract the hour
IF()	If/else construct
IFNULL()	Null if/else construct
IN()	Check whether a value is within a set of values
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
InteriorRingN()	Return N-th interior ring of Polygon
Intersects()	Whether MBRs of two geometries intersect
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS_FREE_LOCK()	Whether the named lock is free
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
IS_USED_LOCK()	Whether the named lock is in use; return connection identifier if true
IsClosed()	Whether a geometry is closed and simple
IsEmpty()	Placeholder function
ISNULL()	Test whether the argument is NULL
IsSimple()	Whether a geometry is simple
LAST_DAY	Return the last day of the month for the argument
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
LCASE()	Synonym for LOWER()
LEAST()	Return the smallest argument
LEFT()	Return the leftmost number of characters as specified
<<	Left shift
LENGTH()	Return the length of a string in bytes
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
LineFromText(), LineStringFromText()	Construct LineString from WKT
LineFromWKB(), LineStringFromWKB()	Construct LineString from WKB
LineString()	Construct LineString from Point values
LN()	Return the natural logarithm of the argument
LOAD_FILE()	Load the named file
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
LOCATE()	Return the position of the first occurrence of substring
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
MATCH	Perform full-text search
MAX()	Return the maximum value
MBRContains()	Whether MBR of one geometry contains MBR of another
MBRDisjoint()	Whether MBRs of two geometries are disjoint
MBREqual()	Whether MBRs of two geometries are equal
MBRIntersects()	Whether MBRs of two geometries intersect
MBROverlaps()	Whether MBRs of two geometries overlap
MBRTouches()	Whether MBRs of two geometries touch
MBRWithin()	Whether MBR of one geometry is within MBR of another
MD5()	Calculate MD5 checksum
MICROSECOND()	Return the microseconds from argument
MID()	Return a substring starting from the specified position
MIN()	Return the minimum value
-	Minus operator
MINUTE()	Return the minute from the argument

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
MLineFromText(), MultiLineStringFromText()	Construct MultiLineString from WKT
MLineFromWKB(), MultiLineStringFromWKB()	Construct MultiLineString from WKB
MOD()	Return the remainder
%, MOD	Modulo operator
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
MPointFromText(), MultiPointFromText()	Construct MultiPoint from WKT
MPointFromWKB(), MultiPointFromWKB()	Construct MultiPoint from WKB
MPolyFromText(), MultiPolygonFromText()	Construct MultiPolygon from WKT
MPolyFromWKB(), MultiPolygonFromWKB()	Construct MultiPolygon from WKB
MultiLineString()	Contruct MultiLineString from LineString values
MultiPoint()	Construct MultiPoint from Point values
MultiPolygon()	Construct MultiPolygon from Polygon values
NAME_CONST()	Causes the column to have the given name
NOT, !	Negates value
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOW()	Return the current date and time
NULLIF()	Return NULL if expr1 = expr2
NumGeometries()	Return number of geometries in geometry collection
NumInteriorRings()	Return number of interior rings in Polygon
NumPoints()	Return number of points in LineString
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
OLD_PASSWORD()	Return the value of the pre-4.1 implementation of PASSWORD
||, OR	Logical OR
ORD()	Return character code for leftmost character of the argument
Overlaps()	Whether MBRs of two geometries overlap
PASSWORD()	Calculate and return a password string
PERIOD_ADD()	Add a period to a year-month

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
PERIOD_DIFF()	Return the number of months between periods
PI()	Return the value of pi
+	Addition operator
Point()	Construct Point from coordinates
PointFromText()	Construct Point from WKT
PointFromWKB()	Construct Point from WKB
PointN()	Return N-th point from LineString
PolyFromText(), PolygonFromText()	Construct Polygon from WKT
PolyFromWKB(), PolygonFromWKB()	Construct Polygon from WKB
Polygon()	Construct Polygon from LineString arguments
POSITION()	Synonym for LOCATE()
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
PROCEDURE ANALYSE()	Analyze the results of a query
QUARTER()	Return the quarter from a date argument
QUOTE()	Escape the argument for use in an SQL statement
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
REGEXP	Pattern matching using regular expressions
RELEASE_LOCK()	Releases the named lock
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
>>	Right shift
RLIKE	Synonym for REGEXP
ROUND()	Round the argument
ROW_COUNT()	The number of rows updated
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SCHEMA()	Synonym for DATABASE()
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
SESSION_USER()	Synonym for USER()
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
SLEEP()	Sleep for a number of seconds
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
SQRT()	Return the square root of the argument
SRID()	Return spatial reference system ID for geometry
StartPoint()	Start Point of LineString
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
STR_TO_DATE()	Convert a string to a date
STRCMP()	Compare two strings
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBTIME()	Subtract times
SUM()	Return the sum
SYSDATE()	Return the time at which the function executes
SYSTEM_USER()	Synonym for USER()
TAN()	Return the tangent of the argument
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
*	Multiplication operator
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
Touches()	Whether one geometry touches another
TRIM()	Remove leading and trailing spaces
TRUNCATE()	Truncate to specified number of decimal places
UCASE()	Synonym for UPPER()
-	Change the sign of the argument

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
UNHEX()	Return a string containing hex representation of a number
UNIX_TIMESTAMP()	Return a UNIX timestamp
UPPER()	Convert to uppercase
USER()	The user name and host name provided by the client
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
UUID()	Return a Universal Unique Identifier (UUID)
VALUES()	Defines the values to be used during an INSERT
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance
VERSION()	Return a string that indicates the MySQL server version
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
Within()	Whether MBR of one geometry is within MBR of another
X()	Return X coordinate of Point
XOR	Logical XOR
Y()	Return Y coordinate of Point
YEAR()	Return the year
YEARWEEK()	Return the year and week

12.2 Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

It is also possible to convert a number to a string explicitly using the CAST() function. Conversion occurs implicitly with the CONCAT() function because it expects string arguments.

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

See later in this section for information about the character set of implicit number-to-string conversions.

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`, except for the `NULL`-safe `<=>` equality comparison operator. For `NULL <=> NULL`, the result is true. No conversion is needed.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for the arguments to `IN()`! To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using `BETWEEN` with date or time values, use `CAST()` to explicitly convert the values to the desired data type.
- If one of the arguments is a decimal value, comparison depends on the other argument. The arguments are compared as decimal values if the other argument is a decimal or integer value, or as floating-point values if the other argument is a floating-point value.
- In all other cases, the arguments are compared as floating-point (real) numbers.

For information about conversion of values from one temporal type to another, see Section 11.3.7, “Conversion Between Date and Time Types”.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

For comparisons of a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *`str_col`* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value `1`, such as `'1'`, `' 1'`, or `'1a'`.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
        -> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
        -> 0
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
        -> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use `CAST()` so that a value will not be converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
        -> 1
```

For more information about floating-point comparisons, see Section B.5.4.8, “Problems with Floating-Point Values”.

Implicit conversion of a numeric or temporal value to a string produces a binary string (a `BINARY`, `VARBINARY`, or `BLOB` value). Such implicit conversions to string typically occur for functions that are passed numeric or temporal values when string values are more usual, and thus can have effects beyond the type of the converted value. Consider the expression `CONCAT(1, 'abc')`. The numeric argument `1` is converted to the binary string `'1'` and the concatenation of that value with the nonbinary string `'abc'` produces the binary string `'1abc'`.

12.3 Operators

Table 12.2 Operators

Name	Description
`AND`, `&&`	Logical AND
`=`	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`:=`	Assign a value
`BETWEEN ... AND ...`	Check whether a value is within a range of values
`BINARY`	Cast a string to a binary string
`&`	Bitwise AND
`~`	Bitwise inversion
`|`	Bitwise OR
`^`	Bitwise XOR
`CASE`	Case operator
`DIV`	Integer division
`/`	Division operator
`=`	Equal operator
`<=>`	NULL-safe equal to operator

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
>	Greater than operator
>=	Greater than or equal operator
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
<<	Left shift
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
-	Minus operator
%, MOD	Modulo operator
NOT, !	Negates value
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
||, OR	Logical OR
+	Addition operator
REGEXP	Pattern matching using regular expressions
>>	Right shift
RLIKE	Synonym for REGEXP
SOUNDS LIKE	Compare sounds
*	Multiplication operator
-	Change the sign of the argument
XOR	Logical XOR

12.3.1 Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
AND, &&
XOR
OR, ||
= (assignment), :=
```

The precedence of = depends on whether it is used as a comparison operator (=) or as an assignment operator (=). When used as a comparison operator, it has the same precedence as <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, and IN. When used as an assignment operator, it has the same precedence as :=. Section 13.7.4, “SET Syntax”, and Section 9.4, “User-Defined Variables”, explain how MySQL determines which interpretation of = should apply.

For operators that occur at the same precedence level within an expression, evaluation proceeds left to right, with the exception that assignments evaluate right to left.

The meaning of some operators depends on the SQL mode:

- By default, || is a logical OR operator. With PIPES_AS_CONCAT enabled, || is string concatenation, with a precedence between ^ and the unary operators.
- By default, ! has a higher precedence than NOT as of MySQL 5.0.2. For earlier versions, or from 5.0.2 on with HIGH_NOT_PRECEDENCE enabled, ! and NOT have the same precedence.

See Section 5.1.7, “Server SQL Modes”.

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;
        -> 7
mysql> SELECT (1+2)*3;
        -> 9
```

12.3.2 Comparison Functions and Operators

Table 12.3 Comparison Operators

Name	Description
BETWEEN ... AND ...	Check whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
=	Equal operator
<=>	NULL-safe equal to operator
>	Greater than operator
>=	Greater than or equal operator
GREATEST()	Return the largest argument
IN()	Check whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
`ISNULL()`	Test whether the argument is NULL
`LEAST()`	Return the smallest argument
`<`	Less than operator
`<=`	Less than or equal operator
`LIKE`	Simple pattern matching
`NOT BETWEEN ... AND ...`	Check whether a value is not within a range of values
`!=, <>`	Not equal operator
`NOT IN()`	Check whether a value is not within a set of values
`NOT LIKE`	Negation of simple pattern matching
`STRCMP()`	Compare two strings

Comparison operations result in a value of `1` (`TRUE`), `0` (`FALSE`), or `NULL`. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

```
=  >  <  >=  <=  <>  !=
```

The descriptions for those operators later in this section detail how they work with row operands. For additional examples of row comparisons in the context of row subqueries, see Section 13.2.9.5, “Row Subqueries”.

Some of the functions in this section (such as `LEAST()` and `GREATEST()`) return values other than `1` (`TRUE`), `0` (`FALSE`), or `NULL`. However, the value they return is based on comparison operations performed according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”.

To convert a value to a specific type for comparison purposes, you can use the `CAST()` function. String values can be converted to a different character set using `CONVERT()`. See Section 12.10, “Cast Functions and Operators”.

By default, string comparisons are not case sensitive and use the current character set. The default is `latin1` (cp1252 West European), which also works well for English.

- `=`

  Equal:

```
mysql> SELECT 1 = 0;
        -> 0
mysql> SELECT '0' = 0;
        -> 1
mysql> SELECT '0.0' = 0;
        -> 1
mysql> SELECT '0.01' = 0;
        -> 0
mysql> SELECT '.01' = 0.01;
        -> 1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For row comparisons, `(a, b) = (x, y)` is equivalent to:

```
(a = x) AND (b = y)
```

- `<=>`

  `NULL`-safe equal. This operator performs an equality comparison like the `=` operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

  The `<=>` operator is equivalent to the standard SQL `IS NOT DISTINCT FROM` operator.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
        -> 1, NULL, NULL
```

  For row comparisons, `(a, b) <=> (x, y)` is equivalent to:

```
(a <=> x) AND (b <=> y)
```

- `<>`, `!=`

  Not equal:

```
mysql> SELECT '.01' <> '0.01';
        -> 1
mysql> SELECT .01 <> '0.01';
        -> 0
mysql> SELECT 'zapp' <> 'zappp';
        -> 1
```

  For row comparisons, `(a, b) <> (x, y)` and `(a, b) != (x, y)` are equivalent to:

```
(a <> x) OR (b <> y)
```

- `<=`

  Less than or equal:

```
mysql> SELECT 0.1 <= 2;
        -> 1
```

  For row comparisons, `(a, b) <= (x, y)` is equivalent to:

```
(a < x) OR ((a = x) AND (b <= y))
```

- `<`

  Less than:

```
mysql> SELECT 2 < 2;
        -> 0
```

  For row comparisons, `(a, b) < (x, y)` is equivalent to:

This documentation is for an older version. If you're

```
(a < x) OR ((a = x) AND (b < y))
```

- `>=`

  Greater than or equal:

```
mysql> SELECT 2 >= 2;
        -> 1
```

  For row comparisons, `(a, b) >= (x, y)` is equivalent to:

```
(a > x) OR ((a = x) AND (b >= y))
```

- `>`

  Greater than:

```
mysql> SELECT 2 > 2;
        -> 0
```

  For row comparisons, `(a, b) > (x, y)` is equivalent to:

```
(a > x) OR ((a = x) AND (b > y))
```

- `IS boolean_value`

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
        -> 1, 1, 1
```

  `IS boolean_value` syntax was added in MySQL 5.0.2.

- `IS NOT boolean_value`

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
        -> 1, 1, 0
```

  `IS NOT boolean_value` syntax was added in MySQL 5.0.2.

- `IS NULL`

  Tests whether a value is `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0, 0, 1
```

  To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL`:

  - If `sql_auto_is_null` variable is set to 1 (the default), then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` function. For details, including the return value after a multiple-row insert, see Section 12.13, “Information Functions”. If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` comparison can be disabled by setting `sql_auto_is_null = 0`. See Section 5.1.4, “Server System Variables”.

- For `DATE` and `DATETIME` columns that are declared as `NOT NULL`, you can find the special date `'0000-00-00'` by using a statement like this:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

This is needed to get some ODBC applications to work because ODBC does not support a `'0000-00-00'` date value.

See Obtaining Auto-Increment Values, and the description for the `FLAG_AUTO_IS_NULL` option at Connector/ODBC Connection Parameters.

- `IS NOT NULL`

Tests whether a value is not `NULL`.

```
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1, 1, 0
```

- *`expr`* `BETWEEN` *`min`* `AND` *`max`*

If *`expr`* is greater than or equal to *`min`* and *`expr`* is less than or equal to *`max`*, `BETWEEN` returns `1`, otherwise it returns `0`. This is equivalent to the expression `(`*`min`* `<=` *`expr`* `AND` *`expr`* `<=` *`max`*`)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”, but applied to all the three arguments.

```
mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;
        -> 1, 0
mysql> SELECT 1 BETWEEN 2 AND 3;
        -> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
        -> 0
```

For best results when using `BETWEEN` with date or time values, use `CAST()` to explicitly convert the values to the desired data type. Examples: If you compare a `DATETIME` to two `DATE` values, convert the `DATE` values to `DATETIME` values. If you use a string constant such as `'2001-1-1'` in a comparison to a `DATE`, cast the string to a `DATE`.

- *`expr`* `NOT BETWEEN` *`min`* `AND` *`max`*

This is the same as `NOT (`*`expr`* `BETWEEN` *`min`* `AND` *`max`*`)`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `COALESCE(value,...)`

  Returns the first non-`NULL` value in the list, or `NULL` if there are no non-`NULL` values.

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

- `GREATEST(value1,value2,...)`

  With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for `LEAST()`.

```
mysql> SELECT GREATEST(2,0);
        -> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> SELECT GREATEST('B','A','C');
        -> 'C'
```

  Before MySQL 5.0.13, `GREATEST()` returns `NULL` only if all arguments are `NULL`. As of 5.0.13, it returns `NULL` if any argument is `NULL`.

- *`expr`* `IN (value,...)`

  Returns `1` if *`expr`* is equal to any of the values in the `IN` list, else returns `0`. If all values are constants, they are evaluated according to the type of *`expr`* and sorted. The search for the item then is done using a binary search. This means `IN` is very quick if the `IN` value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”, but applied to all the arguments.

```
mysql> SELECT 2 IN (0,3,5,7);
        -> 0
mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
        -> 1
```

  `IN` can be used to compare row constructors:

```
mysql> SELECT (3,4) IN ((1,2), (3,4));
        -> 1
mysql> SELECT (3,4) IN ((1,2), (3,5));
        -> 0
```

  You should never mix quoted and unquoted values in an `IN` list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an `IN` expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

  Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

  The number of values in the `IN` list is only limited by the `max_allowed_packet` value.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To comply with the SQL standard, `IN` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

`IN()` syntax can also be used to write certain types of subqueries. See Section 13.2.9.3, “Subqueries with ANY, IN, or SOME”.

- *`expr`* `NOT IN (`*`value`*`,...)`

  This is the same as `NOT (`*`expr`* `IN (`*`value`*`,...))`.

- `ISNULL(`*`expr`*`)`

  If *`expr`* is `NULL`, `ISNULL()` returns `1`, otherwise it returns `0`.

```
mysql> SELECT ISNULL(1+1);
        -> 0
mysql> SELECT ISNULL(1/0);
        -> 1
```

  `ISNULL()` can be used instead of `=` to test whether a value is `NULL`. (Comparing a value to `NULL` using `=` always yields false.)

  The `ISNULL()` function shares some special behaviors with the `IS NULL` comparison operator. See the description of `IS NULL`.

- `INTERVAL(`*`N`*`,`*`N1`*`,`*`N2`*`,`*`N3`*`,...)`

  Returns `0` if *`N`* < *`N1`*, `1` if *`N`* < *`N2`* and so on or `-1` if *`N`* is `NULL`. All arguments are treated as integers. It is required that *`N1`* < *`N2`* < *`N3`* < `...` < *`Nn`* for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
        -> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

- `LEAST(`*`value1`*`,`*`value2`*`,...)`

  With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

  - If the return value is used in an `INTEGER` context or all arguments are integer-valued, they are compared as integers.
  - If the return value is used in a `REAL` context or all arguments are real-valued, they are compared as reals.
  - If the arguments comprise a mix of numbers and strings, they are compared as numbers.
  - If any argument is a nonbinary (character) string, the arguments are compared as nonbinary strings.
  - In all other cases, the arguments are compared as binary strings.

  Before MySQL 5.0.13, `LEAST()` returns `NULL` only if all arguments are `NULL`. As of 5.0.13, it returns `NULL` if any argument is `NULL`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT LEAST(2,0);
        -> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> SELECT LEAST('B','A','C');
        -> 'A'
```

Note that the preceding conversion rules can produce strange results in some borderline cases:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) AS SIGNED);
        -> -9223372036854775808
```

This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

12.3.3 Logical Operators

Table 12.4 Logical Operators

Name	Description
`AND`, `&&`	Logical AND
`NOT`, `!`	Negates value
`||`, `OR`	Logical OR
`XOR`	Logical XOR

In SQL, all logical operators evaluate to `TRUE`, `FALSE`, or `NULL` (`UNKNOWN`). In MySQL, these are implemented as 1 (`TRUE`), 0 (`FALSE`), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any nonzero value for `TRUE`.

MySQL evaluates any nonzero, non-`NULL` value to `TRUE`. For example, the following statements all assess to `TRUE`:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- `NOT`, `!`

  Logical NOT. Evaluates to `1` if the operand is `0`, to `0` if the operand is nonzero, and `NOT NULL` returns `NULL`.

```
mysql> SELECT NOT 10;
        -> 0
mysql> SELECT NOT 0;
        -> 1
mysql> SELECT NOT NULL;
        -> NULL
mysql> SELECT ! (1+1);
        -> 0
mysql> SELECT ! 1+1;
        -> 1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The last example produces `1` because the expression evaluates the same way as `(!1)+1`.

Note that the precedence of the `NOT` operator changed in MySQL 5.0.2. See Section 12.3.1, “Operator Precedence”.

- `AND`, `&&`

  Logical AND. Evaluates to `1` if all operands are nonzero and not `NULL`, to `0` if one or more operands are `0`, otherwise `NULL` is returned.

```
mysql> SELECT 1 AND 1;
        -> 1
mysql> SELECT 1 AND 0;
        -> 0
mysql> SELECT 1 AND NULL;
        -> NULL
mysql> SELECT 0 AND NULL;
        -> 0
mysql> SELECT NULL AND 0;
        -> 0
```

- `OR`, `||`

  Logical OR. When both operands are non-`NULL`, the result is `1` if any operand is nonzero, and `0` otherwise. With a `NULL` operand, the result is `1` if the other operand is nonzero, and `NULL` otherwise. If both operands are `NULL`, the result is `NULL`.

```
mysql> SELECT 1 OR 1;
        -> 1
mysql> SELECT 1 OR 0;
        -> 1
mysql> SELECT 0 OR 0;
        -> 0
mysql> SELECT 0 OR NULL;
        -> NULL
mysql> SELECT 1 OR NULL;
        -> 1
```

- `XOR`

  Logical XOR. Returns `NULL` if either operand is `NULL`. For non-`NULL` operands, evaluates to `1` if an odd number of operands is nonzero, otherwise `0` is returned.

```
mysql> SELECT 1 XOR 1;
        -> 0
mysql> SELECT 1 XOR 0;
        -> 1
mysql> SELECT 1 XOR NULL;
        -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
        -> 1
```

  `a XOR b` is mathematically equal to `(a AND (NOT b)) OR ((NOT a) and b)`.

12.3.4 Assignment Operators

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 12.5 Assignment Operators

Name	Description
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:=	Assign a value

- :=

  Assignment operator. Causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same SET statement. You can perform multiple assignments in the same statement-

  Unlike =, the := operator is never interpreted as a comparison operator. This means you can use := in any valid SQL statement (not just in SET statements) to assign a value to a variable.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
        -> 4
mysql> SELECT @var1;
        -> 4
```

  You can make value assignments using := in other statements besides SELECT, such as UPDATE, as shown here:

```
mysql> SELECT @var1;
        -> 4
mysql> SELECT * FROM t1;
        -> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @var1;
        -> 1
mysql> SELECT * FROM t1;
        -> 2, 3, 5, 7
```

  While it is also possible both to set and to read the value of the same variable in a single SQL statement using the := operator, this is not recommended. Section 9.4, “User-Defined Variables”, explains why you should avoid doing this.

- =

  This operator is used to perform value assignments in two cases, described in the next two paragraphs.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Within a SET statement, = is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a SET statement, = is treated identically to :=.) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same SET statement.

In the SET clause of an UPDATE statement, = also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any WHERE conditions that are part of the UPDATE are met. You can make multiple assignments in the same SET clause of an UPDATE statement.

In any other context, = is treated as a comparison operator.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1
```

For more information, see Section 13.7.4, “SET Syntax”, Section 13.2.10, “UPDATE Syntax”, and Section 13.2.9, “Subquery Syntax”.

12.4 Control Flow Functions

Table 12.6 Flow Control Operators

Name	Description
CASE	Case operator
IF()	If/else construct
IFNULL()	Null if/else construct
NULLIF()	Return NULL if expr1 = expr2

- CASE *value* WHEN [*compare_value*] THEN *result* [WHEN [*compare_value*] THEN *result* ...] [ELSE *result*] END

  CASE WHEN [*condition*] THEN *result* [WHEN [*condition*] THEN *result* ...] [ELSE *result*] END

  The first version returns the *result* where *value*=*compare_value*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after ELSE is returned, or NULL if there is no ELSE part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
    ->     WHEN 2 THEN 'two' ELSE 'more' END;
        -> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
        -> 'true'
mysql> SELECT CASE BINARY 'B'
    ->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> NULL
```

The return type of a `CASE` expression is the compatible aggregated type of all return values, but also depends on the context in which it is used. If used in a string context, the result is returned as a string. If used in a numeric context, the result is returned as a decimal, real, or integer value.

Note

The syntax of the `CASE` *expression* shown here differs slightly from that of the SQL `CASE` *statement* described in Section 13.6.5.1, “CASE Syntax”, for use inside stored programs. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

- `IF(expr1,expr2,expr3)`

  If *`expr1`* is `TRUE` (*`expr1 <> 0`* and *`expr1 <> NULL`*) then `IF()` returns *`expr2`*; otherwise it returns *`expr3`*. `IF()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2,2,3);
        -> 3
mysql> SELECT IF(1<2,'yes','no');
        -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
        -> 'no'
```

  If only one of *`expr2`* or *`expr3`* is explicitly `NULL`, the result type of the `IF()` function is the type of the non-`NULL` expression.

  The default return type of `IF()` (which may matter when it is stored into a temporary table) is calculated as follows.

Expression	Return Value
`expr2` or *`expr3`* returns a string	string
`expr2` or *`expr3`* returns a floating-point value	floating-point
`expr2` or *`expr3`* returns an integer	integer

  If *`expr2`* and *`expr3`* are both strings, the result is case sensitive if either string is case sensitive.

Note

There is also an `IF` *statement*, which differs from the `IF()` *function* described here. See Section 13.6.5.2, “IF Syntax”.

- `IFNULL(expr1,expr2)`

  If *`expr1`* is not `NULL`, `IFNULL()` returns *`expr1`*; otherwise it returns *`expr2`*. `IFNULL()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
        -> 1
mysql> SELECT IFNULL(NULL,10);
        -> 10
mysql> SELECT IFNULL(1/0,10);
        -> 10
mysql> SELECT IFNULL(1/0,'yes');
        -> 'yes'
```

This documentation is for an older version. If you're
documentation is for an older version. If you're

The default result value of `IFNULL(expr1,expr2)` is the more “general” of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. Consider the case of a table based on expressions or where MySQL must internally store a value returned by `IFNULL()` in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| test  | varbinary(4) | NO   |     |         |       |
+-------+-------------+------+-----+---------+-------+
```

In this example, the type of the `test` column is `VARBINARY(4)`.

- `NULLIF(expr1,expr2)`

  Returns `NULL` if *`expr1 = expr2`* is true, otherwise returns *`expr1`*. This is the same as `CASE WHEN` *`expr1 = expr2`* `THEN NULL ELSE` *`expr1`* `END`.

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
        -> 1
```

Note that MySQL evaluates *`expr1`* twice if the arguments are not equal.

12.5 String Functions

Table 12.7 String Operators

Name	Description
`ASCII()`	Return numeric value of left-most character
`BIN()`	Return a string containing binary representation of a number
`BIT_LENGTH()`	Return length of argument in bits
`CHAR()`	Return the character for each integer passed
`CHAR_LENGTH()`	Return number of characters in argument
`CHARACTER_LENGTH()`	Synonym for CHAR_LENGTH()
`CONCAT()`	Return concatenated string
`CONCAT_WS()`	Return concatenate with separator
`ELT()`	Return string at index number
`EXPORT_SET()`	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
`FIELD()`	Return the index (position) of the first argument in the subsequent arguments
`FIND_IN_SET()`	Return the index position of the first argument within the second argument
`FORMAT()`	Return a number formatted to specified number of decimal places

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH	Perform full-text search
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase

String-valued functions return NULL if the length of the result would be greater than the value of the max_allowed_packet system variable. See Section 8.12.2, “Tuning Server Parameters”.

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- ASCII(*str*)

  Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. ASCII() works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

  See also the ORD() function.

- BIN(*N*)

  Returns a string representation of the binary value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,2). Returns NULL if *N* is NULL.

```
mysql> SELECT BIN(12);
        -> '1100'
```

- BIT_LENGTH(*str*)

  Returns the length of the string *str* in bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

- CHAR(*N*,... [USING *charset_name*])

  CHAR() interprets each argument *N* as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As of MySQL 5.0.15, `CHAR()` arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` is equivalent to `CHAR(1,0)`, and `CHAR(256*256)` is equivalent to `CHAR(1,0,0)`:

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(X'65')), CHARSET(CHAR(X'65' USING utf8));
+----------------------+--------------------------------+
| CHARSET(CHAR(X'65')) | CHARSET(CHAR(X'65' USING utf8)) |
+----------------------+--------------------------------+
| binary               | utf8                           |
+----------------------+--------------------------------+
```

If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

Before MySQL 5.0.15, `CHAR()` returns a string in the connection character set and the `USING` clause is unavailable. In addition, each argument is interpreted modulo 256, so `CHAR(256)` and `CHAR(256*256)` both are equivalent to `CHAR(0)`.

- `CHAR_LENGTH(str)`

  Returns the length of the string *`str`*, measured in characters. A multibyte character counts as a single character. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(str)`

  `CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(str1,str2,...)`

  Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

  ```
  SELECT CONCAT(CAST(int_col AS CHAR), char_col);
  ```

  `CONCAT()` returns `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT CONCAT('My', 'S', 'QL');
         -> 'MySQL'
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

For quoted strings, concatenation can be performed by placing the strings next to each other:

```
mysql> SELECT 'My' 'S' 'QL';
        -> 'MySQL'
```

- `CONCAT_WS(separator,str1,str2,...)`

  `CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

```
mysql> SELECT CONCAT_WS(',','First name','Second name','Last Name');
        -> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
        -> 'First name,Last Name'
```

  `CONCAT_WS()` does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `ELT(N,str1,str2,str3,...)`

  `ELT()` returns the *N*th element of the list of strings: *str1* if *N* = 1, *str2* if *N* = 2, and so on. Returns `NULL` if *N* is less than 1 or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

- `EXPORT_SET(bits,on,off[,separator[,number_of_bits]])`

  Returns a string such that for every bit set in the value *bits*, you get an *on* string and for every bit not set in the value, you get an *off* string. Bits in *bits* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *separator* string (the default being the comma character “,”). The number of bits examined is given by *number_of_bits*, which has a default of 64 if not specified. *number_of_bits* is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of −1 is effectively the same as 64.

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
        -> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',',',10);
        -> '0,1,1,0,0,0,0,0,0,0'
```

- `FIELD(str,str1,str2,str3,...)`

  Returns the index (position) of *str* in the *str1*, *str2*, *str3*, `...` list. Returns `0` if *str* is not found.

  If all arguments to `FIELD()` are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If *str* is NULL, the return value is 0 because NULL fails equality comparison with any value. FIELD() is the complement of ELT().

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 0
```

- FIND_IN_SET(*str*,*strlist*)

  Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N* substrings. A string list is a string composed of substrings separated by ",” characters. If the first argument is a constant string and the second is a column of type SET, the FIND_IN_SET() function is optimized to use bit arithmetic. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns NULL if either argument is NULL. This function does not work properly if the first argument contains a comma (",”) character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

- FORMAT(*X*,*D*)

  Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. If *D* is 0, the result has no decimal point or fractional part. *D* should be a constant value.

```
mysql> SELECT FORMAT(12332.123456, 4);
        -> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
        -> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
        -> '12,332'
```

- HEX(*str*), HEX(*N*)

  For a string argument *str*, HEX() returns a hexadecimal string representation of *str* where each byte of each character in *str* is converted to two hexadecimal digits. (Multibyte characters therefore become more than two digits.) The inverse of this operation is performed by the UNHEX() function.

  For a numeric argument *N*, HEX() returns a hexadecimal string representation of the value of *N* treated as a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,16). The inverse of this operation is performed by CONV(HEX(*N*),16,10).

```
mysql> SELECT X'616263', HEX('abc'), UNHEX(HEX('abc'));
        -> 'abc', 616263, 'abc'
mysql> SELECT HEX(255), CONV(HEX(255),16,10);
        -> 'FF', 255
```

- INSERT(*str*,*pos*,*len*,*newstr*)

  Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns NULL if any argument is NULL.

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
        -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
        -> 'QuWhat'
```

This function is multibyte safe.

- `INSTR(str,substr)`

  Returns the position of the first occurrence of substring *`substr`* in string *`str`*. This is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

  This function is multibyte safe, and is case sensitive only if at least one argument is a binary string.

- `LCASE(str)`

  `LCASE()` is a synonym for `LOWER()`.

- `LEFT(str,len)`

  Returns the leftmost *`len`* characters from the string *`str`*, or `NULL` if any argument is `NULL`.

```
mysql> SELECT LEFT('foobarbar', 5);
        -> 'fooba'
```

  This function is multibyte safe.

- `LENGTH(str)`

  Returns the length of the string *`str`*, measured in bytes. A multibyte character counts as multiple bytes. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

```
mysql> SELECT LENGTH('text');
        -> 4
```

Note

The `Length()` OpenGIS spatial function is named `GLength()` in MySQL.

- `LOAD_FILE(file_name)`

  Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` privilege. The file must be readable by all and its size less than `max_allowed_packet` bytes. If the `secure_file_priv` system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

  If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As of MySQL 5.0.19, the `character_set_filesystem` system variable controls interpretation of file names that are given as literal strings.

```
mysql> UPDATE t
           SET blob_col=LOAD_FILE('/tmp/picture')
           WHERE id=1;
```

- `LOCATE(substr,str)`, `LOCATE(substr,str,pos)`

  The first syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*. The second syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*, starting at position *`pos`*. Returns `0` if *`substr`* is not in *`str`*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(str)`

  Returns the string *`str`* with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

  `LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

  This function is multibyte safe.

- `LPAD(str,len,padstr)`

  Returns the string *`str`*, left-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

```
mysql> SELECT LPAD('hi',4,'??');
        -> '??hi'
mysql> SELECT LPAD('hi',1,'??');
        -> 'h'
```

- `LTRIM(str)`

  Returns the string *`str`* with leading space characters removed.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT LTRIM('  barbar');
        -> 'barbar'
```

This function is multibyte safe.

- `MAKE_SET(bits,str1,str2,...)`

  Returns a set value (a string containing substrings separated by “`,`” characters) consisting of the strings that have the corresponding bit in *`bits`* set. *`str1`* corresponds to bit 0, *`str2`* to bit 1, and so on. `NULL` values in *`str1`*, *`str2`*, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
        -> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

- `MID(str,pos,len)`

  `MID(str,pos,len)` is a synonym for `SUBSTRING(str,pos,len)`.

- `OCT(N)`

  Returns a string representation of the octal value of *`N`*, where *`N`* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if *`N`* is `NULL`.

```
mysql> SELECT OCT(12);
        -> '14'
```

- `OCTET_LENGTH(str)`

  `OCTET_LENGTH()` is a synonym for `LENGTH()`.

- `ORD(str)`

  If the leftmost character of the string *`str`* is a multibyte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  If the leftmost character is not a multibyte character, `ORD()` returns the same value as the `ASCII()` function.

```
mysql> SELECT ORD('2');
        -> 50
```

- `POSITION(substr IN str)`

  `POSITION(substr IN str)` is a synonym for `LOCATE(substr,str)`.

- `QUOTE(str)`

This documentation is for an older version. If you're

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash (“\”), single quote (“'”), ASCII NUL, and Control+Z preceded by a backslash. If the argument is NULL, the return value is the word “NULL” without enclosing single quotation marks.

```
mysql> SELECT QUOTE('Don\'t!');
        -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
        -> NULL
```

For comparison, see the quoting rules for literal strings and within the C API in Section 9.1.1, “String Literals”, and Section 20.6.7.53, “mysql_real_escape_string()”.

- `REPEAT(str,count)`

  Returns a string consisting of the string *str* repeated *count* times. If *count* is less than 1, returns an empty string. Returns NULL if *str* or *count* are NULL.

```
mysql> SELECT REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)`

  Returns the string *str* with all occurrences of the string *from_str* replaced by the string *to_str*. `REPLACE()` performs a case-sensitive match when searching for *from_str*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

  This function is multibyte safe.

- `REVERSE(str)`

  Returns the string *str* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

  This function is multibyte safe.

- `RIGHT(str,len)`

  Returns the rightmost *len* characters from the string *str*, or NULL if any argument is NULL.

```
mysql> SELECT RIGHT('foobarbar', 4);
        -> 'rbar'
```

  This function is multibyte safe.

- `RPAD(str,len,padstr)`

  Returns the string *str*, right-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
```


This documentation is for an older version. If you're

```
mysql> SELECT RPAD('hi',5,'?');
        -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

This function is multibyte safe.

- `RTRIM(str)`

  Returns the string `str` with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar   ');
        -> 'barbar'
```

  This function is multibyte safe.

- `SOUNDEX(str)`

  Returns a soundex string from `str`. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All nonalphabetic characters in `str` are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important

When using `SOUNDEX()`, you should be aware of the following limitations:

  - This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
  - This function is not guaranteed to provide consistent results with strings that use multibyte character sets, including `utf-8`.

    We hope to remove these limitations in a future release. See Bug #22638 for more information.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- `expr1 SOUNDS LIKE expr2`

  This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)`.

- `SPACE(N)`

  Returns a string consisting of `N` space characters.

```
mysql> SELECT SPACE(6);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> '      '
```

- `SUBSTR(str,pos)`, `SUBSTR(str FROM pos)`, `SUBSTR(str,pos,len)`, `SUBSTR(str FROM pos FOR len)`

  `SUBSTR()` is a synonym for `SUBSTRING()`.

- `SUBSTRING(str,pos)`, `SUBSTRING(str FROM pos)`, `SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`

  The forms without a *`len`* argument return a substring from string *`str`* starting at position *`pos`*. The forms with a *`len`* argument return a substring *`len`* characters long from string *`str`*, starting at position *`pos`*. The forms that use `FROM` are standard SQL syntax. It is also possible to use a negative value for *`pos`*. In this case, the beginning of the substring is *`pos`* characters from the end of the string, rather than the beginning. A negative value may be used for *`pos`* in any of the forms of this function.

  For all forms of `SUBSTRING()`, the position of the first character in the string from which the substring is to be extracted is reckoned as `1`.

```
mysql> SELECT SUBSTRING('Quadratically',5);
        -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
        -> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
        -> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
        -> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
        -> 'ki'
```

  This function is multibyte safe.

  If *`len`* is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(str,delim,count)`

  Returns the substring from string *`str`* before *`count`* occurrences of the delimiter *`delim`*. If *`count`* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *`count`* is negative, everything to the right of the final delimiter (counting from the right) is returned. `SUBSTRING_INDEX()` performs a case-sensitive match when searching for *`delim`*.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
        -> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
        -> 'mysql.com'
```

  This function is multibyte safe.

- `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)`, `TRIM([remstr FROM] str)`

  Returns the string *`str`* with all *`remstr`* prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. *`remstr`* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM('  bar   ');
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

This function is multibyte safe.

- `UCASE(str)`

  `UCASE()` is a synonym for `UPPER()`.

- `UNHEX(str)`

  For a string argument *`str`*, `UNHEX(str)` interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT X'4D7953514C';
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

  The characters in the argument string must be legal hexadecimal digits: `'0'` .. `'9'`, `'A'` .. `'F'`, `'a'` .. `'f'`. If the argument contains any nonhexadecimal digits, the result is `NULL`:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
+-------------+
```

  A `NULL` result can occur if the argument to `UNHEX()` is a `BINARY` column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example, `'41'` is stored into a `CHAR(3)` column as `'41 '` and retrieved as `'41'` (with the trailing pad space stripped), so `UNHEX()` for the column value returns `'A'`. By contrast `'41'` is stored into a `BINARY(3)` column as `'41\0'` and retrieved as `'41\0'` (with the trailing pad `0x00` byte not stripped). `'\0'` is not a legal hexadecimal digit, so `UNHEX()` for the column value returns `NULL`.

  For a numeric argument *`N`*, the inverse of `HEX(N)` is not performed by `UNHEX()`. Use `CONV(HEX(N),16,10)` instead. See the description of `HEX()`.

- `UPPER(str)`

  Returns the string *`str`* with all characters changed to uppercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

See the description of `LOWER()` for information that also applies to `UPPER()`, such as information about how to perform lettercase conversion of binary strings (`BINARY`, `VARBINARY`, `BLOB`) for which these functions are ineffective.

This function is multibyte safe.

12.5.1 String Comparison Functions

Table 12.8 String Comparison Operators

Name	Description
`LIKE`	Simple pattern matching
`NOT LIKE`	Negation of simple pattern matching
`STRCMP()`	Compare two strings

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- *`expr`* `LIKE` *`pat`* `[ESCAPE '`*`escape_char`*`']`

  Pattern matching using an SQL pattern. Returns `1` (`TRUE`) or `0` (`FALSE`). If either *`expr`* or *`pat`* is `NULL`, the result is `NULL`.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Per the SQL standard, `LIKE` performs matching on a per-character basis, thus it can produce results different from the `=` comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----------------------------------------+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-------------------------------------+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

  In particular, trailing spaces are significant, which is not true for `CHAR` or `VARCHAR` comparisons performed with the `=` operator:

```
mysql> SELECT 'a' = 'a ', 'a' LIKE 'a ';
+------------+--------------+
| 'a' = 'a ' | 'a' LIKE 'a ' |
+------------+--------------+
|          1 |            0 |
+------------+--------------+
1 row in set (0.00 sec)
```

  With `LIKE` you can use the following two wildcard characters in the pattern:

This documentation is for an older version. If you're

- `%` matches any number of characters, even zero characters.
- `_` matches exactly one character.

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the `ESCAPE` character, "`\`" is assumed.

- `\%` matches one "`%`" character.
- `\_` matches one "`_`" character.

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

The escape sequence should be empty or one character long. The expression must evaluate as a constant at execution time. As of MySQL 5.0.16, if the `NO_BACKSLASH_ESCAPES` SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a case sensitive (uses a case-sensitive collation or is a binary string):

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE _latin1 'ABC' COLLATE latin1_general_cs;
        -> 0
mysql> SELECT 'abc' LIKE _latin1 'ABC' COLLATE latin1_bin;
        -> 0
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

As an extension to standard SQL, MySQL permits `LIKE` on numeric expressions.

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Note

Because MySQL uses C escape syntax in strings (for example, "`\n`" to represent a newline character), you must double any "`\`" that you use in `LIKE` strings. For example, to search for "`\n`", specify it as "`\\n`". To search for "`\`", specify it as "`\\\\`"; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Exception: At the end of the pattern string, backslash can be specified as "\\". At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

```
mysql> SELECT filename FROM t1;
+--------------+
| filename     |
+--------------+
| C:           |
| C:\          |
| C:\Programs  |
| C:\Programs\ |
+--------------+
```

To test for values that end with backslash, you can match the values using either of the following patterns:

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+--------------+---------------------+
| filename     | filename LIKE '%\\' |
+--------------+---------------------+
| C:           |                   0 |
| C:\          |                   1 |
| C:\Programs  |                   0 |
| C:\Programs\ |                   1 |
+--------------+---------------------+

mysql> SELECT filename, filename LIKE '%\\\\' FROM t1;
+--------------+----------------------+
| filename     | filename LIKE '%\\\\' |
+--------------+----------------------+
| C:           |                    0 |
| C:\          |                    1 |
| C:\Programs  |                    0 |
| C:\Programs\ |                    1 |
+--------------+----------------------+
```

- *`expr`* `NOT LIKE` *`pat`* `[ESCAPE '`*`escape_char`*`']`

  This is the same as `NOT (`*`expr`* `LIKE` *`pat`* `[ESCAPE '`*`escape_char`*`'])`.

Note

Aggregate queries involving `NOT LIKE` comparisons with columns containing `NULL` may yield unexpected results. For example, consider the following table and data:

```
CREATE TABLE foo (bar VARCHAR(10));

INSERT INTO foo VALUES (NULL), (NULL);
```

The query `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` returns `0`. You might assume that `SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%';` would return `2`. However, this is not the case: The second query returns `0`. This is because `NULL NOT LIKE` *`expr`* always returns `NULL`, regardless of the value of *`expr`*. The same is true for aggregate queries involving `NULL` and comparisons using `NOT RLIKE` or `NOT REGEXP`. In such cases, you must test explicitly for `NOT NULL` using `OR` (and not `AND`), as shown here:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `STRCMP(expr1,expr2)`

  `STRCMP()` returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
        -> -1
mysql> SELECT STRCMP('text2', 'text');
        -> 1
mysql> SELECT STRCMP('text', 'text');
        -> 0
```

  `STRCMP()` performs the comparison using the collation of the arguments.

```
mysql> SET @s1 = _latin1 'x' COLLATE latin1_general_ci;
mysql> SET @s2 = _latin1 'X' COLLATE latin1_general_ci;
mysql> SET @s3 = _latin1 'x' COLLATE latin1_general_cs;
mysql> SET @s4 = _latin1 'X' COLLATE latin1_general_cs;
mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
+------------------+------------------+
| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
+------------------+------------------+
|                0 |                1 |
+------------------+------------------+
```

  If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See Section 10.1.7.5, “Collation of Expressions”.

```
mysql> SELECT STRCMP(@s1, @s3);
ERROR 1267 (HY000): Illegal mix of collations (latin1_general_ci,IMPLICIT)
and (latin1_general_cs,IMPLICIT) for operation 'strcmp'
mysql> SELECT STRCMP(@s1, @s3 COLLATE latin1_general_ci);
+-------------------------------------------+
| STRCMP(@s1, @s3 COLLATE latin1_general_ci) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
```

12.5.2 Regular Expressions

Table 12.9 String Regular Expression Operators

Name	Description
`NOT REGEXP`	Negation of REGEXP
`REGEXP`	Pattern matching using regular expressions
`RLIKE`	Synonym for REGEXP

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. MySQL uses the extended version to support pattern-matching operations performed with the `REGEXP` operator in SQL statements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This section summarizes, with examples, the special characters and constructs that can be used in MySQL for `REGEXP` operations. It does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory. See also Section 3.3.4.7, “Pattern Matching”.

Regular Expression Operators

- *`expr`* `NOT REGEXP` *`pat`*, *`expr`* `NOT RLIKE` *`pat`*

  This is the same as `NOT (`*`expr`* `REGEXP` *`pat`*`)`.

- *`expr`* `REGEXP` *`pat`*, *`expr`* `RLIKE` *`pat`*

  Performs a pattern match of a string expression *`expr`* against a pattern *`pat`*. The pattern can be an extended regular expression, the syntax for which is discussed later in this section. Returns `1` if *`expr`* matches *`pat`*; otherwise it returns `0`. If either *`expr`* or *`pat`* is `NULL`, the result is `NULL`. `RLIKE` is a synonym for `REGEXP`, provided for `mSQL` compatibility.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Note

  Because MySQL uses the C escape syntax in strings (for example, “`\n`” to represent the newline character), you must double any “`\`” that you use in your `REGEXP` strings.

  `REGEXP` is not case sensitive, except when used with binary strings.

```
mysql> SELECT 'Monty!' REGEXP '.*';
        -> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
        -> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
        -> 1  0
mysql> SELECT 'a' REGEXP '^[a-d]';
        -> 1
```

  `REGEXP` and `RLIKE` use the character set and collations of the arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in Section 10.1.7.5, “Collation of Expressions”.

  Warning

  The `REGEXP` and `RLIKE` operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

Syntax of Regular Expressions

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|word` matches either the string `hello` or the string `word`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the `REGEXP` operator may use any of the following special characters and constructs:

- `^`

  Match the beginning of a string.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';                -> 0
mysql> SELECT 'fofo' REGEXP '^fo';                   -> 1
```

- `$`

  Match the end of a string.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';              -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';                 -> 0
```

- `.`

  Match any character (including carriage return and newline).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';                 -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';             -> 1
```

- `a*`

  Match any sequence of zero or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';                  -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';                -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';                   -> 1
```

- `a+`

  Match any sequence of one or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';                  -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';                   -> 0
```

- `a?`

  Match either zero or one `a` character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';                   -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';                  -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';                 -> 0
```

- `de|abc`

  Match either of the sequences `de` or `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';                  -> 1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT 'axe' REGEXP 'pi|apa';                    -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';                    -> 1
mysql> SELECT 'apa' REGEXP '^(pi|apa)$';                -> 1
mysql> SELECT 'pi' REGEXP '^(pi|apa)$';                 -> 1
mysql> SELECT 'pix' REGEXP '^(pi|apa)$';                -> 0
```

- `(abc)*`

  Match zero or more instances of the sequence `abc`.

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';                    -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';                   -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';                  -> 1
```

- `{1}`, `{2,3}`

  `{n}` or `{m,n}` notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. `m` and `n` are integers.

  - `a*`

    Can be written as `a{0,}`.

  - `a+`

    Can be written as `a{1,}`.

  - `a?`

    Can be written as `a{0,1}`.

  To be more precise, `a{n}` matches exactly `n` instances of `a`. `a{n,}` matches `n` or more instances of `a`. `a{m,n}` matches `m` through `n` instances of `a`, inclusive.

  `m` and `n` must be in the range from `0` to `RE_DUP_MAX` (default 255), inclusive. If both `m` and `n` are given, `m` must be less than or equal to `n`.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';              -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';              -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';           -> 1
```

- `[a-dX]`, `[^a-dX]`

  Matches any character that is (or is not, if ^ is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';                 -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';               -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';              -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';             -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';            -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';           -> 0
```

- `[.characters.]`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Within a bracket expression (written using `[` and `]`), matches the sequence of characters of that collating element. `characters` is either a single character or a character name like `newline`. The following table lists the permissible character names.

The following table shows the permissible character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

Name	Character	Name	Character
NUL	0	SOH	001
STX	002	ETX	003
EOT	004	ENQ	005
ACK	006	BEL	007
alert	007	BS	010
backspace	'\b'	HT	011
tab	'\t'	LF	012
newline	'\n'	VT	013
vertical-tab	'\v'	FF	014
form-feed	'\f'	CR	015
carriage-return	'\r'	SO	016
SI	017	DLE	020
DC1	021	DC2	022
DC3	023	DC4	024
NAK	025	SYN	026
ETB	027	CAN	030
EM	031	SUB	032
ESC	033	IS4	034
FS	034	IS3	035
GS	035	IS2	036
RS	036	IS1	037
US	037	space	' '
exclamation-mark	'!'	quotation-mark	'"'
number-sign	'#'	dollar-sign	'$'
percent-sign	'%'	ampersand	'&'
apostrophe	'\''	left-parenthesis	'('
right-parenthesis	')'	asterisk	'*'
plus-sign	'+'	comma	','
hyphen	'-'	hyphen-minus	'-'
period	'.'	full-stop	'.'
slash	'/'	solidus	'/'
zero	'0'	one	'1'

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Character	Name	Character
two	'2'	three	'3'
four	'4'	five	'5'
six	'6'	seven	'7'
eight	'8'	nine	'9'
colon	':'	semicolon	';'
less-than-sign	'<'	equals-sign	'='
greater-than-sign	'>'	question-mark	'?'
commercial-at	'@'	left-square-bracket	'['
backslash	'\\'	reverse-solidus	'\\'
right-square-bracket	']'	circumflex	'^'
circumflex-accent	'^'	underscore	'_'
low-line	'_'	grave-accent	'`'
left-brace	'{'	left-curly-bracket	'{'
vertical-line	'|'	right-brace	'}'
right-curly-bracket	'}'	tilde	'~'
DEL	177		

```
mysql> SELECT '~' REGEXP '[[.~.]]';                    -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';                 -> 1
```

- `[=character_class=]`

  Within a bracket expression (written using `[` and `]`), `[=character_class=]` represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

  Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
alnum	Alphanumeric characters
alpha	Alphabetic characters
blank	Whitespace characters
cntrl	Control characters
digit	Digit characters

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Character Class Name	Meaning
`graph`	Graphic characters
`lower`	Lowercase alphabetic characters
`print`	Graphic or space characters
`punct`	Punctuation characters
`space`	Space, tab, newline, and carriage return
`upper`	Uppercase alphabetic characters
`xdigit`	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1
mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- `[[:<:]]`, `[[:>:]]`

  These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (_).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

12.6 Numeric Functions and Operators

Table 12.10 Numeric Functions and Operators

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ASIN()`	Return the arc sine
`ATAN()`	Return the arc tangent
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`CEIL()`	Return the smallest integer value not less than the argument
`CEILING()`	Return the smallest integer value not less than the argument
`CONV()`	Convert numbers between different number bases
`COS()`	Return the cosine
`COT()`	Return the cotangent
`CRC32()`	Compute a cyclic redundancy check value

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
DEGREES()	Convert radians to degrees
DIV	Integer division
/	Division operator
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
-	Minus operator
MOD()	Return the remainder
%, MOD	Modulo operator
PI()	Return the value of pi
+	Addition operator
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
*	Multiplication operator
TRUNCATE()	Truncate to specified number of decimal places
-	Change the sign of the argument

12.6.1 Arithmetic Operators

Table 12.11 Arithmetic Operators

Name	Description
DIV	Integer division
/	Division operator
-	Minus operator
%, MOD	Modulo operator
+	Addition operator
*	Multiplication operator
-	Change the sign of the argument

The usual arithmetic operators are available. The result is determined according to the following rules:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- In the case of `-`, `+`, and `*`, the result is calculated with `BIGINT` (64-bit) precision if both operands are integers.
- If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the result is signed even if any operand is unsigned.
- If any of the operands of a `+`, `-`, `/`, `*`, `%` is a real or string value, the precision of the result is the precision of the operand with the maximum precision.
- In division performed with `/`, the scale of the result when using two exact-value operands is the scale of the first operand plus the value of the `div_precision_increment` system variable (which is 4 by default). For example, the result of the expression `5.05 / 0.014` has a scale of six decimal places (`360.714286`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, resolves first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.60288653`).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See Section 12.10, “Cast Functions and Operators”.

For information about handling of overflow in numeric expression evaluation, see Section 11.2.6, “Out-of-Range and Overflow Handling”.

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use `DATE_ADD()`; see Section 12.7, “Date and Time Functions”.

- `+`

  Addition:

```
mysql> SELECT 3+5;
        -> 8
```

- `-`

  Subtraction:

```
mysql> SELECT 3-5;
        -> -2
```

- `-`

  Unary minus. This operator changes the sign of the operand.

```
mysql> SELECT - 2;
        -> -2
```

Note

If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63}.

- `*`

This documentation is for an older version. If you're

Multiplication:

```
mysql> SELECT 3*5;
        -> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
        -> 324518553658426726783156020576256.0
```

- `/`

  Division:

  ```
  mysql> SELECT 3/5;
          -> 0.60
  ```

  Division by zero produces a `NULL` result:

  ```
  mysql> SELECT 102/(1-1);
          -> NULL
  ```

  A division is calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer.

- `DIV`

  Integer division. Discards from the division result any fractional part to the right of the decimal point. Incorrect results may occur for noninteger operands that exceed `BIGINT` range.

  ```
  mysql> SELECT 5 DIV 2, -5 DIV 2, 5 DIV -2, -5 DIV -2;
          -> 2, -2, -2, 2
  ```

- *`N`* `%` *`M`*, *`N`* `MOD` *`M`*

  Modulo operation. Returns the remainder of *`N`* divided by *`M`*. For more information, see the description for the `MOD()` function in Section 12.6.2, “Mathematical Functions”.

12.6.2 Mathematical Functions

Table 12.12 Mathematical Functions

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ASIN()`	Return the arc sine
`ATAN()`	Return the arc tangent
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`CEIL()`	Return the smallest integer value not less than the argument
`CEILING()`	Return the smallest integer value not less than the argument
`CONV()`	Convert numbers between different number bases
`COS()`	Return the cosine
`COT()`	Return the cotangent

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
MOD()	Return the remainder
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

All mathematical functions return NULL in the event of an error.

- ABS(*X*)

  Returns the absolute value of *X*.

```
mysql> SELECT ABS(2);
        -> 2
mysql> SELECT ABS(-32);
        -> 32
```

  This function is safe to use with BIGINT values.

- ACOS(*X*)

  Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns NULL if *X* is not in the range -1 to 1.

```
mysql> SELECT ACOS(1);
        -> 0
mysql> SELECT ACOS(1.0001);
        -> NULL
mysql> SELECT ACOS(0);
        -> 1.5707963267949
```

- ASIN(*X*)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns `NULL` if *X* is not in the range `-1` to `1`.

```
mysql> SELECT ASIN(0.2);
        -> 0.20135792079033
mysql> SELECT ASIN('foo');

+-------------+
| ASIN('foo') |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+---------+------+----------------------------------------+
```

- `ATAN(X)`

  Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

  ```
  mysql> SELECT ATAN(2);
          -> 1.1071487177941
  mysql> SELECT ATAN(-2);
          -> -1.1071487177941
  ```

- `ATAN(Y,X)`, `ATAN2(Y,X)`

  Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of `Y / X`, except that the signs of both arguments are used to determine the quadrant of the result.

  ```
  mysql> SELECT ATAN(-2,2);
          -> -0.78539816339745
  mysql> SELECT ATAN2(PI(),0);
          -> 1.5707963267949
  ```

- `CEIL(X)`

  `CEIL()` is a synonym for `CEILING()`.

- `CEILING(X)`

  Returns the smallest integer value not less than *X*.

  ```
  mysql> SELECT CEILING(1.23);
          -> 2
  mysql> SELECT CEILING(-1.23);
          -> -1
  ```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `CONV(N,from_base,to_base)`

  Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *from_base* to base *to_base*. Returns `NULL` if any argument is `NULL`. The

This documentation is for an older version. If you're
This documentation is for an older version. If you're

argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If *from_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. CONV() works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
        -> '1010'
mysql> SELECT CONV('6E',18,8);
        -> '172'
mysql> SELECT CONV(-17,10,-18);
        -> '-H'
mysql> SELECT CONV(10+'10'+'10'+X'0a',10,10);
        -> '40'
```

- COS(*X*)

  Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
        -> -1
```

- COT(*X*)

  Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
        -> -1.5726734063977
mysql> SELECT COT(0);
        -> NULL
```

- CRC32(*expr*)

  Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is NULL if the argument is NULL. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
        -> 3259397556
mysql> SELECT CRC32('mysql');
        -> 2501908538
```

- DEGREES(*X*)

  Returns the argument *X*, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
        -> 180
mysql> SELECT DEGREES(PI() / 2);
        -> 90
```

- EXP(*X*)

  Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is LOG() (using a single argument only) or LN().

```
mysql> SELECT EXP(2);
        -> 7.3890560989307
mysql> SELECT EXP(-2);
        -> 0.13533528323661
mysql> SELECT EXP(0);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> 1
```

- `FLOOR(X)`

  Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23), FLOOR(-1.23);
        -> 1, -2
```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `FORMAT(X,D)`

  Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions”.

- `HEX(N_or_S)`

  This function can be used to obtain a hexadecimal representation of a decimal number or a string; the manner in which it does so varies according to the argument's type. See this function's description in Section 12.5, “String Functions”, for details.

- `LN(X)`

  Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

  This function is synonymous with `LOG(X)`. The inverse of this function is the `EXP()` function.

- `LOG(X)`, `LOG(B,X)`

  If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

  The inverse of this function (when called with a single argument) is the `EXP()` function.

```
mysql> SELECT LOG(2);
        -> 0.69314718055995
mysql> SELECT LOG(-2);
        -> NULL
```

  If called with two parameters, this function returns the logarithm of *X* to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then `NULL` is returned.

```
mysql> SELECT LOG(2,65536);
        -> 16
mysql> SELECT LOG(10,100);
        -> 2
mysql> SELECT LOG(1,100);
        -> NULL
```

  `LOG(B,X)` is equivalent to `LOG(X) / LOG(B)`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `LOG2(X)`

  Returns the base-2 logarithm of `X`.

```
mysql> SELECT LOG2(65536);
        -> 16
mysql> SELECT LOG2(-100);
        -> NULL
```

  `LOG2()` is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression `LOG(X) / LOG(2)`.

- `LOG10(X)`

  Returns the base-10 logarithm of `X`.

```
mysql> SELECT LOG10(2);
        -> 0.30102999566398
mysql> SELECT LOG10(100);
        -> 2
mysql> SELECT LOG10(-100);
        -> NULL
```

  `LOG10(X)` is equivalent to `LOG(10,X)`.

- `MOD(N,M)`, `N % M`, `N MOD M`

  Modulo operation. Returns the remainder of `N` divided by `M`.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
mysql> SELECT 29 MOD 9;
        -> 2
```

  This function is safe to use with `BIGINT` values.

  `MOD()` also works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

  `MOD(N,0)` returns `NULL`.

- `PI()`

  Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.00000000000000000;
        -> 3.141592653589793116
```

- `POW(X,Y)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Returns the value of *X* raised to the power of *Y*.

```
mysql> SELECT POW(2,2);
        -> 4
mysql> SELECT POW(2,-2);
        -> 0.25
```

- `POWER(X,Y)`

  This is a synonym for `POW()`.

- `RADIANS(X)`

  Returns the argument *X*, converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);
        -> 1.5707963267949
```

- `RAND()`, `RAND(N)`

  Returns a random floating-point value *v* in the range `0` <= *v* < `1.0`. If a constant integer argument *N* is specified, it is used as the seed value, which produces a repeatable sequence of column values. In the following example, note that the sequences of values produced by `RAND(3)` is the same both places where it occurs.

```
mysql> CREATE TABLE t (i INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.61914388706828 |
|    2 | 0.93845168309142 |
|    3 | 0.83482678498591 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.35877890638893 |
|    2 | 0.28941420772058 |
|    3 | 0.37073435016976 |
+------+------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.01 sec)
```

The effect of using a nonconstant argument is undefined. As of MySQL 5.0.13, nonconstant arguments are not permitted.

To obtain a random integer *R* in the range *i* <= *R* < *j*, use the expression `FLOOR(i + RAND() * (j - i))`. For example, to obtain a random integer in the range the range `7` <= *R* < `12`, you could use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

`RAND()` in a `WHERE` clause is re-evaluated every time the `WHERE` is executed.

Use of a column with `RAND()` values in an `ORDER BY` or `GROUP BY` clause may yield unexpected results because for either clause a `RAND()` expression can be evaluated multiple times for the same row, each time returning a different result. However, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

`ORDER BY RAND()` combined with `LIMIT` is useful for selecting a random sample from a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000;
```

`RAND()` is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

- `ROUND(X)`, `ROUND(X,D)`

  Rounds the argument *X* to *D* decimal places. The rounding algorithm depends on the data type of *X*. *D* defaults to 0 if not specified. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT ROUND(-1.23);
        -> -1
mysql> SELECT ROUND(-1.58);
        -> -2
mysql> SELECT ROUND(1.58);
        -> 2
mysql> SELECT ROUND(1.298, 1);
        -> 1.3
mysql> SELECT ROUND(1.298, 0);
        -> 1
mysql> SELECT ROUND(23.298, -1);
        -> 20
```

The return type is the same type as that of the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
+------------------+--------------+
| ROUND(150.000,2) | ROUND(150,2) |
+------------------+--------------+
|           150.00 |          150 |
+------------------+--------------+
```

Before MySQL 5.0.3, the behavior of `ROUND()` when the argument is halfway between two integers depends on the C library implementation. Different implementations round to the nearest even number, always up, always down, or always toward zero. If you need one kind of rounding, you should use a well-defined function such as `TRUNCATE()` or `FLOOR()` instead.

As of MySQL 5.0.3, `ROUND()` uses the following rules depending on the type of the first argument:

- For exact-value numbers, `ROUND()` uses the “round half away from zero” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the "round to nearest even" rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For more information, see Section 12.17, “Precision Math”.

- `SIGN(X)`

  Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether *X* is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
```

- `SIN(X)`

  Returns the sine of *X*, where *X* is given in radians.

```
mysql> SELECT SIN(PI());
        -> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
        -> 0
```

- `SQRT(X)`

  Returns the square root of a nonnegative number *X*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT SQRT(4);
        -> 2
mysql> SELECT SQRT(20);
        -> 4.4721359549996
mysql> SELECT SQRT(-16);
        -> NULL
```

- `TAN(X)`

  Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI());
        -> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
        -> 1.5574077246549
```

- `TRUNCATE(X,D)`

  Returns the number *X*, truncated to *D* decimal places. If *D* is `0`, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1028
```

  All numbers are rounded toward zero.

12.7 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 11.3, “Date and Time Types”, for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Table 12.13 Date/Time Functions

Name	Description
`ADDDATE()`	Add time values (intervals) to a date value
`ADDTIME()`	Add time
`CONVERT_TZ()`	Convert from one timezone to another
`CURDATE()`	Return the current date
`CURRENT_DATE()`, `CURRENT_DATE`	Synonyms for CURDATE()
`CURRENT_TIME()`, `CURRENT_TIME`	Synonyms for CURTIME()
`CURRENT_TIMESTAMP()`, `CURRENT_TIMESTAMP`	Synonyms for NOW()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format UNIX timestamp as a date
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
UNIX_TIMESTAMP()	Return a UNIX timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a *date_col* value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as NOW() within a single query always produce the same result. (For our purposes, a single query also includes a call to a stored program (stored routine or trigger) and all subprograms called by that program.) This principle also applies to CURDATE(), CURTIME(), UTC_DATE(), UTC_TIME(), UTC_TIMESTAMP(), and to any of their synonyms.

The CURRENT_TIMESTAMP(), CURRENT_TIME(), CURRENT_DATE(), and FROM_UNIXTIME() functions return values in the connection's current time zone, which is available as the value of the time_zone system variable. In addition, UNIX_TIMESTAMP() assumes that its argument is a datetime value in the current time zone. See Section 10.6, “MySQL Server Time Zone Support”.

Some date functions can be used with “zero” dates or incomplete dates such as '2001-11-00', whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
       -> 0, 0
```

Other functions expect complete dates and return NULL for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
        -> NULL
mysql> SELECT DAYNAME('2006-05-00');
        -> NULL
```

- `ADDDATE(date,INTERVAL expr unit)`, `ADDDATE(expr,days)`

  When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL unit` argument, see the discussion for `DATE_ADD()`.

  ```
  mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  ```

  When invoked with the *days* form of the second argument, MySQL treats it as an integer number of days to be added to *expr*.

  ```
  mysql> SELECT ADDDATE('2008-01-02', 31);
          -> '2008-02-02'
  ```

- `ADDTIME(expr1,expr2)`

  `ADDTIME()` adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression.

  ```
  mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
          -> '2008-01-02 01:01:01.000001'
  mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
          -> '03:00:01.999997'
  ```

- `CONVERT_TZ(dt,from_tz,to_tz)`

  `CONVERT_TZ()` converts a datetime value *dt* from the time zone given by *from_tz* to the time zone given by *to_tz* and returns the resulting value. Time zones are specified as described in Section 10.6, “MySQL Server Time Zone Support”. This function returns `NULL` if the arguments are invalid.

  If the value falls out of the supported range of the `TIMESTAMP` type when converted from *from_tz* to UTC, no conversion occurs. The `TIMESTAMP` range is described in Section 11.1.2, “Date and Time Type Overview”.

  ```
  mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
          -> '2004-01-01 13:00:00'
  mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
          -> '2004-01-01 22:00:00'
  ```

  Note

  To use named time zones such as `'MET'` or `'Europe/Moscow'`, the time zone tables must be properly set up. See Section 10.6, “MySQL Server Time Zone Support”, for instructions.

  If you intend to use `CONVERT_TZ()` while other tables are locked with `LOCK TABLES`, you must also lock the `mysql.time_zone_name` table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CURDATE()`

  Returns the current date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
        -> '2008-06-13'
mysql> SELECT CURDATE() + 0;
        -> 20080613
```

- `CURRENT_DATE`, `CURRENT_DATE()`

  `CURRENT_DATE` and `CURRENT_DATE()` are synonyms for `CURDATE()`.

- `CURRENT_TIME`, `CURRENT_TIME()`

  `CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP()`

  `CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `CURTIME()`

  Returns the current time as a value in `'HH:MM:SS'` or `HHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026.000000
```

- `DATE(`*`expr`*`)`

  Extracts the date part of the date or datetime expression *`expr`*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

- `DATEDIFF(`*`expr1`*`,`*`expr2`*`)`

  `DATEDIFF()` returns *`expr1`* − *`expr2`* expressed as a value in days from one date to the other. *`expr1`* and *`expr2`* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
        -> -31
```

- `DATE_ADD(`*`date`*`,INTERVAL `*`expr unit`*`)`, `DATE_SUB(`*`date`*`,INTERVAL `*`expr unit`*`)`

  These functions perform date arithmetic. The *`date`* argument specifies the starting date or datetime value. *`expr`* is an expression specifying the interval value to be added or subtracted from the starting date. *`expr`* is a string; it may start with a "`-`" for negative intervals. *`unit`* is a keyword indicating the units in which the expression should be interpreted.

  The `INTERVAL` keyword and the *`unit`* specifier are not case sensitive.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following table shows the expected form of the *expr* argument for each *unit* value.

***unit* Value**	**Expected *expr* Format**
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

The return value depends on the arguments:

- DATETIME if the first argument is a DATETIME (or TIMESTAMP) value, or if the first argument is a DATE and the *unit* value uses HOURS, MINUTES, or SECONDS.
- String otherwise.

To ensure that the result is DATETIME, you can use CAST() to convert the first argument to DATETIME.

MySQL permits any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a DATE value and your calculations involve only YEAR, MONTH, and DAY parts (that is, no time parts), the result is a DATE value. Otherwise, the result is a DATETIME value.

Date arithmetic also can be performed using INTERVAL together with the + or - operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`INTERVAL` *`expr unit`* is permitted on either side of the `+` operator if the expression on the other side is a date or datetime value. For the `-` operator, `INTERVAL` *`expr unit`* is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '2009-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2008-12-31';
        -> '2009-01-01'
mysql> SELECT '2005-01-01' - INTERVAL 1 SECOND;
        -> '2004-12-31 23:59:59'
mysql> SELECT DATE_ADD('2000-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2001-01-01 00:00:00'
mysql> SELECT DATE_ADD('2010-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2011-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2005-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2004-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *`unit`* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *`unit`* of `DAY_SECOND`, the value of *`expr`* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

Because *`expr`* is treated as a string, be careful if you specify a nonstring value with `INTERVAL`. For example, with an interval specifier of `HOUR_MINUTE`, `6/4` evaluates to `1.5000` and is treated as 1 hour, 5000 minutes:

```
mysql> SELECT 6/4;
        -> 1.5000
mysql> SELECT DATE_ADD('2009-01-01', INTERVAL 6/4 HOUR_MINUTE);
        -> '2009-01-04 12:20:00'
```

To ensure interpretation of the interval value as you expect, a `CAST()` operation may be used. To treat `6/4` as 1 hour, 5 minutes, cast it to a `DECIMAL` value with a single fractional digit:

```
mysql> SELECT CAST(6/4 AS DECIMAL(3,1));
        -> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
    ->                 INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
        -> '1970-01-01 13:05:00'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 DAY);
        -> '2013-01-02'
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 HOUR);
        -> '2013-01-01 01:00:00'
```

If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('2009-01-30', INTERVAL 1 MONTH);
        -> '2009-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2006-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
        -> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
        -> NULL
```

- `DATE_FORMAT(`*`date`*`,`*`format`*`)`

  Formats the *`date`* value according to the *`format`* string.

  The following specifiers may be used in the *`format`* string. The "`%`" character is required before format specifier characters.

Specifier	Description
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%b`	Abbreviated month name (`Jan`..`Dec`)
`%c`	Month, numeric (`0`..`12`)
`%D`	Day of the month with English suffix (`0th`, `1st`, `2nd`, `3rd`, …)
`%d`	Day of the month, numeric (`00`..`31`)
`%e`	Day of the month, numeric (`0`..`31`)
`%f`	Microseconds (`000000`..`999999`)
`%H`	Hour (`00`..`23`)
`%h`	Hour (`01`..`12`)
`%I`	Hour (`01`..`12`)
`%i`	Minutes, numeric (`00`..`59`)
`%j`	Day of year (`001`..`366`)
`%k`	Hour (`0`..`23`)
`%l`	Hour (`1`..`12`)
`%M`	Month name (`January`..`December`)
`%m`	Month, numeric (`00`..`12`)
`%p`	`AM` or `PM`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Specifier	Description
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK() mode 1
%V	Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
%v	Week (01..53), where Monday is the first day of the week; WEEK() mode 3; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal "%" character
%*x*	*x*, for any "*x*" not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as '2014-00-00'.

As of MySQL 5.0.25, the language used for day and month names and abbreviations is controlled by the value of the lc_time_names system variable (Section 10.7, "MySQL Server Locale Support").

For the %U, %u, %V, and %v specifiers, see the description of the WEEK() function for information about the mode values. The mode affects how week numbering occurs.

As of MySQL 5.0.36, DATE_FORMAT() returns a string with a character set and collation given by character_set_connection and collation_connection so that it can return month and weekday names containing non-ASCII characters. Before 5.0.36, the return value is a binary string.

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
        -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
    ->                '%D %y %a %d %m %b %j');
        -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
    ->                '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
        -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
        -> '00'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `DATE_SUB(date,INTERVAL expr unit)`

  See the description for `DATE_ADD()`.

- `DAY(date)`

  `DAY()` is a synonym for `DAYOFMONTH()`.

- `DAYNAME(date)`

  Returns the name of the weekday for *`date`*. As of MySQL 5.0.25, the language used for the name is controlled by the value of the `lc_time_names` system variable (Section 10.7, “MySQL Server Locale Support”).

```
mysql> SELECT DAYNAME('2007-02-03');
        -> 'Saturday'
```

- `DAYOFMONTH(date)`

  Returns the day of the month for *`date`*, in the range `1` to `31`, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero day part.

```
mysql> SELECT DAYOFMONTH('2007-02-03');
        -> 3
```

- `DAYOFWEEK(date)`

  Returns the weekday index for *`date`* (`1` = Sunday, `2` = Monday, …, `7` = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('2007-02-03');
        -> 7
```

- `DAYOFYEAR(date)`

  Returns the day of the year for *`date`*, in the range `1` to `366`.

```
mysql> SELECT DAYOFYEAR('2007-02-03');
        -> 34
```

- `EXTRACT(unit FROM date)`

  The `EXTRACT()` function uses the same kinds of unit specifiers as `DATE_ADD()` or `DATE_SUB()`, but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '2009-07-02');
       -> 2009
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
       -> 200907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
       -> 20102
mysql> SELECT EXTRACT(MICROSECOND
    ->                FROM '2003-01-02 10:30:00.000123');
        -> 123
```

- `FROM_DAYS(N)`

This documentation is for an older version. If you're

Given a day number *N*, returns a DATE value.

```
mysql> SELECT FROM_DAYS(730669);
        -> '2007-07-03'
```

Use FROM_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See Section 12.8, “What Calendar Is Used By MySQL?”.

- FROM_UNIXTIME(*unix_timestamp*), FROM_UNIXTIME(*unix_timestamp*,*format*)

  Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. *unix_timestamp* is an internal timestamp value such as is produced by the UNIX_TIMESTAMP() function.

  If *format* is given, the result is formatted according to the *format* string, which is used the same way as listed in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(1447430881);
        -> '2015-11-13 10:08:01'
mysql> SELECT FROM_UNIXTIME(1447430881) + 0;
        -> 20151113100801
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                      '%Y %D %M %h:%i:%s %x');
        -> '2015 13th November 10:08:01 2015'
```

  Note: If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between TIMESTAMP values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the UNIX_TIMESTAMP() function.

- GET_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})

  Returns a format string. This function is useful in combination with the DATE_FORMAT() and the STR_TO_DATE() functions.

  The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the DATE_FORMAT() function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Function Call	Result
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

TIMESTAMP can also be used as the first argument to GET_FORMAT(), in which case the function returns the same values as for DATETIME.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
        -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
        -> '2003-10-31'
```

- HOUR(*time*)

  Returns the hour for *time*. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
mysql> SELECT HOUR('272:59:59');
        -> 272
```

- LAST_DAY(*date*)

  Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

- LOCALTIME, LOCALTIME()

  LOCALTIME and LOCALTIME() are synonyms for NOW().

- LOCALTIMESTAMP, LOCALTIMESTAMP()

  LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

- MAKEDATE(*year*,*dayofyear*)

  Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is NULL.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
        -> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
        -> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> NULL
```

- `MAKETIME(hour,minute,second)`

  Returns a time value calculated from the *`hour`*, *`minute`*, and *`second`* arguments.

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

- `MICROSECOND(expr)`

  Returns the microseconds from the time or datetime expression *`expr`* as a number in the range from `0` to `999999`.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('2009-12-31 23:59:59.000010');
        -> 10
```

- `MINUTE(time)`

  Returns the minute for *`time`*, in the range `0` to `59`.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

- `MONTH(date)`

  Returns the month for *`date`*, in the range `1` to `12` for January to December, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

- `MONTHNAME(date)`

  Returns the full name of the month for *`date`*. As of MySQL 5.0.25, the language used for the name is controlled by the value of the `lc_time_names` system variable (Section 10.7, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

- `NOW()`

  Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();
        -> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 20071215235026.000000
```

  `NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began

This documentation is for an older version. If you're
This documentation is for an older version. If you're

to execute.) This differs from the behavior for `SYSDATE()`, which returns the exact time at which it executes as of MySQL 5.0.12.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. Setting the timestamp to a nonzero value causes each subsequent invocation of `NOW()` to return that value. Setting the timestamp to zero cancels this effect so that `NOW()` once again returns the current date and time.

See the description for `SYSDATE()` for additional information about the differences between the two functions.

- `PERIOD_ADD(P,N)`

  Adds *`N`* months to period *`P`* (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`. Note that the period argument *`P`* is *not* a date value.

  ```
  mysql> SELECT PERIOD_ADD(200801,2);
         -> 200803
  ```

- `PERIOD_DIFF(P1,P2)`

  Returns the number of months between periods *`P1`* and *`P2`*. *`P1`* and *`P2`* should be in the format `YYMM` or `YYYYMM`. Note that the period arguments *`P1`* and *`P2`* are *not* date values.

  ```
  mysql> SELECT PERIOD_DIFF(200802,200703);
         -> 11
  ```

- `QUARTER(date)`

  Returns the quarter of the year for *`date`*, in the range `1` to `4`.

  ```
  mysql> SELECT QUARTER('2008-04-01');
         -> 2
  ```

- `SECOND(time)`

  Returns the second for *`time`*, in the range `0` to `59`.

  ```
  mysql> SELECT SECOND('10:05:03');
         -> 3
  ```

- `SEC_TO_TIME(seconds)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Returns the *`seconds`* argument, converted to hours, minutes, and seconds, as a `TIME` value. The range of the result is constrained to that of the `TIME` data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

- `STR_TO_DATE(`*`str`*`,`*`format`*`)`

This is the inverse of the `DATE_FORMAT()` function. It takes a string *`str`* and a format string *`format`*. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts. If the date, time, or datetime value extracted from *`str`* is illegal, `STR_TO_DATE()` returns `NULL` and, as of MySQL 5.0.3, produces a warning.

The server scans *`str`* attempting to match *`format`* to it. The format string can contain literal characters and format specifiers beginning with `%`. Literal characters in *`format`* must match literally in *`str`*. Format specifiers in *`format`* must match a date or time part in *`str`*. For the specifiers that can be used in *`format`*, see the `DATE_FORMAT()` function description.

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
        -> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
        -> '2013-05-01'
```

Scanning starts at the beginning of *`str`* and fails if *`format`* is found not to match. Extra characters at the end of *`str`* are ignored.

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
        -> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
        -> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
        -> '09:30:17'
```

Unspecified date or time parts have a value of 0, so incompletely specified values in *`str`* produce a result with some or all parts set to 0:

```
mysql> SELECT STR_TO_DATE('abc','abc');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
        -> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
        -> '00:00:09'
```

Range checking on the parts of date values is as described in Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”. This means, for example, that “zero” dates or dates with part values of 0 are permitted unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

You cannot use format `"%X%V"` to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
        -> '2004-10-18'
```

- `SUBDATE(`*`date`*`,INTERVAL` *`expr unit`*`)`, `SUBDATE(`*`expr`*`,`*`days`*`)`

  When invoked with the `INTERVAL` form of the second argument, `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL` *`unit`* argument, see the discussion for `DATE_ADD()`.

```
mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
```

  The second form enables the use of an integer value for *`days`*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *`expr`*.

```
mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
        -> '2007-12-02 12:00:00'
```

- `SUBTIME(`*`expr1`*`,`*`expr2`*`)`

  `SUBTIME()` returns *`expr1`* − *`expr2`* expressed as a value in the same format as *`expr1`*. *`expr1`* is a time or datetime expression, and *`expr2`* is a time expression.

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
        -> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

- `SYSDATE()`

  Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context.

  As of MySQL 5.0.12, `SYSDATE()` returns the time at which it executes. This differs from the behavior for `NOW()`, which returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`.

Because `SYSDATE()` can return different values even within the same statement, and is not affected by `SET TIMESTAMP`, it is nondeterministic and therefore unsafe for replication. If that is a problem, you can start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. The nondeterministic nature of `SYSDATE()` also means that indexes cannot be used for evaluating expressions that refer to it.

- `TIME(`*`expr`*`)`

  Extracts the time part of the time or datetime expression *`expr`* and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
        -> '01:02:03.000123'
```

- `TIMEDIFF(`*`expr1`*`,`*`expr2`*`)`

  `TIMEDIFF()` returns *`expr1`* − *`expr2`* expressed as a time value. *`expr1`* and *`expr2`* are time or date-and-time expressions, but both must be of the same type.

  The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` and `UNIX_TIMESTAMP()`, both of which return integers.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
    ->                 '2008-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

- `TIMESTAMP(`*`expr`*`)`, `TIMESTAMP(`*`expr1`*`,`*`expr2`*`)`

  With a single argument, this function returns the date or datetime expression *`expr`* as a datetime value. With two arguments, it adds the time expression *`expr2`* to the date or datetime expression *`expr1`* and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(`*`unit`*`,`*`interval`*`,`*`datetime_expr`*`)`

  Adds the integer expression *`interval`* to the date or datetime expression *`datetime_expr`*. The unit for *`interval`* is given by the *`unit`* argument, which should be one of the following values: `FRAC_SECOND` (microseconds), `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

  Beginning with MySQL 5.0.60, it is possible to use `MICROSECOND` in place of `FRAC_SECOND` with this function, and `FRAC_SECOND` is deprecated. `FRAC_SECOND` is removed in MySQL 5.5.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The *unit* value may be specified using one of keywords as shown, or with a prefix of SQL_TSI_. For example, DAY and SQL_TSI_DAY both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

- TIMESTAMPDIFF(*unit*,*datetime_expr1*,*datetime_expr2*)

  Returns *datetime_expr2* − *datetime_expr1*, where *datetime_expr1* and *datetime_expr2* are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part '00:00:00' where necessary. The unit for the result (an integer) is given by the *unit* argument. The legal values for *unit* are the same as those listed in the description of the TIMESTAMPADD() function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
        -> 128885
```

Note

The order of the date or datetime arguments for this function is the opposite of that used with the TIMESTAMP() function when invoked with 2 arguments.

- TIME_FORMAT(*time*,*format*)

  This is used like the DATE_FORMAT() function, but the *format* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a NULL value or 0.

  If the *time* value contains an hour part that is greater than 23, the %H and %k hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
        -> '100 100 04 04 4'
```

- TIME_TO_SEC(*time*)

  Returns the *time* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
        -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
        -> 2378
```

- TO_DAYS(*date*)

  Given a date *date*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
        -> 728779
mysql> SELECT TO_DAYS('2007-10-07');
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> 733321
```

`TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.8, “What Calendar Is Used By MySQL?”, for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time Types”. For example, `'2008-10-07'` and `'08-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
        -> 733687, 733687
```

In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_DAYS()` returns the values shown here:

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----------------------+
| to_days('0000-00-00') |
+-----------------------+
|                  NULL |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+---------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_DAYS('0000-01-01');
+-----------------------+
| to_days('0000-01-01') |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

This is true whether or not the `ALLOW_INVALID_DATES` SQL server mode (available in MySQL 5.0.2 and later) is enabled.

- `UNIX_TIMESTAMP()`, `UNIX_TIMESTAMP(`*`date`*`)`

  If called with no argument, returns a Unix timestamp (seconds since `'1970-01-01 00:00:00'` UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a *`date`* argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. *`date`* may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD`. The server interprets *`date`* as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in Section 10.6, “MySQL Server Time Zone Support”.

  ```
  mysql> SELECT UNIX_TIMESTAMP();
          -> 1447431666
  mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19');
          -> 1447431619
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When `UNIX_TIMESTAMP()` is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()`, it returns `0`.

Note: If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between `TIMESTAMP` values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes, it is possible for two `UNIX_TIMESTAMP()` to map two `TIMESTAMP` values to the same Unix timestamp value. `FROM_UNIXTIME()` will map that value back to only one of the original `TIMESTAMP` values. Here is an example, using `TIMESTAMP` values in the `CET` time zone:

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1111885200);
+---------------------------+
| FROM_UNIXTIME(1111885200) |
+---------------------------+
| 2005-03-27 03:00:00       |
+---------------------------+
```

If you want to subtract `UNIX_TIMESTAMP()` columns, you might want to cast the result to signed integers. See Section 12.10, “Cast Functions and Operators”.

- `UTC_DATE`, `UTC_DATE()`

  Returns the current UTC date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

  ```
  mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
         -> '2003-08-14', 20030814
  ```

- `UTC_TIME`, `UTC_TIME()`

  Returns the current UTC time as a value in `'HH:MM:SS'` or `HHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context.

  ```
  mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
         -> '18:07:53', 180753.000000
  ```

- `UTC_TIMESTAMP`, `UTC_TIMESTAMP()`

  Returns the current UTC date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context.

  ```
  mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        -> '2003-08-14 18:08:04', 20030814180804.000000
```

- `WEEK(date[,mode])`

  This function returns the week number for *`date`*. The two-argument form of `WEEK()` enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from `0` to `53` or from `1` to `53`. If the *`mode`* argument is omitted, the value of the `default_week_format` system variable is used. See Section 5.1.4, “Server System Variables”.

  The following table describes how the *`mode`* argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

  For *`mode`* values with a meaning of “with 4 or more days this year,” weeks are numbered according to ISO 8601:1988:

  - If the week containing January 1 has 4 or more days in the new year, it is week 1.
  - Otherwise, it is the last week of the previous year, and the next week is week 1.

```
mysql> SELECT WEEK('2008-02-20');
        -> 7
mysql> SELECT WEEK('2008-02-20',0);
        -> 7
mysql> SELECT WEEK('2008-02-20',1);
        -> 8
mysql> SELECT WEEK('2008-12-31',1);
        -> 53
```

  Note that if a date falls in the last week of the previous year, MySQL returns `0` if you do not use `2`, `3`, `6`, or `7` as the optional *`mode`* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
        -> 2000, 0
```

  One might argue that `WEEK()` should return `52` because the given date actually occurs in the 52nd week of 1999. `WEEK()` returns `0` instead so that the return value is “the week number in the given year.” This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

  If you prefer a result evaluated with respect to the year that contains the first day of the week for the given date, use `0`, `2`, `5`, or `7` as the optional *`mode`* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(date)`

  Returns the weekday index for *`date`* (`0` = Monday, `1` = Tuesday, … `6` = Sunday).

```
mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
        -> 6
mysql> SELECT WEEKDAY('2007-11-06');
        -> 1
```

- `WEEKOFYEAR(date)`

  Returns the calendar week of the date as a number in the range from `1` to `53`. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)`.

```
mysql> SELECT WEEKOFYEAR('2008-02-20');
        -> 8
```

- `YEAR(date)`

  Returns the year for *`date`*, in the range `1000` to `9999`, or `0` for the “zero” date.

```
mysql> SELECT YEAR('1987-01-01');
        -> 1987
```

- `YEARWEEK(date)`, `YEARWEEK(date,mode)`

  Returns year and week for a date. The year in the result may be different from the year in the date argument for the first and the last week of the year.

  The *`mode`* argument works exactly like the *`mode`* argument to `WEEK()`. For the single-argument syntax, a *`mode`* value of 0 is used. Unlike `WEEK()`, the value of `default_week_format` does not influence `YEARWEEK()`.

```
mysql> SELECT YEARWEEK('1987-01-01');
        -> 198652
```

  Note that the week number is different from what the `WEEK()` function would return (`0`) for optional arguments `0` or `1`, as `WEEK()` then returns the week in the context of the given year.

12.8 What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL `DATE` or `DATETIME` values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

12.9 Full-Text Search Functions

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])

search_modifier: { IN BOOLEAN MODE | WITH QUERY EXPANSION }
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type `FULLTEXT`.
- Full-text indexes can be used only with `MyISAM` tables, and can be created only for `CHAR`, `VARCHAR`, or `TEXT` columns.
- A `FULLTEXT` index definition can be given in the `CREATE TABLE` statement when a table is created, or added later using `ALTER TABLE` or `CREATE INDEX`.
- For large data sets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` syntax. `MATCH()` takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

There are three types of full-text searches:

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Common words such as “some” or “then” are stopwords and do not match if present in the search string. The `IN BOOLEAN MODE` modifier specifies a boolean search. For more information, see Section 12.9.2, “Boolean Full-Text Searches”.
- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies. In addition, words that are

This documentation is for an older version. If you're
This documentation is for an older version. If you're

present in 50% or more of the rows are considered common and do not match. Full-text searches are natural language searches if no modifier is given.

- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The `WITH QUERY EXPANSION` modifier specifies a query expansion search. For more information, see Section 12.9.3, “Full-Text Searches with Query Expansion”.

Constraints on full-text searching are listed in Section 12.9.5, “Full-Text Restrictions”.

The `myisam_ftdump` utility can be used to dump the contents of a full-text index. This may be helpful for debugging full-text queries. See Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”.

12.9.1 Natural Language Full-Text Searches

By default, the `MATCH()` function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a `FULLTEXT` index. The search string is given as the argument to `AGAINST()`. For each row in the table, `MATCH()` returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the `MATCH()` list.

```
mysql> CREATE TABLE articles (
    ->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   title VARCHAR(200),
    ->   body TEXT,
    ->   FULLTEXT (title,body)
    -> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
    -> ('MySQL Tutorial','DBMS stands for DataBase ...'),
    -> ('How To Use MySQL Well','After you went through a ...'),
    -> ('Optimizing MySQL','In this tutorial we will show ...'),
    -> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    -> ('MySQL vs. YourSQL','In the following database comparison ...'),
    -> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)
```

By default, the search is performed in case-insensitive fashion. However, you can perform a case-sensitive full-text search by using a binary collation for the indexed columns. For example, a column that uses the `latin1` character set of can be assigned a collation of `latin1_bin` to make it case sensitive for full-text searches.

When `MATCH()` is used in a `WHERE` clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
    -> WHERE MATCH (title,body)
    -> AGAINST ('database');
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

However, you might find it quicker to rewrite the query as follows:

```
mysql> SELECT
    -> COUNT(IF(MATCH (title,body) AGAINST ('database'), 1, NULL))
    -> AS count
    -> FROM articles;
+-------+
| count |
+-------+
|     2 |
+-------+
1 row in set (0.00 sec)
```

The first query sorts the results by relevance whereas the second does not. However, the second query performs a full table scan and the first does not. The first may be faster if the search matches few rows; otherwise, the second may be faster because it would read many rows anyway.

For natural-language full-text searches, it is a requirement that the columns named in the `MATCH()` function be the same columns included in some `FULLTEXT` index in your table. For the preceding query, note that the columns named in the `MATCH()` function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. If you wanted to search the `title` or `body` separately, you would need to create separate `FULLTEXT` indexes for each column.

It is also possible to perform a boolean search or a search with query expansion. These search types are described in Section 12.9.2, “Boolean Full-Text Searches”, and Section 12.9.3, “Full-Text Searches with Query Expansion”.

A full-text search that uses an index can name columns only from a single table in the `MATCH()` clause because an index cannot span multiple tables. A boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the `MATCH()` function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the `SELECT` statement includes neither `WHERE` nor `ORDER BY` clauses:

```
mysql> SELECT id, MATCH (title,body) AGAINST ('Tutorial')
    -> FROM articles;
+----+----------------------------------------+
| id | MATCH (title,body) AGAINST ('Tutorial') |
+----+----------------------------------------+
|  1 |                      0.65545833110809 |
|  2 |                                     0 |
|  3 |                      0.66266459226608 |
|  4 |                                     0 |
|  5 |                                     0 |
|  6 |                                     0 |
+----+----------------------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, specify `MATCH()` twice: once in the `SELECT` list and once in the `WHERE` clause. This causes no additional overhead, because the MySQL optimizer notices that the two `MATCH()` calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root') AS score
    -> FROM articles WHERE MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root');
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

The MySQL `FULLTEXT` implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (“`'`”), but not more than one in a row. This means that `aaa'bbb` is regarded as one word, but `aaa''bbb` is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the `FULLTEXT` parser; `'aaa'bbb'` would be parsed as `aaa'bbb`.

The `FULLTEXT` parser determines where words start and end by looking for certain delimiter characters; for example, “ ” (space), “`,`” (comma), and “`.`” (period). If words are not separated by delimiters (as in, for example, Chinese), the `FULLTEXT` parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a `FULLTEXT` index, you must preprocess them so that they are separated by some arbitrary delimiter such as “`"`”.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is four characters.
- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overwritten by a user-defined list.

The default stopword list is given in Section 12.9.4, “Full-Text Stopwords”. The default minimum word length and stopword list can be changed as described in Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Consequently, a word that is present in many documents has a lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results. For example, although the word “MySQL” is present in every row of the `articles` table shown earlier, a search for the word produces no results:

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The search result is empty because the word “MySQL” is present in at least 50% of the rows. As such, it is effectively treated as a stopword. For large data sets, this is the most desirable behavior: A natural language query should not return every second row from a 1GB table. For small data sets, it may be less desirable.

A word that matches half of the rows in a table is less likely to locate relevant documents. In fact, it most likely finds plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that rows containing the word are assigned a low semantic value for *the particular data set in which they occur*. A given word may reach the 50% threshold in one data set but not another.

The 50% threshold has a significant implication when you first try full-text searching to see how it works: If you create a table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results. Be sure to insert at least three rows, and preferably many more. Users who need to bypass the 50% limitation can use the boolean search mode; see Section 12.9.2, “Boolean Full-Text Searches”.

12.9.2 Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the `IN BOOLEAN MODE` modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the `+` and `-` operators indicate that a word is required to be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    -> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-----------------------+------------------------------------+
| id | title                 | body                               |
+----+-----------------------+------------------------------------+
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...       |
|  2 | How To Use MySQL Well | After you went through a ...       |
|  3 | Optimizing MySQL      | In this tutorial we will show ...  |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ... |
|  6 | MySQL Security        | When configured properly, MySQL ... |
+----+-----------------------+------------------------------------+
```

Note

In implementing this feature, MySQL uses what is sometimes referred to as *implied Boolean logic*, in which

- `+` stands for `AND`
- `-` stands for `NOT`
- [*no operator*] implies `OR`

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold.
- They do not automatically sort rows in order of decreasing relevance. You can see this from the preceding query result: The row with the highest relevance is the one that contains “MySQL” twice, but it is listed last, not first.
- They can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The minimum and maximum word length full-text parameters apply.
- The stopword list applies.

The boolean full-text search capability supports the following operators:

- `+`

  A leading plus sign indicates that this word *must* be present in each row that is returned.

- `-`

  A leading minus sign indicates that this word must *not* be present in any of the rows that are returned.

  Note: The `-` operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by `-` returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

  By default (when neither `+` nor `-` is specified) the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.

- `> <`

  These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example following this list.

- `( )`

  Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

  A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

  The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it should be *appended* to the word to be affected. Words match if they begin with the word preceding the `*` operator.

  If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short (as determined from the `ft_min_word_len` setting) or a stopword. This occurs because the word is not seen as too short or a stopword, but as a prefix that must be present in the document in the form of a word that begins with the prefix. Suppose that `ft_min_word_len=4`. `ft_min_word_len=4`. Then a search for `'+word +the*'` will likely return fewer rows than a search for `'+word +the'`:

  - The former query remains as is and requires both *`word`* and `the*` (a word starting with `the`) to be present in the document.
  - The latter query is transformed to *`+word`* (requiring only *`word`* to be present). `the` is both too short and a stopword, and either condition is enough to cause it to be ignored.

- `"`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A phrase that is enclosed within double quote (“`"`”) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. Prior to MySQL 5.0.3, the engine then performed a substring search for the phrase in the records that were found, so the match must include nonword characters in the phrase. As of MySQL 5.0.3, nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"` in MySQL 5.0.3, but not before.

If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

  Find rows that contain at least one of the two words.

- `'+apple +juice'`

  Find rows that contain both words.

- `'+apple macintosh'`

  Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

  Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

  Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

  Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

  Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

  Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the “`"`” characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

12.9.3 Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body)
    -> AGAINST ('database' WITH QUERY EXPANSION);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  3 | Optimizing MySQL  | In this tutorial we will show ...        |
+----+-------------------+------------------------------------------+
3 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, it is meaningful to use only when a search phrase is rather short.

12.9.4 Full-Text Stopwords

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the `character_set_server` and `collation_server` system variables). False hits or misses may occur for stopword lookups if the stopword file or columns used for full-text indexing or searches have a character set or collation different from `character_set_server` or `collation_server`.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case insensitive if the collation is `latin1_swedish_ci`, whereas lookups are case sensitive if the collation is `latin1_general_cs` or `latin1_bin`.

The following table shows the default list of full-text stopwords. In a MySQL source distribution, you can find this list in the `myisam/ft_static.c` file.

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow

This documentation is for an older version. If you're
This documentation is for an older version. If you're

allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored

This documentation is for an older version. If you're
This documentation is for an older version. If you're

immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	known	knows	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends

This documentation is for an older version. If you're
This documentation is for an older version. If you're

th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
wouldn't	yes	yet	you	you'd
you'll	you're	you've	your	yours
yourself	yourselves	zero		

12.9.5 Full-Text Restrictions

- Full-text searches are supported for `MyISAM` tables only.
- Full-text searches can be used with most multibyte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set. However, although `FULLTEXT` indexes on `ucs2` columns cannot be used, you can perform `IN BOOLEAN MODE` searches on a `ucs2` column that has no such index.
- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the `FULLTEXT` parser *cannot determine where words begin and end in these and other such languages*. The implications of this and some workarounds for the problem are described in Section 12.9, “Full-Text Search Functions”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Although the use of multiple character sets within a single table is supported, all columns in a `FULLTEXT` index must use the same character set and collation.
- The `MATCH()` column list must match exactly the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` is `IN BOOLEAN MODE`. Boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.
- The argument to `AGAINST()` must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.
- Index hints are more limited for `FULLTEXT` searches than for non-`FULLTEXT` searches. See Section 8.9.2, “Index Hints”.
- The '%' character is not a supported wildcard character for full-text searches.

12.9.6 Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.17, “Installing MySQL from Source”.

Full-text search is carefully tuned for the most effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing*.

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing so are given later in this section.

- The minimum and maximum lengths of words to be indexed are defined by the `ft_min_word_len` and `ft_max_word_len` system variables. (See Section 5.1.4, “Server System Variables”.) The default minimum value is four characters; the default maximum is version dependent. If you change either value, you must rebuild your `FULLTEXT` indexes. For example, if you want three-character words to be searchable, you can set the `ft_min_word_len` variable by putting the following lines in an option file:

  ```
  [mysqld]
  ft_min_word_len=3
  ```

  Then restart the server and rebuild your `FULLTEXT` indexes. Note particularly the remarks regarding `myisamchk` in the instructions following this list.

- To override the default stopword list, set the `ft_stopword_file` system variable. (See Section 5.1.4, “Server System Variables”.) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your `FULLTEXT` indexes.

  The stopword list is free-form. That is, you may use any nonalphanumeric character such as newline, space, or comma to separate stopwords. Exceptions are the underscore character (“_”) and a single apostrophe (“'”) which are treated as part of a word. The character set of the stopword list is the server's default character set; see Section 10.1.3.1, “Server Character Set and Collation”.

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in `myisam/ftdefs.h`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case.

Note

By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the `MATCH()` function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches, set the `ft_boolean_syntax` system variable. This variable can be changed while the server is running, but you must have the `SUPER` privilege to do so. No rebuilding of indexes is necessary in this case. See Section 5.1.4, “Server System Variables”, which describes the rules governing how to set this variable.

- If you want to change the set of characters that are considered word characters, you can do so in several ways, as described in the following list. After making the modification, you must rebuild the indexes for each table that contains any `FULLTEXT` indexes. Suppose that you want to treat the hyphen character ('-') as a word character. Use one of these methods:

  - Modify the MySQL source: In `myisam/ftdefs.h`, see the `true_word_char()` and `misc_word_char()` macros. Add `'-'` to one of those macros and recompile MySQL.

  - Modify a character set file: This requires no recompilation. The `true_word_char()` macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the `<ctype><map>` array in one of the character set XML files to specify that `'-'` is a “letter.” Then use the given character set for your `FULLTEXT` indexes. For information about the `<ctype><map>` array format, see Section 10.3.1, “Character Definition Arrays”.

  - Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For general information about adding collations, see Section 10.4, “Adding a Collation to a Character Set”. For an example specific to full-text indexing, see Section 12.9.7, “Adding a Collation for Full-Text Indexing”.

If you modify full-text variables that affect indexing (`ft_min_word_len`, `ft_max_word_len`, or `ft_stopword_file`), or if you change the stopword file itself, you must rebuild your `FULLTEXT` indexes after making the changes and restarting the server. To rebuild the indexes in this case, it is sufficient to do a `QUICK` repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Alternatively, use `ALTER TABLE` with the `DROP INDEX` and `ADD INDEX` options to drop and re-create each `FULLTEXT` index. In some cases, this may be faster than a repair operation.

Each table that contains any `FULLTEXT` index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

If you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values for `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` for index modification is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE` statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.9.7 Adding a Collation for Full-Text Indexing

This section describes how to add a new collation for full-text searches. The sample collation is like `latin1_swedish_ci` but treats the `'-'` character as a letter rather than as a punctuation character so that it can be indexed as a word character. General information about adding collations is given in Section 10.4, “Adding a Collation to a Character Set”; it is assumed that you have read it and are familiar with the files involved.

To add a collation for full-text indexing, use this procedure:

1. Add a collation to the `Index.xml` file. The collation ID must be unused, so choose a value different from 62 if that ID is already taken on your system.

   ```
   <charset name="latin1">
   ...
   <collation name="latin1_fulltext_ci" id="62"/>
   </charset>
   ```

2. Declare the sort order for the collation in the `latin1.xml` file. In this case, the order can be copied from `latin1_swedish_ci`:

   ```
   <collation name="latin1_fulltext_ci">
   <map>
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
   20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
   30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
   40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
   50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
   60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
   50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
   80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
   90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
   A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
   B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
   41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
44 4E 4F 4F 4F 4F 5D D7 D8 55 55 55 59 59 DE DF
41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5D F7 D8 55 55 55 59 59 DE FF
</map>
</collation>
```

3. Modify the `ctype` array in `latin1.xml`. Change the value corresponding to 0x2D (which is the code for the `'-'` character) from 10 (punctuation) to 01 (small letter). In the following array, this is the element in the fourth row down, third value from the end.

```
<ctype>
<map>
00
20 20 20 20 20 20 20 20 20 28 28 28 28 28 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
48 10 10 10 10 10 10 10 10 10 10 10 10 01 10 10
84 84 84 84 84 84 84 84 84 10 10 10 10 10 10 10
10 81 81 81 81 81 81 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 10 10 10 10 10 10
10 82 82 82 82 82 82 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02 02 02 02 10 10 10 10 20
10 00 10 02 10 10 10 10 10 10 10 01 10 01 00 01 00
00 10 10 10 10 10 10 10 10 10 02 10 02 00 02 01
48 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 10 01 01 01 01 01 01 01 02
02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 10 02 02 02 02 02 02 02 02
</map>
</ctype>
```

4. Restart the server.

5. To employ the new collation, include it in the definition of columns that are to use it:

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE t1 (
    -> a TEXT CHARACTER SET latin1 COLLATE latin1_fulltext_ci,
    -> FULLTEXT INDEX(a)
    -> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.47 sec)
```

6. Test the collation to verify that hyphen is considered as a word character:

```
mysql> INSERT INTO t1 VALUEs ('----'),('....'),('abcd');
Query OK, 3 rows affected (0.22 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1 WHERE MATCH a AGAINST ('----' IN BOOLEAN MODE);
+------+
| a    |
+------+
| ---- |
+------+
1 row in set (0.00 sec)
```

12.10 Cast Functions and Operators

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 12.14 Cast Functions

Name	Description
BINARY	Cast a string to a binary string
CAST()	Cast a value as a certain type
CONVERT()	Cast a value as a certain type

- BINARY

  The BINARY operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column is not defined as BINARY or BLOB. BINARY also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

  In a comparison, BINARY affects the entire operation; it can be given before either operand with the same result.

  BINARY *str* is shorthand for CAST(*str* AS BINARY).

  Note that in some contexts, if you cast an indexed column to BINARY, MySQL is not able to use the index efficiently.

- CAST(*expr* AS *type*)

  The CAST() function takes an expression of any type and produces a result value of a specified type, similar to CONVERT(). See the description of CONVERT() for more information.

- CONVERT(*expr*,*type*), CONVERT(*expr* USING *transcoding_name*)

  The CONVERT() and CAST() functions take an expression of any type and produce a result value of a specified type.

  CAST() and CONVERT(... USING ...) are standard SQL syntax. The non-USING form of CONVERT() is ODBC syntax.

  CONVERT() with USING converts data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string 'abc' in the default character set to the corresponding string in the utf8 character set:

```
SELECT CONVERT('abc' USING utf8);
```

  The *type* for the result can be one of the following values:

  - BINARY[(*N*)]
  - CHAR[(*N*)]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `DATE`
- `DATETIME`
- `DECIMAL[(M[,D])]`
- `SIGNED [INTEGER]`
- `TIME`
- `UNSIGNED [INTEGER]`

`BINARY` produces a string with the `BINARY` data type. See Section 11.4.2, "The BINARY and VARBINARY Types" for a description of how this affects comparisons. If the optional length *`N`* is given, `BINARY(N)` causes the cast to use no more than *`N`* bytes of the argument. As of MySQL 5.0.17, values shorter than *`N`* bytes are padded with `0x00` bytes to a length of *`N`*.

`CHAR(N)` causes the cast to use no more than *`N`* characters of the argument.

The `DECIMAL` type is available as of MySQL 5.0.8.

Normally, you cannot compare a `BLOB` value or other binary string in case-insensitive fashion because binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the `CONVERT()` function to convert the value to a nonbinary string. Comparisons of the result use the string collation. For example, if the character set of the result has a case-insensitive collation, a `LIKE` operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

To use a different character set, substitute its name for `latin1` in the preceding statement. To specify a particular collation for the converted string, use a `COLLATE` clause following the `CONVERT()` call, as described in Section 10.1.9.2, "CONVERT() and CAST()". For example, to use `latin1_german1_ci`:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) COLLATE latin1_german1_ci
  FROM tbl_name;
```

`CONVERT()` can be used more generally for comparing strings that are represented in different character sets.

`LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

The cast functions are useful when you want to create a column with a specific type in a `CREATE TABLE ... SELECT` statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The functions also can be useful for sorting `ENUM` columns in lexical order. Normally, sorting of `ENUM` columns occurs using the internal numeric values. Casting the values to `CHAR` results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST(str AS BINARY)` is the same thing as `BINARY str`. `CAST(expr AS CHAR)` treats the expression as a string with the default character set.

`CAST()` also changes the result if you use it as part of a more complex expression such as `CONCAT('Date: ',CAST(NOW() AS DATE))`.

You should not use `CAST()` to extract data in different formats but instead use string functions like `LEFT()` or `EXTRACT()`. See Section 12.7, “Date and Time Functions”.

To cast a string to a numeric value in numeric context, you normally do not have to do anything other than to use the string value as though it were a number:

```
mysql> SELECT 1+'1';
       -> 2
```

If you use a string in an arithmetic operation, it is converted to a floating-point number during expression evaluation.

If you use a number in string context, the number automatically is converted to a string:

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

For information about implicit conversion of numbers to strings, see Section 12.2, “Type Conversion in Expression Evaluation”.

When using an explicit `CAST()` on a `TIMESTAMP` value in a statement that does not select from any tables, the value is treated by MySQL as a string prior to performing any conversion. This results in the value being truncated when casting to a numeric type, as shown here:

```
mysql> SELECT CAST(TIMESTAMP '2014-09-08 18:07:54' AS SIGNED);
+-----------------------------------------------+
| CAST(TIMESTAMP '2014-09-08 18:07:54' AS SIGNED) |
+-----------------------------------------------+
|                                          2014 |
+-----------------------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------------------------+
| Level   | Code | Message                                                     |
+---------+------+-------------------------------------------------------------+
| Warning | 1292 | Truncated incorrect INTEGER value: '2014-09-08 18:07:54' |
+---------+------+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

This does not apply when selecting rows from a table, as shown here:

```
mysql> USE test;

Database changed
mysql> CREATE TABLE c_test (col TIMESTAMP);
Query OK, 0 rows affected (0.07 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> INSERT INTO c_test VALUES ('2014-09-08 18:07:54');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT col, CAST(col AS UNSIGNED) AS c_col FROM c_test;
+---------------------+----------------+
| col                 | c_col          |
+---------------------+----------------+
| 2014-09-08 18:07:54 | 20140908180754 |
+---------------------+----------------+
1 row in set (0.00 sec)
```

This is a known issue which is resolved in MySQL 5.6.

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as `+` or `-`) and one of the operands is an unsigned integer, the result is unsigned by default (see Section 12.6.1, “Arithmetic Operators”). You can override this by using the `SIGNED` or `UNSIGNED` cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED);
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
        -> -1
```

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, `DECIMAL` column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

The SQL mode affects the result of conversion operations. Examples:

- If you convert a “zero” date string to a date, `CONVERT()` and `CAST()` return `NULL` when the `NO_ZERO_DATE` SQL mode is enabled. As of MySQL 5.0.4, they also produce a warning.
- For integer subtraction, if the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the subtraction result is signed even if any operand is unsigned.

For more information, see Section 5.1.7, “Server SQL Modes”.

12.11 Bit Functions and Operators

Table 12.15 Bit Functions and Operators

Name	Description
`BIT_COUNT()`	Return the number of bits that are set
`&`	Bitwise AND
`~`	Bitwise inversion
`|`	Bitwise OR
`^`	Bitwise XOR
`<<`	Left shift
`>>`	Right shift

Bit functions and operators comprise `BIT_COUNT()`, `BIT_AND()`, `BIT_OR()`, `BIT_XOR()`, `&`, `|`, `^`, `~`, `<<`, and `>>`. (`BIT_AND()`, `BIT_OR()`, and `BIT_XOR()` are aggregate functions described at Section 12.16.1, “GROUP BY (Aggregate) Function Descriptions”.) Bit functions and operators require `BIGINT` (64-bit integer) arguments and return `BIGINT` values, so they have a maximum range of 64 bits. Arguments

This documentation is for an older version. If you're
This documentation is for an older version. If you're

of other types (such as the `BINARY` and `VARBINARY` binary string types) are converted to `BIGINT` and truncation might occur.

The following list describes available bit functions and operators:

- `|`

  Bitwise OR:

  ```
  mysql> SELECT 29 | 15;
         -> 31
  ```

  The result is an unsigned 64-bit integer.

- `&`

  Bitwise AND:

  ```
  mysql> SELECT 29 & 15;
         -> 13
  ```

  The result is an unsigned 64-bit integer.

- `^`

  Bitwise XOR:

  ```
  mysql> SELECT 1 ^ 1;
         -> 0
  mysql> SELECT 1 ^ 0;
         -> 1
  mysql> SELECT 11 ^ 3;
         -> 8
  ```

  The result is an unsigned 64-bit integer.

- `<<`

  Shifts a longlong (`BIGINT`) number to the left.

  ```
  mysql> SELECT 1 << 2;
         -> 4
  ```

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- `>>`

  Shifts a longlong (`BIGINT`) number to the right.

  ```
  mysql> SELECT 4 >> 2;
         -> 1
  ```

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- `~`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Invert all bits.

```
mysql> SELECT 5 & ~1;
        -> 4
```

The result is an unsigned 64-bit integer.

- `BIT_COUNT(N)`

  Returns the number of bits that are set in the argument *N*.

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
        -> 4, 3
```

12.12 Encryption and Compression Functions

Table 12.16 Encryption Functions

Name	Description
`AES_DECRYPT()`	Decrypt using AES
`AES_ENCRYPT()`	Encrypt using AES
`COMPRESS()`	Return result as a binary string
`DECODE()`	Decodes a string encrypted using ENCODE()
`DES_DECRYPT()`	Decrypt a string
`DES_ENCRYPT()`	Encrypt a string
`ENCODE()`	Encode a string
`ENCRYPT()`	Encrypt a string
`MD5()`	Calculate MD5 checksum
`OLD_PASSWORD()`	Return the value of the pre-4.1 implementation of PASSWORD
`PASSWORD()`	Calculate and return a password string
`SHA1()`, `SHA()`	Calculate an SHA-1 160-bit checksum
`UNCOMPRESS()`	Uncompress a string compressed
`UNCOMPRESSED_LENGTH()`	Return the length of a string before compression

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a `VARBINARY` or `BLOB` binary string data type. This will avoid potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (`CHAR`, `VARCHAR`, `TEXT`).

For functions such as `MD5()` or `SHA1()` that return a string of hex digits, the return value cannot be converted to uppercase or compared in case-insensitive fashion as is. You must convert the value to a nonbinary string. See the discussion of binary string conversion in Section 12.10, “Cast Functions and Operators”.

If an application stores values from a function such as `MD5()` or `SHA1()` that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using `UNHEX()` and storing the result in a `BINARY(N)` column. Each pair of hex digits requires one byte in binary form, so the value of *N* depends on the length of the hex string. *N* is 16 for an `MD5()` value and 20 for a `SHA1()` value.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The size penalty for storing the hex string in a `CHAR` column is at least two times, up to six times if the value is stored in a column that uses the `utf8` character set (where each character uses 3 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores `MD5()` string values in a `CHAR(32)` column:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use `UNHEX()` and `BINARY(16)` instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.

Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead.

Caution

Passwords or other sensitive values supplied as arguments to encryption functions are sent in cleartext to the MySQL server unless an SSL connection is used. Also, such values will appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- `AES_DECRYPT(`*`crypt_str`*`,`*`key_str`*`)`

  This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of `AES_ENCRYPT()`.

- `AES_ENCRYPT(`*`str`*`,`*`key_str`*`)`

  `AES_ENCRYPT()` and `AES_DECRYPT()` implement encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” The AES standard permits various key lengths. These functions implement AES with a 128-bit key length, but you can extend them to 256 bits by modifying the source. The key length is a trade off between performance and security.

  `AES_ENCRYPT()` encrypts the string *`str`* using the key string *`key_str`* and returns a binary string containing the encrypted output. `AES_DECRYPT()` decrypts the encrypted string *`crypt_str`* using the key string *`key_str`* and returns the original cleartext string. If either function argument is `NULL`, the function returns `NULL`.

  The *`str`* and *`crypt_str`* arguments can be any length, and padding is automatically added to *`str`* so it is a multiple of a block as required by block-based algorithms such as AES. This padding is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

automatically removed by the AES_DECRYPT() function. The length of *crypt_str* can be calculated using this formula:

```
16 * (trunc(string_length / 16) + 1)
```

For a key length of 128 bits, the most secure way to pass a key to the *key_str* argument is to create a truly random 128-bit value and pass it as a binary value. For example:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')));
```

A passphrase can be used to generate an AES key by hashing the passphrase. For example:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text', SHA1('My secret passphrase')));
```

Do not pass a password or passphrase directly to *crypt_str*, hash it first. Previous versions of this documentation suggested the former approach, but it is no longer recommended as the examples shown here are more secure.

If AES_DECRYPT() detects invalid data or incorrect padding, it returns NULL. However, it is possible for AES_DECRYPT() to return a non-NULL value (possibly garbage) if the input data or the key is invalid.

- COMPRESS(*string_to_compress*)

  Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as zlib. Otherwise, the return value is always NULL. The compressed string can be uncompressed with UNCOMPRESS().

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
mysql> SELECT LENGTH(COMPRESS(''));
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

  The compressed string contents are stored the following way:

  - Empty strings are stored as empty strings.
  - Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra "." character is added to avoid problems with endspace trimming should the result be stored in a CHAR or VARCHAR column. (However, use of nonbinary string data types such as CHAR or VARCHAR to store compressed strings is not recommended anyway because character set conversion may occur. Use a VARBINARY or BLOB binary string column instead.)

- DECODE(*crypt_str*,*pass_str*)

  Decrypts the encrypted string *crypt_str* using *pass_str* as the password. *crypt_str* should be a string returned from ENCODE().

- DES_DECRYPT(*crypt_str*[,*key_str*])

  Decrypts a string encrypted with DES_ENCRYPT(). If an error occurs, this function returns NULL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This function works only if MySQL has been configured with SSL support. See Section 6.3.6, “Using Secure Connections”.

If no *key_str* argument is given, DES_DECRYPT() examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the SUPER privilege. The key file can be specified with the --des-key-file server option.

If you pass this function a *key_str* argument, that string is used as the key for decrypting the message.

If the *crypt_str* argument does not appear to be an encrypted string, MySQL returns the given *crypt_str*.

- DES_ENCRYPT(*str*[,{*key_num*|*key_str*}])

Encrypts the string with the given key using the Triple-DES algorithm.

This function works only if MySQL has been configured with SSL support. See Section 6.3.6, “Using Secure Connections”.

The encryption key to use is chosen based on the second argument to DES_ENCRYPT(), if one was given. With no argument, the first key from the DES key file is used. With a *key_num* argument, the given key number (0 to 9) from the DES key file is used. With a *key_str* argument, the given key string is used to encrypt *str*.

The key file can be specified with the --des-key-file server option.

The return string is a binary string where the first character is CHAR(128 | *key_num*). If an error occurs, DES_ENCRYPT() returns NULL.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *key_num* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *key_num* value must be a number in the range from 0 to 9. Lines in the file may be in any order. *des_key_str* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to DES_ENCRYPT().

You can tell MySQL to read new key values from the key file with the FLUSH DES_KEY_FILE statement. This requires the RELOAD privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
    > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `ENCODE(str,pass_str)`

  Encrypt *`str`* using *`pass_str`* as the password. The result is a binary string of the same length as *`str`*. To decrypt the result, use `DECODE()`.

  The strength of the encryption is based on how good the random generator is. It should suffice for short strings.

- `ENCRYPT(str[,salt])`

  Encrypts *`str`* using the Unix `crypt()` system call and returns a binary string. The *`salt`* argument must be a string with at least two characters or the result will be `NULL`. If no *`salt`* argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
        -> 'VxuFAJXVARROc'
```

  `ENCRYPT()` ignores all but the first eight characters of *`str`*, at least on some systems. This behavior is determined by the implementation of the underlying `crypt()` system call.

  The use of `ENCRYPT()` with the `ucs2` multibyte character set is not recommended because the system call expects a string terminated by a zero byte.

  If `crypt()` is not available on your system (as is the case with Windows), `ENCRYPT()` always returns `NULL`.

- `MD5(str)`

  Calculates an MD5 128-bit checksum for the string. The value is returned as a binary string of 32 hex digits, or `NULL` if the argument was `NULL`. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

  This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

  See the note regarding the MD5 algorithm at the beginning this section.

- `OLD_PASSWORD(str)`

  `OLD_PASSWORD()` was added when the implementation of `PASSWORD()` was changed in MySQL 4.1 to improve security. `OLD_PASSWORD()` returns the value of the pre-4.1 implementation of `PASSWORD()` as a binary string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version MySQL 5.0 server without locking them out. See Section 6.1.2.4, “Password Hashing in MySQL”.

- `PASSWORD(str)`

  Returns a hashed password string calculated from the cleartext password *`str`*. The return value is a binary string, or `NULL` if the argument is `NULL`. This function is the SQL interface to the algorithm used by the server to encrypt MySQL passwords for storage in the `mysql.user` grant table.

  The `old_passwords` system variable controls the password hashing method used by the `PASSWORD()` function. It also influences password hashing performed by `CREATE USER` and `GRANT` statements that specify a password using an `IDENTIFIED BY` clause.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The value determines whether or not to use “old” native MySQL password hashing. A value of 0 (or `OFF`) causes passwords to be encrypted using the format available from MySQL 4.1 on. A value of 1 (or `ON`) causes password encryption to use the older pre-4.1 format.

If `old_passwords=1`, `PASSWORD(str)` returns the same value as `OLD_PASSWORD(str)`. The latter function is not affected by the value of `old_passwords`.

```
mysql> SET old_passwords = 0;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+-------------------------------------------+------------------------+
| PASSWORD('mypass')                        | OLD_PASSWORD('mypass') |
+-------------------------------------------+------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 | 6f8c114b58f2ce9e       |
+-------------------------------------------+------------------------+

mysql> SET old_passwords = 1;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+-------------------+------------------------+
| PASSWORD('mypass') | OLD_PASSWORD('mypass') |
+-------------------+------------------------+
| 6f8c114b58f2ce9e   | 6f8c114b58f2ce9e       |
+-------------------+------------------------+
```

Encryption performed by `PASSWORD()` is one-way (not reversible). It is not the same type of encryption as used for Unix passwords; for that, use `ENCRYPT()`.

Note

`PASSWORD()` is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider `MD5()` or `SHA1()` instead. Also see RFC 2195, section 2 (Challenge-Response Authentication Mechanism (CRAM)), for more information about handling passwords and authentication securely in your applications.

Caution

Statements that invoke `PASSWORD()` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about password logging in the server logs, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Logging”.

- `SHA1(str)`, `SHA(str)`

Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a binary string of 40 hex digits, or `NULL` if the argument was `NULL`. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. You can also use `SHA1()` as a cryptographic function for storing passwords. `SHA()` is synonymous with `SHA1()`.

```
mysql> SELECT SHA1('abc');
       -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

`SHA1()` can be considered a cryptographically more secure equivalent of `MD5()`. However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `UNCOMPRESS(string_to_uncompress)`

  Uncompresses a string compressed by the `COMPRESS()` function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
       -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
       -> NULL
```

- `UNCOMPRESSED_LENGTH(compressed_string)`

  Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
       -> 30
```

12.13 Information Functions

Table 12.17 Information Functions

Name	Description
`BENCHMARK()`	Repeatedly execute an expression
`CHARSET()`	Return the character set of the argument
`COERCIBILITY()`	Return the collation coercibility value of the string argument
`COLLATION()`	Return the collation of the string argument
`CONNECTION_ID()`	Return the connection ID (thread ID) for the connection
`CURRENT_USER()`, `CURRENT_USER`	The authenticated user name and host name
`DATABASE()`	Return the default (current) database name
`FOUND_ROWS()`	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
`LAST_INSERT_ID()`	Value of the AUTOINCREMENT column for the last INSERT
`ROW_COUNT()`	The number of rows updated
`SCHEMA()`	Synonym for DATABASE()
`SESSION_USER()`	Synonym for USER()
`SYSTEM_USER()`	Synonym for USER()
`USER()`	The user name and host name provided by the client
`VERSION()`	Return a string that indicates the MySQL server version

- `BENCHMARK(count,expr)`

  The `BENCHMARK()` function executes the expression `expr` repeatedly `count` times. It may be used to time how quickly MySQL processes the expression. The result value is always `0`. The intended use is from within the `mysql` client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+----------------------------------------------+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+----------------------------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute `BENCHMARK()` several times, and to interpret the result with regard to how heavily loaded the server machine is.

`BENCHMARK()` is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

- Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, `BENCHMARK(10, (SELECT * FROM t))` will fail if the table `t` has more than one column or more than one row.
- Executing a `SELECT expr` statement *`N`* times differs from executing `SELECT BENCHMARK(N, expr)` in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *`N`* times each. The latter involves only runtime evaluation *`N`* times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of `BENCHMARK()` thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

- `CHARSET(str)`

  Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

- `COERCIBILITY(str)`

  Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
```

The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
`0`	Explicit collation	Value with `COLLATE` clause
`1`	No collation	Concatenation of strings with different collations
`2`	Implicit collation	Column value

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Coercibility	Meaning	Example
3	System constant	`USER()` return value
4	Coercible	Literal string
5	Ignorable	`NULL` or an expression derived from `NULL`

Before MySQL 5.0.3, the return values are shown as follows, and functions such as `USER()` have a coercibility of 2:

Coercibility	Meaning	Example
0	Explicit collation	Value with `COLLATE` clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value, stored routine parameter or local variable
3	Coercible	Literal string

- `COLLATION(str)`

  Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
        -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
        -> 'utf8_general_ci'
```

- `CONNECTION_ID()`

  Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

  The value returned by `CONNECTION_ID()` is the same type of value as displayed in the `Id` column of `SHOW PROCESSLIST` output.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

- `CURRENT_USER`, `CURRENT_USER()`

  Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the `utf8` character set.

  The value of `CURRENT_USER()` can differ from the value of `USER()`.

```
mysql> SELECT USER();
        -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
        -> '@localhost'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The example illustrates that although the client specified a user name of `davida` (as indicated by the value of the `USER()` function), the server authenticated the client using an anonymous user account (as seen by the empty user name part of the `CURRENT_USER()` value). One way this might occur is that there is no account listed in the grant tables for `davida`.

Within a stored program or view, `CURRENT_USER()` returns the account for the user who defined the object (as given by its `DEFINER` value) unless defined with the `SQL SECURITY INVOKER` characteristic. In the latter case, `CURRENT_USER()` returns the object's invoker. This applies to stored programs as of MySQL 5.0.10 and to views as of MySQL 5.0.24. (For older versions, `CURRENT_USER()` returns the account for the object's invoker.)

Triggers and events have no option to define the `SQL SECURITY` characteristic, so for these objects, `CURRENT_USER()` returns the account for the user who defined the object. To return the invoker, use `USER()` or `SESSION_USER()`.

- `DATABASE()`

  Returns the default (current) database name as a string in the `utf8` character set. If there is no default database, `DATABASE()` returns `NULL`. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

```
mysql> SELECT DATABASE();
        -> 'test'
```

- `FOUND_ROWS()`

  A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include a `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

  The second `SELECT` returns a number indicating how many rows the first `SELECT` would have returned had it been written without the `LIMIT` clause.

  In the absence of the `SQL_CALC_FOUND_ROWS` option in the most recent successful `SELECT` statement, `FOUND_ROWS()` returns the number of rows in the result set returned by that statement. If the statement includes a `LIMIT` clause, `FOUND_ROWS()` returns the number of rows up to the limit. For example, `FOUND_ROWS()` returns 10 or 60, respectively, if the statement includes `LIMIT 10` or `LIMIT 50, 10`.

  The row count available through `FOUND_ROWS()` is transient and not intended to be available past the statement following the `SELECT SQL_CALC_FOUND_ROWS` statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

  If you are using `SELECT SQL_CALC_FOUND_ROWS`, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without `LIMIT`, because the result set need not be sent to the client.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using `FOUND_ROWS()` enables you to determine how many other pages are needed for the rest of the result.

The use of `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` is more complex for `UNION` statements than for simple `SELECT` statements, because `LIMIT` may occur at multiple places in a `UNION`. It may be applied to individual `SELECT` statements in the `UNION`, or global to the `UNION` result as a whole.

The intent of `SQL_CALC_FOUND_ROWS` for `UNION` is that it should return the row count that would be returned without a global `LIMIT`. The conditions for use of `SQL_CALC_FOUND_ROWS` with `UNION` are:

- The `SQL_CALC_FOUND_ROWS` keyword must appear in the first `SELECT` of the `UNION`.
- The value of `FOUND_ROWS()` is exact only if `UNION ALL` is used. If `UNION` without `ALL` is used, duplicate removal occurs and the value of `FOUND_ROWS()` is only approximate.
- If no `LIMIT` is present in the `UNION`, `SQL_CALC_FOUND_ROWS` is ignored and returns the number of rows in the temporary table that is created to process the `UNION`.

Beyond the cases described here, the behavior of `FOUND_ROWS()` is undefined (for example, its value following a `SELECT` statement that fails with an error).

Important

`FOUND_ROWS()` is not replicated reliably, and should not be used with databases that are to be replicated.

- `LAST_INSERT_ID()`, `LAST_INSERT_ID(`*`expr`*`)`

`LAST_INSERT_ID()` (with no argument) returns a `BIGINT` (64-bit) value representing the first automatically generated value that was set for an `AUTO_INCREMENT` column by the *most recently executed* `INSERT` statement to affect such a column. For example, after inserting a row that generates an `AUTO_INCREMENT` value, you can get the value like this:

```
mysql> SELECT LAST_INSERT_ID();
        -> 195
```

if a table contains an `AUTO_INCREMENT` column and `INSERT ... ON DUPLICATE KEY UPDATE` updates (rather than inserts) a row, the value of `LAST_INSERT_ID()` is not meaningful. For a workaround, see Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

The currently executing statement does not affect the value of `LAST_INSERT_ID()`. Suppose that you generate an `AUTO_INCREMENT` value with one statement, and then refer to `LAST_INSERT_ID()` in a multiple-row `INSERT` statement that inserts rows into a table with its own `AUTO_INCREMENT` column. The value of `LAST_INSERT_ID()` will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to `LAST_INSERT_ID()` and `LAST_INSERT_ID(`*`expr`*`)`, the effect is undefined.)

If the previous statement returned an error, the value of `LAST_INSERT_ID()` is undefined. For transactional tables, if the statement is rolled back due to an error, the value of `LAST_INSERT_ID()` is left undefined. For manual `ROLLBACK`, the value of `LAST_INSERT_ID()` is not restored to that before the transaction; it remains as it was at the point of the `ROLLBACK`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value. (Before MySQL 5.0.12, the value is not restored and following statements do see a changed value.)

The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first `AUTO_INCREMENT` value generated for most recent statement affecting an `AUTO_INCREMENT` column *by that client*. This value cannot be affected by other clients, even if they generate `AUTO_INCREMENT` values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of `LAST_INSERT_ID()` is not changed if you set the `AUTO_INCREMENT` column of a row to a non-“magic” value (that is, a value that is not `NULL` and not `0`).

Important

If you insert multiple rows using a single `INSERT` statement, `LAST_INSERT_ID()` returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same `INSERT` statement against some other server.

For example:

```
mysql> USE test;
Database changed
mysql> CREATE TABLE t (
    ->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   name VARCHAR(10) NOT NULL
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t VALUES (NULL, 'Bob');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
+----+------+
1 row in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO t VALUES
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    -> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+------+
4 rows in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
1 row in set (0.00 sec)
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was `2`, and it is this value that is returned by `LAST_INSERT_ID()` for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `AUTO_INCREMENT` counter is not incremented and `LAST_INSERT_ID()` returns `0`, which reflects that no row was inserted.

If *`expr`* is given as an argument to `LAST_INSERT_ID()`, the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()`. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

   ```
   mysql> CREATE TABLE sequence (id INT NOT NULL);
   mysql> INSERT INTO sequence VALUES (0);
   ```

2. Use the table to generate sequence numbers like this:

   ```
   mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
   mysql> SELECT LAST_INSERT_ID();
   ```

   The `UPDATE` statement increments the sequence counter and causes the next call to `LAST_INSERT_ID()` to return the updated value. The `SELECT` statement retrieves that value. The `mysql_insert_id()` C API function can also be used to get the value. See Section 20.6.7.37, “mysql_insert_id()”.

You can generate sequences without calling `LAST_INSERT_ID()`, but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the `UPDATE` statement and get their own sequence value with the `SELECT` statement (or `mysql_insert_id()`), without affecting or being affected by other clients that generate their own sequence values.

Note that `mysql_insert_id()` is only updated after `INSERT` and `UPDATE` statements, so you cannot use the C API function to retrieve the value for `LAST_INSERT_ID(`*`expr`*`)` after executing other SQL statements like `SELECT` or `SET`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `ROW_COUNT()`

  `ROW_COUNT()` returns the number of rows changed, deleted, or inserted by the last statement if it was an `UPDATE`, `DELETE`, or `INSERT`. For other statements, the value may not be meaningful.

  For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows “found”; that is, matched by the `WHERE` clause.

  For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

  For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value is 1 if the row is inserted as a new row and 2 if an existing row is updated.

  The `ROW_COUNT()` value is similar to the value from the `mysql_affected_rows()` C API function and the row count that the `mysql` client displays following statement execution.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

  `ROW_COUNT()` was added in MySQL 5.0.1.

  Important

  `ROW_COUNT()` is not replicated reliably.

- `SCHEMA()`

  This function is a synonym for `DATABASE()`. It was added in MySQL 5.0.2.

- `SESSION_USER()`

  `SESSION_USER()` is a synonym for `USER()`.

- `SYSTEM_USER()`

  `SYSTEM_USER()` is a synonym for `USER()`.

- `USER()`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Returns the current MySQL user name and host name as a string in the `utf8` character set.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of `CURRENT_USER()`.

- `VERSION()`

  Returns a string that indicates the MySQL server version. The string uses the `utf8` character set. The value might have a suffix in addition to the version number. See the description of the `version` system variable in Section 5.1.4, “Server System Variables”.

```
mysql> SELECT VERSION();
        -> '5.0.96-standard'
```

12.14 Spatial Analysis Functions

MySQL provides functions to perform various operations on spatial data. These functions can be grouped into several major categories according to the type of operation they perform:

- Functions that create geometries in various formats (WKT, WKB, internal)
- Functions that convert geometries between formats
- Functions that access qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

For general background about MySQL support for using spatial data, see Section 11.5, “Extensions for Spatial Data”.

12.14.1 Spatial Function Reference

The following table lists each spatial function and provides a short description of each one.

Table 12.18 Spatial Functions

Name	Description
`Area()`	Return Polygon or MultiPolygon area
`AsBinary()`, `AsWKB()`	Convert from internal geometry format to WKB
`AsText()`, `AsWKT()`	Convert from internal geometry format to WKT
`Centroid()`	Return centroid as a point
`Contains()`	Whether MBR of one geometry contains MBR of another
`Crosses()`	Whether one geometry crosses another
`Dimension()`	Dimension of geometry
`Disjoint()`	Whether MBRs of two geometries are disjoint

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
EndPoint()	End Point of LineString
Envelope()	Return MBR of geometry
Equals()	Whether MBRs of two geometries are equal
ExteriorRing()	Return exterior ring of Polygon
GeomCollFromText(), GeometryCollectionFromText()	Return geometry collection from WKT
GeomCollFromWKB(), GeometryCollectionFromWKB()	Return geometry collection from WKB
GeometryCollection()	Construct geometry collection from geometries
GeometryN()	Return N-th geometry from geometry collection
GeometryType()	Return name of geometry type
GeomFromText(), GeometryFromText()	Return geometry from WKT
GeomFromWKB(), GeometryFromWKB()	Return geometry from WKB
GLength()	Return length of LineString
InteriorRingN()	Return N-th interior ring of Polygon
Intersects()	Whether MBRs of two geometries intersect
IsClosed()	Whether a geometry is closed and simple
IsEmpty()	Placeholder function
IsSimple()	Whether a geometry is simple
LineFromText(), LineStringFromText()	Construct LineString from WKT
LineFromWKB(), LineStringFromWKB()	Construct LineString from WKB
LineString()	Construct LineString from Point values
MBRContains()	Whether MBR of one geometry contains MBR of another
MBRDisjoint()	Whether MBRs of two geometries are disjoint
MBREqual()	Whether MBRs of two geometries are equal
MBRIntersects()	Whether MBRs of two geometries intersect
MBROverlaps()	Whether MBRs of two geometries overlap
MBRTouches()	Whether MBRs of two geometries touch
MBRWithin()	Whether MBR of one geometry is within MBR of another
MLineFromText(), MultiLineStringFromText()	Construct MultiLineString from WKT
MLineFromWKB(), MultiLineStringFromWKB()	Construct MultiLineString from WKB
MPointFromText(), MultiPointFromText()	Construct MultiPoint from WKT
MPointFromWKB(), MultiPointFromWKB()	Construct MultiPoint from WKB

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
`MPolyFromText()`, `MultiPolygonFromText()`	Construct MultiPolygon from WKT
`MPolyFromWKB()`, `MultiPolygonFromWKB()`	Construct MultiPolygon from WKB
`MultiLineString()`	Contruct MultiLineString from LineString values
`MultiPoint()`	Construct MultiPoint from Point values
`MultiPolygon()`	Construct MultiPolygon from Polygon values
`NumGeometries()`	Return number of geometries in geometry collection
`NumInteriorRings()`	Return number of interior rings in Polygon
`NumPoints()`	Return number of points in LineString
`Overlaps()`	Whether MBRs of two geometries overlap
`Point()`	Construct Point from coordinates
`PointFromText()`	Construct Point from WKT
`PointFromWKB()`	Construct Point from WKB
`PointN()`	Return N-th point from LineString
`PolyFromText()`, `PolygonFromText()`	Construct Polygon from WKT
`PolyFromWKB()`, `PolygonFromWKB()`	Construct Polygon from WKB
`Polygon()`	Construct Polygon from LineString arguments
`SRID()`	Return spatial reference system ID for geometry
`StartPoint()`	Start Point of LineString
`Touches()`	Whether one geometry touches another
`Within()`	Whether MBR of one geometry is within MBR of another
`X()`	Return X coordinate of Point
`Y()`	Return Y coordinate of Point

12.14.2 Argument Handling by Spatial Functions

Spatial values, or geometries, have the properties described at Section 11.5.2.2, “Geometry Class”. The following discussion lists general spatial function argument-handling characteristics. Specific functions or groups of functions may have additional argument-handling characteristics, as discussed in the sections where those function descriptions occur.

Spatial functions are defined only for valid geometry values. If an invalid geometry is passed to a spatial function, the result is undefined.

The Spatial Reference Identifier (SRID) of a geometry identifies the coordinate space in which the geometry is defined. In MySQL, the SRID value is an integer associated with the geometry value. However, all calculations are done assuming SRID 0, representing cartesian (planar) coordinates, regardless of the actual SRID value. In the future, calculations may use the specified SRID values. To ensure SRID 0 behavior, create geometries using SRID 0. SRID 0 is the default for new geometries if no SRID is specified.

The maximum usable SRID value is $2^{32}-1$. If a larger value is given, only the lower 32 bits are used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Geometry values produced by any spatial function inherit the SRID of the geometry arguments.

12.14.3 Functions That Create Geometry Values from WKT Values

These functions take as arguments a Well-Known Text (WKT) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` accepts a WKT value of any geometry type as its first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

For a description of WKT format, see Well-Known Text (WKT) Format.

- `GeomCollFromText(wkt[,srid])`, `GeometryCollectionFromText(wkt[,srid])`

  Constructs a `GeometryCollection` value using its WKT representation and SRID.

- `GeomFromText(wkt[,srid])`, `GeometryFromText(wkt[,srid])`

  Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(wkt[,srid])`, `LineStringFromText(wkt[,srid])`

  Constructs a `LineString` value using its WKT representation and SRID.

- `MLineFromText(wkt[,srid])`, `MultiLineStringFromText(wkt[,srid])`

  Constructs a `MultiLineString` value using its WKT representation and SRID.

- `MPointFromText(wkt[,srid])`, `MultiPointFromText(wkt[,srid])`

  Constructs a `MultiPoint` value using its WKT representation and SRID.

- `MPolyFromText(wkt[,srid])`, `MultiPolygonFromText(wkt[,srid])`

  Constructs a `MultiPolygon` value using its WKT representation and SRID.

- `PointFromText(wkt[,srid])`

  Constructs a `Point` value using its WKT representation and SRID.

- `PolyFromText(wkt[,srid])`, `PolygonFromText(wkt[,srid])`

  Constructs a `Polygon` value using its WKT representation and SRID.

12.14.4 Functions That Create Geometry Values from WKB Values

These functions take as arguments a `BLOB` containing a Well-Known Binary (WKB) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

As of MySQL 5.0.82, these functions also accept geometry objects for compatibility with the changes made in MySQL 5.0.82 to the return value of the functions in Section 12.14.5, “MySQL-Specific Functions That Create Geometry Values”. Thus, those functions may continue to be used to provide the first argument to the functions in this section.

`GeomFromWKB()` accepts a WKB value of any geometry type as its first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

For a description of WKB format, see Well-Known Binary (WKB) Format.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `GeomCollFromWKB(wkb[,srid])`, `GeometryCollectionFromWKB(wkb[,srid])`

  Constructs a `GeometryCollection` value using its WKB representation and SRID.

- `GeomFromWKB(wkb[,srid])`, `GeometryFromWKB(wkb[,srid])`

  Constructs a geometry value of any type using its WKB representation and SRID.

- `LineFromWKB(wkb[,srid])`, `LineStringFromWKB(wkb[,srid])`

  Constructs a `LineString` value using its WKB representation and SRID.

- `MLineFromWKB(wkb[,srid])`, `MultiLineStringFromWKB(wkb[,srid])`

  Constructs a `MultiLineString` value using its WKB representation and SRID.

- `MPointFromWKB(wkb[,srid])`, `MultiPointFromWKB(wkb[,srid])`

  Constructs a `MultiPoint` value using its WKB representation and SRID.

- `MPolyFromWKB(wkb[,srid])`, `MultiPolygonFromWKB(wkb[,srid])`

  Constructs a `MultiPolygon` value using its WKB representation and SRID.

- `PointFromWKB(wkb[,srid])`

  Constructs a `Point` value using its WKB representation and SRID.

- `PolyFromWKB(wkb[,srid])`, `PolygonFromWKB(wkb[,srid])`

  Constructs a `Polygon` value using its WKB representation and SRID.

12.14.5 MySQL-Specific Functions That Create Geometry Values

MySQL provides a set of useful nonstandard functions for creating geometry values. The functions described in this section are MySQL extensions to the OpenGIS specification.

As of MySQL 5.0.82, these functions produce geometry objects from either WKB values or geometry objects as arguments. If any argument is not a proper WKB or geometry representation of the proper object type, the return value is `NULL`.

Before MySQL 5.0.82, these functions produce `BLOB` values containing WKB representations of geometry values with no SRID from WKB arguments. The WKB value returned from these functions can be converted to geometry arguments by using them as the first argument to functions in the `GeomFromWKB()` function family.

For example, as of MySQL 5.0.82, you can insert the geometry return value from `Point()` directly into a `POINT` column:

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

Prior to MySQL 5.0.82, convert the WKB return value to a `Point` before inserting it:

```
INSERT INTO t1 (pt_col) VALUES(GeomFromWKB(Point(1,2)));
```

- `GeometryCollection(g1,g2,...)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Constructs a `GeometryCollection`.

If the argument contains a nonsupported geometry, the return value is `NULL`.

- `LineString(pt1,pt2,...)`

  Constructs a `LineString` value from a number of `Point` or WKB `Point` arguments. If the number of arguments is less than two, the return value is `NULL`.

- `MultiLineString(ls1,ls2,...)`

  Constructs a `MultiLineString` value using `LineString` or WKB `LineString` arguments.

- `MultiPoint(pt1,pt2,...)`

  Constructs a `MultiPoint` value using `Point` or WKB `Point` arguments.

- `MultiPolygon(poly1,poly2,...)`

  Constructs a `MultiPolygon` value from a set of `Polygon` or WKB `Polygon` arguments.

- `Point(x,y)`

  Constructs a `Point` using its coordinates.

- `Polygon(ls1,ls2,...)`

  Constructs a `Polygon` value from a number of `LineString` or WKB `LineString` arguments. If any argument does not represent a `LinearRing` (that is, not a closed and simple `LineString`), the return value is `NULL`.

12.14.6 Geometry Format Conversion Functions

MySQL supports the functions listed in this section for converting geometry values from internal geometry format to WKT or WKB format.

In addition, there are functions to convert a string from WKT or WKB format to internal geometry format. See Section 12.14.3, “Functions That Create Geometry Values from WKT Values”, and Section 12.14.4, “Functions That Create Geometry Values from WKB Values”.

- `AsBinary(g)`, `AsWKB(g)`

  Converts a value in internal geometry format to its WKB representation and returns the binary result.

```
SELECT AsBinary(g) FROM geom;
```

- `AsText(g)`, `AsWKT(g)`

  Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+--------------------------+
| AsText(GeomFromText(@g)) |
+--------------------------+
| LINESTRING(1 1,2 2,3 3)  |
+--------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

12.14.7 Geometry Property Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, the `Area()` polygon function returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

12.14.7.1 General Geometry Property Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(g)`

  Returns the inherent dimension of the geometry value *g*. The result can be −1, 0, 1, or 2. The meaning of these values is given in Section 11.5.2.2, “Geometry Class”.

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+---------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
```

- `Envelope(g)`

  Returns the minimum bounding rectangle (MBR) for the geometry value *g*. The result is returned as a `Polygon` value that is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----------------------------------------------------+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)'))) |
+-----------------------------------------------------+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                      |
+-----------------------------------------------------+
```

- `GeometryType(g)`

  Returns a binary string indicating the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable `Geometry` subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+---------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+---------------------------------------+
| POINT                                 |
+---------------------------------------+
```

- `IsEmpty(g)`

  This function is a placeholder that returns 0 for any valid geometry value, 1 for any invalid geometry value or `NULL`.

  MySQL does not support GIS `EMPTY` values such as `POINT EMPTY`.

- `IsSimple(g)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In MySQL 5.0, this function is a placeholder that always returns 0.

The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See Section 11.5.2.1, “The Geometry Class Hierarchy”.)

- `SRID(g)`

  Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+---------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+---------------------------------------------+
|                                         101 |
+---------------------------------------------+
```

12.14.7.2 Point Property Functions

A `Point` consists of X and Y coordinates, which may be obtained using the following functions:

- `X(p)`

  Returns the X-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT X(POINT(56.7, 53.34));
+-----------------------+
| X(POINT(56.7, 53.34)) |
+-----------------------+
|                  56.7 |
+-----------------------+
```

- `Y(p)`

  Returns the Y-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT Y(POINT(56.7, 53.34));
+-----------------------+
| Y(POINT(56.7, 53.34)) |
+-----------------------+
|                 53.34 |
+-----------------------+
```

12.14.7.3 LineString and MultiLineString Property Functions

A `LineString` consists of `Point` values. You can extract particular points of a `LineString`, count the number of points that it contains, or obtain its length.

Some functions in this section also work for `MultiLineString` values.

- `EndPoint(ls)`

  Returns the `Point` that is the endpoint of the `LineString` value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
| POINT(3 3)                          |
+-------------------------------------+
```

- `GLength(ls)`

  Returns a double-precision number indicating the length of the `LineString` or `MultiLineString` value *`ls`* in its associated spatial reference. The length of a `MultiLineString` value is equal to the sum of the lengths of its elements.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+----------------------------+
| GLength(GeomFromText(@ls)) |
+----------------------------+
|         2.8284271247461903 |
+----------------------------+

mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----------------------------+
| GLength(GeomFromText(@mls)) |
+-----------------------------+
|           4.242640687119286 |
+-----------------------------+
```

  `GLength()` is a nonstandard name. It corresponds to the OpenGIS `Length()` function. (There is an existing SQL function `Length()` that calculates the length of string values.)

- `IsClosed(ls)`

  For a `LineString` value *`ls`*, `IsClosed()` returns 1 if *`ls`* is closed (that is, its `StartPoint()` and `EndPoint()` values are the same).

  For a `MultiLineString` value *`ls`*, `IsClosed()` returns 1 if *`ls`* is closed (that is, the `StartPoint()` and `EndPoint()` values are the same for each `LineString` in *`ls`*).

  `IsClosed()` returns 0 if *`ls`* is not closed, and `NULL` if *`ls`* is `NULL`.

```
mysql> SET @ls1 = 'LineString(1 1,2 2,3 3,2 2)';
mysql> SET @ls2 = 'LineString(1 1,2 2,3 3,1 1)';

mysql> SELECT IsClosed(GeomFromText(@ls1));
+------------------------------+
| IsClosed(GeomFromText(@ls1)) |
+------------------------------+
|                            0 |
+------------------------------+

mysql> SELECT IsClosed(GeomFromText(@ls2));
+------------------------------+
| IsClosed(GeomFromText(@ls2)) |
+------------------------------+
|                            1 |
+------------------------------+

mysql> SET @ls3 = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';

mysql> SELECT IsClosed(GeomFromText(@ls3));
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+-----------------------------+
| IsClosed(GeomFromText(@ls3)) |
+-----------------------------+
|                           0 |
+-----------------------------+
```

- `NumPoints(ls)`

  Returns the number of `Point` objects in the `LineString` value `ls`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----------------------------+
| NumPoints(GeomFromText(@ls)) |
+-----------------------------+
|                           3 |
+-----------------------------+
```

- `PointN(ls,N)`

  Returns the `N`-th `Point` in the `Linestring` value `ls`. Points are numbered beginning with 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-------------------------------------+
| AsText(PointN(GeomFromText(@ls),2)) |
+-------------------------------------+
| POINT(2 2)                          |
+-------------------------------------+
```

- `StartPoint(ls)`

  Returns the `Point` that is the start point of the `LineString` value `ls`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+---------------------------------------+
| AsText(StartPoint(GeomFromText(@ls))) |
+---------------------------------------+
| POINT(1 1)                            |
+---------------------------------------+
```

12.14.7.4 Polygon and MultiPolygon Property Functions

These functions return properties of `Polygon` or `MultiPolygon` values.

- `Area(poly)`

  Returns a double-precision number indicating the area of the argument, as measured in its spatial reference system. For arguments of dimension 0 or 1, the result is 0.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
|                         4 |
+---------------------------+

mysql> SET @mpoly =
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    -> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+---------------------------+
| Area(GeomFromText(@mpoly)) |
+---------------------------+
|                         8 |
+---------------------------+
```

- `Centroid(mpoly)`

  Returns the mathematical centroid for the `MultiPolygon` value *`mpoly`* as a `Point`. The result is not guaranteed to be on the `MultiPolygon`.

```
mysql> SET @poly =
    -> GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7,5 5))');
mysql> SELECT GeometryType(@poly),AsText(Centroid(@poly));
+--------------------+-----------------------------------------+
| GeometryType(@poly) | AsText(Centroid(@poly))                 |
+--------------------+-----------------------------------------+
| POLYGON            | POINT(4.958333333333333 4.958333333333333) |
+--------------------+-----------------------------------------+
```

- `ExteriorRing(poly)`

  Returns the exterior ring of the `Polygon` value *`poly`* as a `LineString`.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-------------------------------------------+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+-------------------------------------------+
```

- `InteriorRingN(poly,N)`

  Returns the *`N`*-th interior ring for the `Polygon` value *`poly`* as a `LineString`. Rings are numbered beginning with 1.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----------------------------------------------+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)               |
+-----------------------------------------------+
```

- `NumInteriorRings(poly)`

  Returns the number of interior rings in the `Polygon` value *`poly`*.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+--------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

12.14.7.5 GeometryCollection Property Functions

These functions return properties of `GeometryCollection` values.

- `GeometryN(gc,N)`

  Returns the *N*-th geometry in the `GeometryCollection` value *gc*. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+--------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+--------------------------------------+
| POINT(1 1)                           |
+--------------------------------------+
```

- `NumGeometries(gc)`

  Returns the number of geometries in the `GeometryCollection` value *gc*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+----------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+----------------------------------+
|                                2 |
+----------------------------------+
```

12.14.8 Spatial Operator Functions

Section 12.14.7, “Geometry Property Functions”, discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- `Envelope(g)`
- `StartPoint(ls)`
- `EndPoint(ls)`
- `PointN(ls,N)`
- `ExteriorRing(poly)`
- `InteriorRingN(poly,N)`
- `GeometryN(gc,N)`

12.14.9 Functions That Test Spatial Relations Between Geometry Objects

The functions described in this section take two geometries as arguments and return a qualitative or quantitative relation between them.

MySQL implements two sets of functions using function names defined by the OpenGIS specification. One set tests the relationship between two geometry values using precise object shapes, the other set uses object minimum bounding rectangles (MBRs).

There is also a MySQL-specific set of MBR-based functions available to test the relationship between two geometry values.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

12.14.9.1 Spatial Relation Functions That Use Object Shapes

The OpenGIS specification defines the following functions. They test the relationship between two geometry values `g1` and `g2`, using precise object shapes. The return values 1 and 0 indicate true and false, respectively.

- `Crosses(g1,g2)`

  Returns 1 if *`g1`* spatially crosses *`g2`*. Returns `NULL` if `g1` is a `Polygon` or a `MultiPolygon`, or if *`g2`* is a `Point` or a `MultiPoint`. Otherwise, returns 0.

  The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

  - The two geometries intersect
  - Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
  - Their intersection is not equal to either of the two given geometries

- `Touches(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially touches *`g2`*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

12.14.9.2 Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)

The OpenGIS specification defines the following functions that test the relationship between two geometry values `g1` and `g2`. The MySQL implementation uses minimum bounding rectangles, so these functions return the same result as the corresponding MBR-based functions. The return values 1 and 0 indicate true and false, respectively.

- `Contains(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* completely contains *`g2`*. This tests the opposite relationship as `Within()`.

- `Disjoint(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially disjoint from (does not intersect) *`g2`*.

- `Equals(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially equal to *`g2`*.

- `Intersects(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially intersects *`g2`*.

- `Overlaps(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially overlaps *`g2`*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Within(g1,g2)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Returns 1 or 0 to indicate whether *g1* is spatially within *g2*. This tests the opposite relationship as `Contains()`.

12.14.9.3 MySQL-Specific Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)

MySQL provides several MySQL-specific functions that test relations between minimum bounding rectangles of two geometries `g1` and `g2`. The return values 1 and 0 indicate true and false, respectively.

- `MBRContains(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* contains the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRWithin()`.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRWithin(@g2,@g1);
+---------------------+-------------------+
| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |
+---------------------+-------------------+
|                   1 |                 1 |
+---------------------+-------------------+
```

- `MBRDisjoint(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- `MBREqual(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are the same.

- `MBRIntersects(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* intersect.

- `MBROverlaps(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* overlap. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `MBRTouches(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* touch. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `MBRWithin(g1,g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is within the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRContains()`.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-------------------+-------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-------------------+-------------------+
|                 1 |                 0 |
+-------------------+-------------------+
```

12.15 Miscellaneous Functions

Table 12.19 Miscellaneous Functions

Name	Description
DEFAULT()	Return the default value for a table column
GET_LOCK()	Get a named lock
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
IS_FREE_LOCK()	Whether the named lock is free
IS_USED_LOCK()	Whether the named lock is in use; return connection identifier if true
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
NAME_CONST()	Causes the column to have the given name
RAND()	Return a random floating-point value
RELEASE_LOCK()	Releases the named lock
SLEEP()	Sleep for a number of seconds
UUID()	Return a Universal Unique Identifier (UUID)
VALUES()	Defines the values to be used during an INSERT

- `DEFAULT(col_name)`

  Returns the default value for a table column. Starting with MySQL 5.0.2, an error results if the column has no default value.

  ```
  mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
  ```

- `FORMAT(X,D)`

  Formats the number *`X`* to a format like `'#,###,###.##'`, rounded to *`D`* decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions”.

- `GET_LOCK(str,timeout)`

  Tries to obtain a lock with a name given by the string *`str`*, using a timeout of *`timeout`* seconds. Returns `1` if the lock was obtained successfully, `0` if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`).

  A lock obtained with `GET_LOCK()` is released explicitly by executing `RELEASE_LOCK()` or a new `GET_LOCK()`, or implicitly when your session terminates (either normally or abnormally).

  Locks obtained with `GET_LOCK()` are not released when transactions commit or roll back.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`GET_LOCK()` can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked within one session, `GET_LOCK()` blocks any request by another session for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *`db_name.str`* or *`app_name.str`*.

```
mysql> SELECT GET_LOCK('lock1',10);
        -> 1
mysql> SELECT IS_FREE_LOCK('lock2');
        -> 1
mysql> SELECT GET_LOCK('lock2',10);
        -> 1
mysql> SELECT RELEASE_LOCK('lock2');
        -> 1
mysql> SELECT RELEASE_LOCK('lock1');
        -> NULL
```

The second `RELEASE_LOCK()` call returns `NULL` because the lock `'lock1'` was automatically released by the second `GET_LOCK()` call.

If multiple clients are waiting for a lock, the order in which they will acquire it is undefined. Applications should not assume that clients will acquire the lock in the same order that they issued the lock requests.

Note

If a client attempts to acquire a lock that is already held by another client, it blocks according to the *`timeout`* argument. If the blocked client terminates, its thread does not die until the lock request times out. This is a known bug (fixed in MySQL 5.5).

- `INET_ATON(`*`expr`*`)`

  Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address in network byte order (big endian). `INET_ATON()` returns `NULL` if it does not understand its argument.

  ```
  mysql> SELECT INET_ATON('10.0.5.9');
          -> 167773449
  ```

  For this example, the return value is calculated as $10\times256^3 + 0\times256^2 + 5\times256 + 9$.

  `INET_ATON()` may or may not return a non-`NULL` result for short-form IP addresses (such as `'127.1'` as a representation of `'127.0.0.1'`). Because of this, `INET_ATON()`a should not be used for such addresses.

  Note

  To store values generated by `INET_ATON()`, use an `INT UNSIGNED` column rather than `INT`, which is signed. If you use a signed column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See Section 11.2.6, “Out-of-Range and Overflow Handling”.

- `INET_NTOA(`*`expr`*`)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Given a numeric IPv4 network address in network byte order, returns the dotted-quad representation of the address as a binary string. `INET_NTOA()` returns `NULL` if it does not understand its argument.

```
mysql> SELECT INET_NTOA(167773449);
       -> '10.0.5.9'
```

- `IS_FREE_LOCK(str)`

  Checks whether the lock named *`str`* is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

- `IS_USED_LOCK(str)`

  Checks whether the lock named *`str`* is in use (that is, locked). If so, it returns the connection identifier of the client session that holds the lock. Otherwise, it returns `NULL`.

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

  This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the slave SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

  If a *`timeout`* value is specified, `MASTER_POS_WAIT()` stops waiting when *`timeout`* seconds have elapsed. *`timeout`* must be greater than 0; a zero or negative *`timeout`* means no timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

- `NAME_CONST(name,value)`

  Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name. The arguments should be constants.

```
mysql> SELECT NAME_CONST('myname', 14);
+--------+
| myname |
+--------+
|     14 |
+--------+
```

  This function was added in MySQL 5.0.12. It is for internal use only. The server uses it when writing statements from stored programs that contain references to local program variables, as described in Section 18.6, “Binary Logging of Stored Programs”, You might see this function in the output from `mysqlbinlog`.

  For your applications, you can obtain exactly the same result as in the example just shown by using simple aliasing, like this:

```
mysql> SELECT 14 AS myname;
+--------+
| myname |
+--------+
|     14 |
+--------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
1 row in set (0.00 sec)
```

See Section 13.2.8, “SELECT Syntax”, for more information about column aliases.

- `RELEASE_LOCK(str)`

  Releases the lock named by the string *`str`* that was obtained with `GET_LOCK()`. Returns `1` if the lock was released, `0` if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

  The `DO` statement is convenient to use with `RELEASE_LOCK()`. See Section 13.2.3, “DO Syntax”.

- `SLEEP(duration)`

  Sleeps (pauses) for the number of seconds given by the *`duration`* argument, then returns 0. If `SLEEP()` is interrupted, it returns 1. The duration may have a fractional part. This function was added in MySQL 5.0.12.

- `UUID()`

  Returns a Universal Unique Identifier (UUID) generated according to RFC 4122, “A Universally Unique IDentifier (UUID) URN Namespace” (http://www.ietf.org/rfc/rfc4122.txt).

  A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate devices not connected to each other.

  Warning

  Although `UUID()` values are intended to be unique, they are not necessarily unguessable or unpredictable. If unpredictability is required, UUID values should be generated some other way.

  `UUID()` returns a value that conforms to UUID version 1 as described in RFC 4122. The value is a 128-bit number represented as a `utf8` string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

  - The first three numbers are generated from the low, middle, and high parts of a timestamp. The high part also includes the UUID version number.
  - The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
  - The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host device has no Ethernet card, or it is unknown how to find the hardware address of an interface on the host operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

    The MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
        -> '6ccd780c-baba-1026-9564-0040f4311e29'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

`UUID()` does not work with statement-based replication.

- `VALUES(col_name)`

  In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause of `INSERT` statements and returns `NULL` otherwise. See Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

  When sleep returns normally (without interruption), it returns 0:

```
mysql> SELECT SLEEP(1000);
+-------------+
| SLEEP(1000) |
+-------------+
|           0 |
+-------------+
```

  When `SLEEP()` is the only thing invoked by a query that is interrupted, it returns 1 and the query itself returns no error. This statement is interrupted using `KILL QUERY` from another session:

```
mysql> SELECT SLEEP(1000);
+-------------+
| SLEEP(1000) |
+-------------+
|           1 |
+-------------+
```

  When `SLEEP()` is only part of a query that is interrupted, the query returns an error. This statement is interrupted using `KILL QUERY` from another session:

```
mysql> SELECT 1 FROM t1 WHERE SLEEP(1000);
ERROR 1317 (70100): Query execution was interrupted
```

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

12.16 GROUP BY (Aggregate) Functions

12.16.1 GROUP BY (Aggregate) Function Descriptions

Table 12.20 Aggregate (`GROUP BY`) Functions

Name	Description
`AVG()`	Return the average value of the argument
`BIT_AND()`	Return bitwise AND
`BIT_OR()`	Return bitwise OR
`BIT_XOR()`	Return bitwise XOR

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Description
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
GROUP_CONCAT()	Return a concatenated string
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

This section describes group (aggregate) functions that operate on sets of values. Unless otherwise stated, group functions ignore NULL values.

If you use a group function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows. For more information, see Section 12.16.3, “MySQL Handling of GROUP BY”.

For numeric arguments, the variance and standard deviation functions return a DOUBLE value. The SUM() and AVG() functions return a DECIMAL value for exact-value arguments (integer or DECIMAL), and a DOUBLE value for approximate-value arguments (FLOAT or DOUBLE). (Before MySQL 5.0.3, SUM() and AVG() return DOUBLE for all numeric arguments.)

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as SUM() or AVG() that expect a numeric argument cast the argument to a number if necessary. For SET or ENUM values, the cast operation causes the underlying numeric value to be used.

The BIT_AND(), BIT_OR(), and BIT_XOR() aggregate functions perform bit operations. They require BIGINT (64-bit integer) arguments and return BIGINT values. Arguments of other types are converted to BIGINT and truncation might occur.

- AVG([DISTINCT] *expr*)

  Returns the average value of *expr*. The DISTINCT option can be used as of MySQL 5.0.3 to return the average of the distinct values of *expr*.

  AVG() returns NULL if there were no matching rows.

```
mysql> SELECT student_name, AVG(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `BIT_AND(expr)`

  Returns the bitwise `AND` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  `BIT_AND()` returns `18446744073709551615` if there were no matching rows. (This is the value of an unsigned `BIGINT` value with all bits set to 1.)

- `BIT_OR(expr)`

  Returns the bitwise `OR` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  `BIT_OR()` returns `0` if there were no matching rows.

- `BIT_XOR(expr)`

  Returns the bitwise `XOR` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  `BIT_XOR()` returns `0` if there were no matching rows.

- `COUNT(expr)`

  Returns a count of the number of non-`NULL` values of *`expr`* in the rows retrieved by a `SELECT` statement. The result is a `BIGINT` value.

  `COUNT()` returns `0` if there were no matching rows.

```
mysql> SELECT student.student_name,COUNT(*)
    ->        FROM student,course
    ->        WHERE student.student_id=course.student_id
    ->        GROUP BY student_name;
```

  `COUNT(*)` is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain `NULL` values.

  `COUNT(*)` is optimized to return very quickly if the `SELECT` retrieves from one table, no other columns are retrieved, and there is no `WHERE` clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

  This optimization applies only to `MyISAM` tables only, because an exact row count is stored for this storage engine and can be accessed very quickly. For transactional storage engines such as `InnoDB` and `BDB`, storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- `COUNT(DISTINCT expr,[expr...])`

  Returns a count of the number of rows with different non-`NULL` *`expr`* values.

  `COUNT(DISTINCT)` returns `0` if there were no matching rows.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

  In MySQL, you can obtain the number of distinct expression combinations that do not contain `NULL` by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside `COUNT(DISTINCT ...)`.

- `GROUP_CONCAT(expr)`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(test_score)
    ->     FROM student
    ->     GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(DISTINCT test_score
    ->                ORDER BY test_score DESC SEPARATOR ' ')
    ->     FROM student
    ->     GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. To eliminate duplicate values, use the DISTINCT clause. To sort values in the result, use the ORDER BY clause. To sort in reverse order, add the DESC (descending) keyword to the name of the column you are sorting by in the ORDER BY clause. The default is ascending order; this may be specified explicitly using the ASC keyword. The default separator between values in a group is comma (“,”). To specify a separator explicitly, use SEPARATOR followed by the string literal value that should be inserted between group values. To eliminate the separator altogether, specify SEPARATOR ''.

The result is truncated to the maximum length that is given by the group_concat_max_len system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of max_allowed_packet. The syntax to change the value of group_concat_max_len at runtime is as follows, where *val* is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

The return value is a nonbinary or binary string, depending on whether the arguments are nonbinary or binary strings. The result type is TEXT or BLOB unless group_concat_max_len is less than or equal to 512, in which case the result type is VARCHAR or VARBINARY. (Prior to MySQL 5.0.19, GROUP_CONCAT() returned TEXT or BLOB group_concat_max_len greater than 512 only if the query included an ORDER BY clause.)

See also CONCAT() and CONCAT_WS(): Section 12.5, “String Functions”.

- MAX([DISTINCT] *expr*)

Returns the maximum value of *expr*. MAX() may take a string argument; in such cases, it returns the maximum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The DISTINCT keyword can be used to find the maximum of the distinct values of *expr*, however, this produces the same result as omitting DISTINCT.

MAX() returns NULL if there were no matching rows.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

For `MAX()`, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them. This is expected to be rectified in a future MySQL release.

- `MIN([DISTINCT] expr)`

  Returns the minimum value of *`expr`*. `MIN()` may take a string argument; in such cases, it returns the minimum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The `DISTINCT` keyword can be used to find the minimum of the distinct values of *`expr`*, however, this produces the same result as omitting `DISTINCT`.

  `MIN()` returns `NULL` if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

  For `MIN()`, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them. This is expected to be rectified in a future MySQL release.

- `STD(expr)`

  Returns the population standard deviation of *`expr`*. This is an extension to standard SQL. As of MySQL 5.0.3, the standard SQL function `STDDEV_POP()` can be used instead.

  `STD()` returns `NULL` if there were no matching rows.

- `STDDEV(expr)`

  Returns the population standard deviation of *`expr`*. This function is provided for compatibility with Oracle. As of MySQL 5.0.3, the standard SQL function `STDDEV_POP()` can be used instead.

  `STDDEV()` returns `NULL` if there were no matching rows.

- `STDDEV_POP(expr)`

  Returns the population standard deviation of *`expr`* (the square root of `VAR_POP()`). This function was added in MySQL 5.0.3. Before 5.0.3, you can use `STD()` or `STDDEV()`, which are equivalent but not standard SQL.

  `STDDEV_POP()` returns `NULL` if there were no matching rows.

- `STDDEV_SAMP(expr)`

  Returns the sample standard deviation of *`expr`* (the square root of `VAR_SAMP()`. This function was added in MySQL 5.0.3.

  `STDDEV_SAMP()` returns `NULL` if there were no matching rows.

- `SUM([DISTINCT] expr)`

  Returns the sum of *`expr`*. If the return set has no rows, `SUM()` returns `NULL`. The `DISTINCT` keyword can be used to sum only the distinct values of *`expr`*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`SUM()` returns `NULL` if there were no matching rows.

- `VAR_POP(expr)`

  Returns the population standard variance of *`expr`*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. This function was added in MySQL 5.0.3. Before 5.0.3, you can use `VARIANCE()`, which is equivalent but is not standard SQL.

  `VAR_POP()` returns `NULL` if there were no matching rows.

- `VAR_SAMP(expr)`

  Returns the sample variance of *`expr`*. That is, the denominator is the number of rows minus one. This function was added in MySQL 5.0.3.

  `VAR_SAMP()` returns `NULL` if there were no matching rows.

- `VARIANCE(expr)`

  Returns the population standard variance of *`expr`*. This is an extension to standard SQL. As of MySQL 5.0.3, the standard SQL function `VAR_POP()` can be used instead.

  `VARIANCE()` returns `NULL` if there were no matching rows.

12.16.2 GROUP BY Modifiers

The `GROUP BY` clause permits a `WITH ROLLUP` modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. `ROLLUP` thus enables you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a table named `sales` has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

The table's contents can be summarized per year with a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause causes the query to produce another row that shows the grand total over all year values:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

The grand total super-aggregate line is identified by the value `NULL` in the `year` column.

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a “break” (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary on the `sales` table based on `year`, `country`, and `product` might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2001 | Finland | Phone      |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
+------+---------+------------+-------------+
```

The output indicates summary values only at the year/country/product level of analysis. When `ROLLUP` is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | India   | NULL       |        1350 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2000 | USA     | NULL       |        1575 |
| 2000 | NULL    | NULL       |        4525 |
| 2001 | Finland | Phone      |          10 |
| 2001 | Finland | NULL       |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
| 2001 | USA     | NULL       |        3000 |
| 2001 | NULL    | NULL       |        3010 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

For this query, adding `ROLLUP` causes the output to include summary information at four levels of analysis, not just one. Here is how to interpret the `ROLLUP` output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the `product` column set to `NULL`.
- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the `country` and `products` columns set to `NULL`.
- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the `year`, `country`, and `products` columns set to `NULL`.

Other Considerations When using ROLLUP

The following items list some behaviors specific to the MySQL implementation of `ROLLUP`.

When you use `ROLLUP`, you cannot also use an `ORDER BY` clause to sort the results. In other words, `ROLLUP` and `ORDER BY` are mutually exclusive. However, you still have some control over sort order. `GROUP BY` in MySQL sorts results, and you can use explicit `ASC` and `DESC` keywords with columns named in the `GROUP BY` list to specify sort order for individual columns. (The higher-level summary rows added by `ROLLUP` still appear after the rows from which they are calculated, regardless of the sort order.)

`LIMIT` can be used to restrict the number of rows returned to the client. `LIMIT` is applied after `ROLLUP`, so the limit applies against the extra rows added by `ROLLUP`. For example:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP
    -> LIMIT 5;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
+------+---------+------------+-------------+
```

Using `LIMIT` with `ROLLUP` may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The `NULL` indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the `GROUP BY` clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to `NULL`. (If you specify grouping columns by column number, the server identifies which columns to set to `NULL` by number.)

Because the `NULL` values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as `NULL` values within the query itself. For example, you cannot add `HAVING product IS NULL` to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the `NULL` values do appear as `NULL` on the client side and can be tested as such using any MySQL client programming interface.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL permits a column that does not appear in the `GROUP BY` list to be named in the select list. In this case, the server is free to choose any value from this nonaggregated column in summary rows, and this includes the extra rows added by `WITH ROLLUP`. For example, in the following query, `country` is a nonaggregated column that does not appear in the `GROUP BY` list and values chosen for this column are indeterminate:

```
mysql> SELECT year, country, SUM(profit)
    -> FROM sales GROUP BY year WITH ROLLUP;
+------+---------+-------------+
| year | country | SUM(profit) |
+------+---------+-------------+
| 2000 | India   |        4525 |
| 2001 | USA     |        3010 |
| NULL | USA     |        7535 |
+------+---------+-------------+
```

This behavior occurs if the `ONLY_FULL_GROUP_BY` SQL mode is not enabled. If that mode is enabled, the server rejects the query as illegal because `country` is not listed in the `GROUP BY` clause. For more information about nonaggregated columns and `GROUP BY`, see Section 12.16.3, “MySQL Handling of GROUP BY”.

12.16.3 MySQL Handling of GROUP BY

In standard SQL, a query that includes a `GROUP BY` clause cannot refer to nonaggregated columns in the select list that are not named in the `GROUP BY` clause. For example, this query is illegal in standard SQL because the nonaggregated `name` column in the select list does not appear in the `GROUP BY`:

```
SELECT o.custid, c.name, MAX(o.payment)
  FROM orders AS o, customers AS c
  WHERE o.custid = c.custid
  GROUP BY o.custid;
```

For the query to be legal, the `name` column must be omitted from the select list or named in the `GROUP BY` clause.

MySQL extends the standard SQL use of `GROUP BY` so that the select list can refer to nonaggregated columns not named in the `GROUP BY` clause. This means that the preceding query is legal in MySQL. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. However, this is useful primarily when all values in each nonaggregated column not named in the `GROUP BY` are the same for each group. The server is free to choose any value from each group, so unless they are the same, the values chosen are indeterminate. Furthermore, the selection of values from each group cannot be influenced by adding an `ORDER BY` clause. Result set sorting occurs after values have been chosen, and `ORDER BY` does not affect which values within each group the server chooses.

A similar MySQL extension applies to the `HAVING` clause. In standard SQL, a query cannot refer to nonaggregated columns in the `HAVING` clause that are not named in the `GROUP BY` clause. To simplify calculations, a MySQL extension permits references to such columns. This extension assumes that the nongrouped columns have the same group-wise values. Otherwise, the result is indeterminate.

To disable the MySQL `GROUP BY` extension and enable standard SQL behavior, enable the `ONLY_FULL_GROUP_BY` SQL mode. In this case, columns not named in the `GROUP BY` clause cannot be used in the select list or `HAVING` clause unless enclosed in an aggregate function.

The select list extension also applies to `ORDER BY`. That is, you can refer to nonaggregated columns in the `ORDER BY` clause that do not appear in the `GROUP BY` clause. (However, as mentioned previously, `ORDER BY` does not affect which values are chosen from nonaggregated columns; it only sorts them after they have been chosen.) This extension does not apply if the `ONLY_FULL_GROUP_BY` SQL mode is enabled.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If a query has aggregate functions and no `GROUP BY` clause, it cannot have nonaggregated columns in the select list, `HAVING` condition, or `ORDER BY` list with `ONLY_FULL_GROUP_BY` enabled:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

Without `GROUP BY`, there is a single group and it is indeterminate which `name` value to choose for the group.

Another MySQL extension to standard SQL permits references in the `HAVING` clause to aliased expressions in the select list. Enabling `ONLY_FULL_GROUP_BY` prevents this. For example, the following query returns `name` values that occur only once in table `orders`; the query is accepted regardless of whether `ONLY_FULL_GROUP_BY` is enabled:

```
SELECT name, COUNT(name) FROM orders
  GROUP BY name
  HAVING COUNT(name) = 1;
```

The following query is accepted only if `ONLY_FULL_GROUP_BY` is disabled.

```
SELECT name, COUNT(name) AS c FROM orders
  GROUP BY name
  HAVING c = 1;
```

In some cases, you can use `MIN()` and `MAX()` to obtain a specific column value even if it is not unique. If the `sort` column contains integers no larger than 6 digits, the following query gives the value of `column` from the row containing the smallest `sort` value:

```
SUBSTR(MIN(CONCAT(LPAD(sort,6,'0'),column)),7)
```

See Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Column”.

If you are trying to follow standard SQL, you cannot use expressions in `GROUP BY` clauses. As a workaround, use an alias for the expression:

```
SELECT id, FLOOR(value/100) AS val
  FROM tbl_name
  GROUP BY id, val;
```

MySQL permits expressions in `GROUP BY` clauses, so the alias is unnecessary:

```
SELECT id, FLOOR(value/100)
  FROM tbl_name
  GROUP BY id, FLOOR(value/100);
```

12.17 Precision Math

MySQL 5.0 introduces precision math: numeric value handling that results in more accurate results and more control over invalid values than in earlier versions of MySQL. Precision math is based on two implementation changes:

- The introduction of SQL modes in MySQL 5.0 that control how strict the server is about accepting or rejecting invalid data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The introduction in MySQL 5.0.3 of a library for fixed-point arithmetic.

These changes result in the following characteristics for numeric operations and provide improved compliance with standard SQL:

- **Precise calculations**: For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, MySQL treats a number such as `.0001` as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly `1`, not a value that is merely “close” to 1.
- **Well-defined rounding behavior**: For exact-value numbers, the result of `ROUND()` depends on its argument, not on environmental factors such as how the underlying C library works.
- **Platform independence**: Operations on exact numeric values are the same across different platforms such as Windows and Unix.
- **Control over handling of invalid values**: Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you can treat division by zero as an error rather than as an operation that produces a result of `NULL`. The choice of which approach to take is determined by the setting of the server SQL mode.

The following discussion covers several aspects of how precision math works, including possible incompatibilities with older applications. At the end, some examples are given that demonstrate how MySQL handles numeric operations precisely. For information about controlling the SQL mode, see Section 5.1.7, “Server SQL Modes”.

12.17.1 Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (integer and `DECIMAL` types) and exact-value numeric literals. Approximate-value data types and numeric literals are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

12.17.2 DECIMAL Data Type Characteristics

This section discusses the characteristics of the `DECIMAL` data type (and its synonyms) as of MySQL 5.0.3, with particular regard to the following topics:

- Maximum number of digits
- Storage format

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Storage requirements
- The nonstandard MySQL extension to the upper range of `DECIMAL` columns

Some of these characteristics result in possible incompatibilities for applications that are written for older versions of MySQL. These incompatibilities are noted throughout this section.

The declaration syntax for a `DECIMAL` column remains `DECIMAL(M,D)`, although the range of values for the arguments has changed somewhat:

- *`M`* is the maximum number of digits (the precision). It has a range of 1 to 65. This introduces a possible incompatibility for older applications, because previous versions of MySQL permit a range of 1 to 254. (The precision of 65 digits actually applies as of MySQL 5.0.6. From 5.0.3 to 5.0.5, the precision is 64 digits.)
- *`D`* is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than *`M`*.

The maximum value of 65 for *`M`* means that calculations on `DECIMAL` values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals differs from before. (Prior to MySQL 5.0.3, decimal values could have up to 254 digits. However, calculations were done using floating-point and thus were approximate, not exact.) This change in the range of literal values is another possible source of incompatibility for older applications.

Values for `DECIMAL` columns no longer are represented as strings that require 1 byte per digit or sign character. Instead, a binary format is used that packs nine decimal digits into 4 bytes. This change to `DECIMAL` storage format changes the storage requirements as well. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires 4 bytes, and any remaining digits require some fraction of 4 bytes. The storage required for remaining digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1–2	1
3–4	2
5–6	3
7–9	4

For example, a `DECIMAL(18,9)` column has nine digits on either side of the decimal point, so the integer part and the fractional part each require 4 bytes. A `DECIMAL(20,6)` column has fourteen integer digits and six fractional digits. The integer digits require four bytes for nine of the digits and 3 bytes for the remaining five digits. The six fractional digits require 3 bytes.

As a result of the change from string to numeric format for `DECIMAL` storage, `DECIMAL` columns no longer store a leading `+` or `-` character or leading `0` digits. Before MySQL 5.0.3, if you inserted `+0003.1` into a `DECIMAL(5,1)` column, it was stored as `+0003.1`. As of MySQL 5.0.3, it is stored as `3.1`. For negative numbers, a literal `-` character is no longer stored. Applications that rely on the older behavior must be modified to account for this change.

The change of storage format also means that `DECIMAL` columns no longer support the nonstandard extension that permitted values larger than the range implied by the column definition. Formerly, 1 byte was allocated for storing the sign character. For positive values that needed no sign byte, MySQL permitted an extra digit to be stored instead. For example, a `DECIMAL(3,0)` column must support a range of at least `-999` to `999`, but MySQL would permit storing values from `1000` to `9999` as well, by using

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the sign byte to store an extra digit. This extension to the upper range of DECIMAL columns is no longer supported. As of MySQL 5.0.3, a DECIMAL(M,D) column permits at most M - D digits to the left of the decimal point. This can result in an incompatibility if an application has a reliance on MySQL permitting "too-large" values.

The SQL standard requires that the precision of NUMERIC(M,D) be *exactly* M digits. For DECIMAL(M,D), the standard requires a precision of at least M digits but permits more. In MySQL, DECIMAL(M,D) and NUMERIC(M,D) are the same, and both have a precision of exactly M digits.

Summary of incompatibilities:

The following list summarizes the incompatibilities that result from changes to DECIMAL column and value handling. You can use it as guide when porting older applications for use with MySQL 5.0.3 and up.

- For DECIMAL(M,D), the maximum M is 65, not 254.
- Calculations involving exact-value decimal numbers are accurate to 65 digits. This is fewer than the maximum number of digits permitted before MySQL 5.0.3 (254 digits), but the exact-value precision is greater. Calculations formerly were done with double-precision floating-point, which has a precision of 52 bits (about 15 decimal digits).
- The nonstandard MySQL extension to the upper range of DECIMAL columns is no longer supported.
- Leading "+" and "0" characters are not stored.

The behavior used by the server for DECIMAL columns in a table depends on the version of MySQL used to create the table. If your server is from MySQL 5.0.3 or higher, but a table that was created before 5.0.3 has DECIMAL columns, the old behavior still applies to those columns. To convert the table to the newer DECIMAL format, dump it with mysqldump and reload it.

For a full explanation of the internal format of DECIMAL values, see the file strings/decimal.c in a MySQL source distribution. The format is explained (with an example) in the decimal2bin() function.

12.17.3 Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for INSERT into a column with an exact data type (DECIMAL or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (If strict SQL mode is not enabled, truncation for INSERT is permissible.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.
- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using DECIMAL exact arithmetic and has a precision of 65 digits. The term "exact" is subject to the limits of what can be represented in binary. For example, 1.0/3.0 can be approximated in decimal notation as .333..., but not written as an exact number, so (1.0/3.0)*3.0 does not evaluate to exactly 1.0.
- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as BIGINT (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Inserts into numeric columns are affected by the SQL mode, which is controlled by the `sql_mode` system variable. (See Section 5.1.7, “Server SQL Modes”.) The following discussion mentions strict mode (selected by the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` mode values) and `ERROR_FOR_DIVISION_BY_ZERO`. To turn on all restrictions, you can simply use `TRADITIONAL` mode, which includes both strict mode values and `ERROR_FOR_DIVISION_BY_ZERO`:

```
mysql> SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column (`DECIMAL` or integer), it is inserted with its exact value if it is within the column range.

If the value has too many digits in the fractional part, rounding occurs and a warning is generated. Rounding is done as described in Section 12.17.4, “Rounding Behavior”.

If the value has too many digits in the integer part, it is too large and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

Underflow is not detected, so underflow handling is undefined.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has nonnumeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. This includes the empty string.
- A string that begins with a number can be converted, but the trailing nonnumeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

By default, division by zero produces a result of `NULL` and no warning. By setting the SQL mode appropriately, division by zero can be restricted.

With the `ERROR_FOR_DIVISION_BY_ZERO` SQL mode enabled, MySQL handles division by zero differently:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires `ERROR_FOR_DIVISION_BY_ZERO` in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and `ERROR_FOR_DIVISION_BY_ZERO` modes.

`sql_mode` Value	Result
`''` (Default)	No warning, no error; `i` is set to `NULL`.
strict	No warning, no error; `i` is set to `NULL`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

sql_mode Value	Result
ERROR_FOR_DIVISION_BY_ZERO	Warning, no error; i is set to NULL.
strict,ERROR_FOR_DIVISION_BY_ZERO	Error condition; no row is inserted.

12.17.4 Rounding Behavior

This section discusses precision math rounding for the ROUND() function and for inserts into columns with exact-value types (DECIMAL and integer).

The ROUND() function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, ROUND() uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that ROUND() uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For inserts into a DECIMAL or integer column, the target is an exact data type, so rounding uses “round half away from zero,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 2

mysql> SELECT d FROM t;
+------+
| d    |
+------+
| 3    |
| 3    |
+------+
```

12.17.5 Precision Math Examples

This section provides some examples that show how precision math improves query results in MySQL 5.0 compared to older versions. These examples demonstrate the principles described in Section 12.17.3, “Expression Handling”, and Section 12.17.4, “Rounding Behavior”.

Example 1. Numbers are used with their exact value as given when possible.

Before MySQL 5.0.3, numbers that are treated as floating-point values produce inexact results:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT (.1 + .2) = .3;
+----------------+
| (.1 + .2) = .3 |
+----------------+
|              1 |
+----------------+
```

As of MySQL 5.0.3, numbers are used as given when possible:

```
mysql> SELECT (.1 + .2) = .3;
+----------------+
| (.1 + .2) = .3 |
+----------------+
|              1 |
+----------------+
```

For floating-point values, results are inexact:

```
mysql> SELECT (.1E0 + .2E0) = .3E0;
+----------------------+
| (.1E0 + .2E0) = .3E0 |
+----------------------+
|                    0 |
+----------------------+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds `.0001` to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

The sum for both `d` and `f` logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```
+--------+------------------+
| d      | f                |
+--------+------------------+
| 1.0000 | 0.99999999999991 |
+--------+------------------+
```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers *`X1`* and *`X2`* that have scale *`S1`* and *`S2`*, the scale of the result is *`S1 + S2`*:

Before MySQL 5.0.3, this is what happens:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
```


This documentation is for an older version. If you're

```
|      0.00 |
+-----------+
```

The displayed value is incorrect. The value was calculated correctly in this case, but not displayed to the required scale. To see that the calculated value actually was .0001, try this:

```
mysql> SELECT .01 * .01 + .0000;
+-------------------+
| .01 * .01 + .0000 |
+-------------------+
|            0.0001 |
+-------------------+
```

As of MySQL 5.0.3, the displayed scale is correct:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
| 0.0001    |
+-----------+
```

Example 3. Rounding behavior for exact-value numbers is well-defined.

Before MySQL 5.0.3, rounding behavior (for example, with the `ROUND()` function) is dependent on the implementation of the underlying C library. This results in inconsistencies from platform to platform. For example, you might get a different value on Windows than on Linux, or a different value on x86 machines than on PowerPC machines.

As of MySQL 5.0.3, rounding happens like this:

- Rounding for exact-value columns (`DECIMAL` and integer) and exact-valued numbers uses the “round half away from zero” rule. Values with a fractional part of .5 or greater are rounded away from zero to the nearest integer, as shown here:

  ```
  mysql> SELECT ROUND(2.5), ROUND(-2.5);
  +------------+-------------+
  | ROUND(2.5) | ROUND(-2.5) |
  +------------+-------------+
  | 3          | -3          |
  +------------+-------------+
  ```

- Rounding for floating-point values uses the C library, which on many systems uses the “round to nearest even” rule. Values with any fractional part on such systems are rounded to the nearest even integer:

  ```
  mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
  +--------------+---------------+
  | ROUND(2.5E0) | ROUND(-2.5E0) |
  +--------------+---------------+
  |            2 |            -2 |
  +--------------+---------------+
  ```

Example 4. In strict mode, inserting a value that is out of range for a column causes an error, rather than truncation to a legal value.

Before MySQL 5.0.2 (or in 5.0.2 and later, without strict mode), truncation to a legal value occurs:

```
mysql> CREATE TABLE t (i TINYINT);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

As of MySQL 5.0.2, an error occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` set, division by zero causes an error, not a result of `NULL`.

Before MySQL 5.0.2 (or when not using strict mode in 5.0.2 or a later version), division by zero has a result of `NULL`:

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
| NULL |
+------+
1 row in set (0.00 sec)
```

As of MySQL 5.0.2, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

Example 6. Exact-value literals are evaluated as exact values.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Prior to MySQL 5.0.3, exact-value and approximate-value literals both are evaluated as double-precision floating-point values:

```
mysql> SELECT VERSION();
+------------+
| VERSION()  |
+------------+
| 4.1.18-log |
+------------+
1 row in set (0.01 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.07 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | double(3,1) |      |     | 0.0     |       |
| b     | double      |      |     | 0       |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.04 sec)
```

As of MySQL 5.0.3, the approximate-value literal is evaluated using floating point, but the exact-value literal is handled as `DECIMAL`:

```
mysql> SELECT VERSION();
+------------+
| VERSION()  |
+------------+
| 5.0.19-log |
+------------+
1 row in set (0.17 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.19 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| a     | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b     | double               | NO   |     | 0       |       |
+-------+----------------------+------+-----+---------+-------+
2 rows in set (0.02 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

Before MySQL 5.0.3, the result is a double no matter the argument type:

```
mysql> DESCRIBE y;
+--------+--------------+------+-----+---------+-------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| AVG(i) | double(17,4) | YES  |     | NULL    |       |
| AVG(d) | double(17,4) | YES  |     | NULL    |       |
| AVG(f) | double       | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

As of MySQL 5.0.3, the result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type:

```
mysql> DESCRIBE y;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| AVG(i) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

From MySQL 5.0.3 to 5.0.6, the first two columns are `DECIMAL(64,0)`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 13 SQL Statement Syntax

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This chapter describes the syntax for the SQL statements supported by MySQL.

13.1 Data Definition Statements

13.1.1 ALTER DATABASE Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. To use `ALTER DATABASE`, you need the `ALTER` privilege on the database. `ALTER SCHEMA` is a synonym for `ALTER DATABASE` as of MySQL 5.0.2.

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. Section 10.1, “Character Set Support”, discusses character set and collation names.

You can see what character sets and collations are available using, respectively, the `SHOW CHARACTER SET` and `SHOW COLLATION` statements. See Section 13.7.5.3, “SHOW CHARACTER SET Syntax”, and Section 13.7.5.4, “SHOW COLLATION Syntax”, for more information.

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)

The database name can be omitted, in which case the statement applies to the default database.

13.1.2 ALTER FUNCTION Syntax

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an `ALTER FUNCTION` statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using `DROP FUNCTION` and `CREATE FUNCTION`.

As of MySQL 5.0.3, you must have the `ALTER ROUTINE` privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the `ALTER FUNCTION` statement might also require the `SUPER` privilege, as described in Section 18.6, “Binary Logging of Stored Programs”.

13.1.3 ALTER PROCEDURE Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored procedure. More than one change may be specified in an `ALTER PROCEDURE` statement. However, you cannot change the parameters or body of a stored procedure using this statement; to make such changes, you must drop and re-create the procedure using `DROP PROCEDURE` and `CREATE PROCEDURE`.

As of MySQL 5.0.3, you must have the `ALTER ROUTINE` privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. This behavior can be changed by disabling the `automatic_sp_privileges` system variable. See Section 18.2.2, “Stored Routines and MySQL Privileges”.

13.1.4 ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...]

alter_specification:
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name ]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX|KEY} [index_name]
        [index_type] (index_col_name,...) [index_type]
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (index_col_name,...) [index_type]
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX|KEY] [index_name]
        [index_type] (index_col_name,...) [index_type]
  | ADD [FULLTEXT|SPATIAL] [INDEX|KEY] [index_name]
        (index_col_name,...) [index_type]
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        reference_definition
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP {INDEX|KEY} index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO|AS] new_tbl_name
  | ORDER BY col_name [, col_name] ...
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

table_options:
    table_option [[,] table_option] ...  (see CREATE TABLE options)
```

`ALTER TABLE` changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

Following the table name, specify the alterations to be made. If none are given, `ALTER TABLE` does nothing.

The syntax for many of the permissible alterations is similar to clauses of the `CREATE TABLE` statement. See Section 13.1.10, “CREATE TABLE Syntax”, for more information.

`table_options` signifies table options of the kind that can be used in the `CREATE TABLE` statement, such as `ENGINE`, `AUTO_INCREMENT`, `AVG_ROW_LENGTH`, `MAX_ROWS`, or `ROW_FORMAT`. For a list of all table options and a description of each, see Section 13.1.10, “CREATE TABLE Syntax”. However, `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. These warnings can be displayed with `SHOW WARNINGS`. See Section 13.7.5.37, “SHOW WARNINGS Syntax”.

If you use `ALTER TABLE` to change a column specification but `DESCRIBE` *`tbl_name`* indicates that your column was not changed, it is possible that MySQL ignored your modification for one of the reasons described in Section 13.1.10.4, “Silent Column Specification Changes”.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. These warnings can be displayed with `SHOW WARNINGS`. See Section 13.7.5.37, “SHOW WARNINGS Syntax”.

For information on troubleshooting `ALTER TABLE`, see Section B.5.6.1, “Problems with ALTER TABLE”.

Storage, Performance, and Concurrency Considerations

In most cases, `ALTER TABLE` makes a temporary copy of the original table. MySQL waits for other operations that are modifying the table, then proceeds. It incorporates the alteration into the copy, deletes the original table, and renames the new one. While `ALTER TABLE` is executing, the original table is readable by other sessions. Updates and writes to the table that begin after the `ALTER TABLE` operation

This documentation is for an older version. If you're
This documentation is for an older version. If you're

begins are stalled until the new table is ready, then are automatically redirected to the new table without any failed updates. The temporary copy of the original table is created in the database directory of the new table. This can differ from the database directory of the original table for `ALTER TABLE` operations that rename the table to a different database.

If you use `ALTER TABLE tbl_name RENAME TO new_tbl_name` without any other options, MySQL simply renames any files that correspond to the table `tbl_name` without making a copy. (You can also use the `RENAME TABLE` statement to rename tables. See Section 13.1.20, “RENAME TABLE Syntax”.) Any privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

If you use any option to `ALTER TABLE` other than `RENAME`, MySQL always creates a temporary table, even if the data wouldn't strictly need to be copied (such as when you change the name of a column). For `MyISAM` tables, you can speed up index re-creation (the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` system variable to a high value.

- To use `ALTER TABLE`, you need `ALTER`, `CREATE`, and `INSERT` privileges for the table. Renaming a table requires `ALTER` and `DROP` on the old table, `ALTER`, `CREATE`, and `INSERT` on the new table.
- `IGNORE` is a MySQL extension to standard SQL. It controls how `ALTER TABLE` works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If `IGNORE` is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If `IGNORE` is specified, only one row is used of rows with duplicates on a unique key. The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.
- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- *`table_options`* signifies table options of the kind that can be used in the `CREATE TABLE` statement, such as `ENGINE`, `AUTO_INCREMENT`, `AVG_ROW_LENGTH`, `MAX_ROWS`, or `ROW_FORMAT`. For a list of all table options and a description of each, see Section 13.1.10, “CREATE TABLE Syntax”. However, `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

  For example, to convert a table to be an `InnoDB` table, use this statement:

  ```
  ALTER TABLE t1 ENGINE = InnoDB;
  ```

  When you specify an `ENGINE` clause, `ALTER TABLE` rebuilds the table. This is true even if the table already has the specified storage engine.

  The outcome of attempting to change a table's storage engine is affected by whether the desired storage engine is available and the setting of the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in Section 5.1.7, “Server SQL Modes”.

  As of MySQL 5.0.23, to prevent inadvertent loss of data, `ALTER TABLE` cannot be used to change the storage engine of a table to `MERGE` or `BLACKHOLE`.

  To change the value of the `AUTO_INCREMENT` counter to be used for new rows, do this:

  ```
  ALTER TABLE t2 AUTO_INCREMENT = value;
  ```

  You cannot reset the counter to a value less than or equal to any that have already been used. For `MyISAM`, if the value is less than or equal to the maximum value currently in the `AUTO_INCREMENT` column, the value is reset to the current maximum plus one. For `InnoDB`, you can use `ALTER TABLE ... AUTO_INCREMENT = value` as of MySQL 5.0.3, but *if the value is less than the current maximum value in the column, no error occurs and the current sequence value is not changed.*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- You can issue multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses in a single `ALTER TABLE` statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per `ALTER TABLE` statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- `CHANGE col_name`, `DROP col_name`, and `DROP INDEX` are MySQL extensions to standard SQL.
- The word `COLUMN` is optional and can be omitted.
- *`column_definition`* clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. See Section 13.1.10, “CREATE TABLE Syntax”.
- You can rename a column using a `CHANGE old_col_name new_col_name column_definition` clause. To do so, specify the old and new column names and the definition that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
ALTER TABLE t1 CHANGE a b INTEGER;
```

  To change a column's type but not the name, `CHANGE` syntax still requires an old and new column name, even if they are the same. For example:

```
ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

  You can also use `MODIFY` to change a column's type without renaming it:

```
ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

  `MODIFY` is an extension to `ALTER TABLE` for Oracle compatibility.

  When you use `CHANGE` or `MODIFY`, *`column_definition`* must include the data type and all attributes that should apply to the new column, other than index attributes such as `PRIMARY KEY` or `UNIQUE`. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column `col1` is defined as `INT UNSIGNED DEFAULT 1 COMMENT 'my column'` and you modify the column as follows:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

  The resulting column will be defined as `BIGINT`, but will not include the attributes `UNSIGNED DEFAULT 1 COMMENT 'my column'`. To retain them, the statement should be:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

- When you change a data type using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.

Warning

This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated. To prevent the operation from succeeding if conversions to the new data type would result in loss of data,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

enable strict SQL mode before using `ALTER TABLE` (see Section 5.1.7, “Server SQL Modes”).

- To add a column at a specific position within a table row, use `FIRST` or `AFTER col_name`. The default is to add the column last. You can also use `FIRST` and `AFTER` in `CHANGE` or `MODIFY` operations to reorder columns within a table.

- `ALTER ... SET DEFAULT` or `ALTER ... DROP DEFAULT` specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be `NULL`, the new default is `NULL`. If the column cannot be `NULL`, MySQL assigns a default value as described in Section 11.6, “Data Type Default Values”.

- `DROP INDEX` removes an index. This is a MySQL extension to standard SQL. See Section 13.1.15, “DROP INDEX Syntax”. If you are unsure of the index name, use `SHOW INDEX FROM tbl_name`.

- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well. If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use `DROP TABLE` instead.

- `DROP PRIMARY KEY` drops the primary key. If there is no primary key, an error occurs.

  If you add a `UNIQUE INDEX` or `PRIMARY KEY` to a table, MySQL stores it before any nonunique index to permit detection of duplicate keys as early as possible.

- Some storage engines permit you to specify an index type when creating an index. The syntax for the *`index_type`* specifier is `USING type_name`. For details about `USING`, see Section 13.1.8, “CREATE INDEX Syntax”. Before MySQL 5.0.60, `USING` can be given only before the index column list. As of 5.0.60, the preferred position is after the column list. Support for use of the option before the column list will be removed in a future MySQL release.

  `index_option` values specify additional options for an index. `USING` is one such option. For details about permissible *`index_option`* values, see Section 13.1.8, “CREATE INDEX Syntax”.

- After an `ALTER TABLE` statement, it may be necessary to run `ANALYZE TABLE` to update index cardinality information. See Section 13.7.5.18, “SHOW INDEX Syntax”.

- `ORDER BY` enables you to create the new table with the rows in a specific order. This option is useful primarily when you know that you are mostly to query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

  Note

  The table does not remain in the specified order after inserts and deletes.

  `ORDER BY` syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by `ASC` or `DESC` to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted. This clause should be given last after any other clauses.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`ORDER BY` does not make sense for `InnoDB` tables because `InnoDB` always orders table rows according to the clustered index. The same is true for `BDB` tables that contain a user-defined `PRIMARY KEY`.

- If you use `ALTER TABLE` on a `MyISAM` table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). This should make `ALTER TABLE` much faster when you have many indexes.

  For `MyISAM` tables, key updating can be controlled explicitly. Use `ALTER TABLE ... DISABLE KEYS` to tell MySQL to stop updating nonunique indexes. Then use `ALTER TABLE ... ENABLE KEYS` to re-create missing indexes. `MyISAM` does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using `ALTER TABLE ... DISABLE KEYS` requires the `INDEX` privilege in addition to the privileges mentioned earlier.

  While the nonunique indexes are disabled, they are ignored for statements such as `SELECT` and `EXPLAIN` that otherwise would use them.

- If `ALTER TABLE` for an `InnoDB` table results in changes to column values (for example, because a column is truncated), `InnoDB`'s `FOREIGN KEY` constraint checks do not notice possible violations caused by changing the values.

- The `FOREIGN KEY` and `REFERENCES` clauses are supported by the `InnoDB` storage engine, which implements `ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (...) REFERENCES ... (...)`. See Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”. For other storage engines, the clauses are parsed but ignored. The `CHECK` clause is parsed but ignored by all storage engines. See Section 13.1.10, “CREATE TABLE Syntax”. The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See Section 1.8.2, “MySQL Differences from Standard SQL”.

  For `ALTER TABLE`, unlike `CREATE TABLE`, `ADD FOREIGN KEY` ignores *`index_name`* if given and uses an automatically generated foreign key name. As a workaround, include the `CONSTRAINT` clause to specify the foreign key name:

```
ADD CONSTRAINT name FOREIGN KEY (....) ...
```

Important

The inline `REFERENCES` specifications where the references are defined as part of the column specification are silently ignored by `InnoDB`. InnoDB only accepts `REFERENCES` clauses defined as part of a separate `FOREIGN KEY` specification.

- `InnoDB` supports the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

  For more information, see Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”.

- Adding and dropping a foreign key in separate clauses of a single `ALTER TABLE` statement may be problematic in some cases and is therefore unsupported. Use separate statements for each operation.

- For an `InnoDB` table that is created with its own tablespace in an `.ibd` file, that file can be discarded and imported. To discard the `.ibd` file, use this statement:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

This deletes the current `.ibd` file, so be sure that you have a backup first. Attempting to access the table while the tablespace file is discarded results in an error.

To import the backup `.ibd` file back into the table, copy it into the database directory, and then issue this statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

The tablespace file must have been created on the server into which it is imported later.

See Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”.

- To change the table default character set and all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name
CONVERT TO CHARACTER SET charset_name [COLLATE collation_name];
```

The statement also changes the collation of all character columns. If you specify no `COLLATE` clause to indicate which collation to use, the statement uses default collation for the character set. If this collation is inappropriate for the intended table use (for example, if it would change from a case-sensitive collation to a case-insensitive collation), specify a collation explicitly.

For a column that has a data type of `VARCHAR` or one of the `TEXT` types, `CONVERT TO CHARACTER SET` will change the data type as necessary to ensure that the new column is long enough to store as many characters as the original column. For example, a `TEXT` column has two length bytes, which store the byte-length of values in the column, up to a maximum of 65,535. For a `latin1 TEXT` column, each character requires a single byte, so the column can store up to 65,535 characters. If the column is converted to `utf8`, each character might require up to 3 bytes, for a maximum possible length of 3 × 65,535 = 196,605 bytes. That length will not fit in a `TEXT` column's length bytes, so MySQL will convert the data type to `MEDIUMTEXT`, which is the smallest string type for which the length bytes can record a value of 196,605. Similarly, a `VARCHAR` column might be converted to `MEDIUMTEXT`.

To avoid data type changes of the type just described, do not use `CONVERT TO CHARACTER SET`. Instead, use `MODIFY` to change individual columns. For example:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8;
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8;
```

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer will have a character set and a subsequent `CONVERT TO` operation will not apply to them.

If *`charset_name`* is `DEFAULT`, the database character set is used.

Warning

The `CONVERT TO` operation converts column values between the character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with `ALTER TABLE ... ADD column`).

When `foreign_key_checks` is enabled, which is the default setting, character set conversion is not permitted on tables that include a character string column used in a foreign key constraint. The workaround is to disable `foreign_key_checks` before performing the character set conversion. You must perform the conversion on both tables involved in the foreign key constraint before re-enabling `foreign_key_checks`. If you re-enable `foreign_key_checks` after converting only one of the tables, an `ON DELETE CASCADE` or `ON UPDATE CASCADE` operation could corrupt data in the referencing table due to implicit conversion that occurs during these operations (Bug #45290, Bug #74816).

With the `mysql_info()` C API function, you can find out how many rows were copied by `ALTER TABLE`, and (when `IGNORE` is used) how many rows were deleted due to duplication of unique key values. See Section 20.6.7.35, “mysql_info()”.

13.1.4.1 ALTER TABLE Examples

Begin with a table `t1` that is created as shown here:

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d` and a `UNIQUE` index on column `a`:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

To remove column `c`:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
  ADD PRIMARY KEY (c);
```

Note that we indexed `c` (as a `PRIMARY KEY`) because `AUTO_INCREMENT` columns must be indexed, and also that we declare `c` as `NOT NULL` because primary key columns cannot be `NULL`.

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers automatically. For `MyISAM` tables, you can set the first sequence number by executing `SET INSERT_ID=value` before `ALTER TABLE` or by using the `AUTO_INCREMENT=value` table option. See Section 5.1.4, “Server System Variables”.

With `MyISAM` tables, if you do not change the `AUTO_INCREMENT` column, the sequence number is not affected. If you drop an `AUTO_INCREMENT` column and then add another `AUTO_INCREMENT` column, the numbers are resequenced beginning with 1.

When replication is used, adding an `AUTO_INCREMENT` column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

This set of statements will also produce a new table `t2` identical to `t1`, with the addition of an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP TABLE t1;
ALTER TABLE t2 RENAME t1;
```

13.1.5 ALTER VIEW Syntax

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of a view, which must exist. The syntax is similar to that for `CREATE VIEW` and the effect is the same as for `CREATE OR REPLACE VIEW`. See Section 13.1.12, “CREATE VIEW Syntax”. This statement requires the `CREATE VIEW` and `DROP` privileges for the view, and some privilege for each column referred to in the `SELECT` statement. As of MySQL 5.0.52, `ALTER VIEW` is permitted only to the original definer or users with the `SUPER` privilege.

This statement was added in MySQL 5.0.1. The `DEFINER` and `SQL SECURITY` clauses may be used as of MySQL 5.0.16 to specify the security context to be used when checking access privileges at view invocation time. For details, see Section 13.1.12, “CREATE VIEW Syntax”.

13.1.6 CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

`CREATE DATABASE` creates a database with the given name. To use this statement, you need the `CREATE` privilege for the database. `CREATE SCHEMA` is a synonym for `CREATE DATABASE` as of MySQL 5.0.2.

An error occurs if the database exists and you did not specify `IF NOT EXISTS`.

`create_specification` options specify database characteristics. Database characteristics are stored in the `db.opt` file in the database directory. The `CHARACTER SET` clause specifies the default database character set. The `COLLATE` clause specifies the default database collation. Section 10.1, “Character Set Support”, discusses character set and collation names.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the `CREATE DATABASE` statement creates only a directory under the MySQL data directory and the `db.opt` file. Rules for permissible database names are given in Section 9.2, “Schema Object Names”.

If you manually create a directory under the data directory (for example, with `mkdir`), the server considers it a database directory and it shows up in the output of `SHOW DATABASES`.

You can also use the `mysqladmin` program to create databases. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

13.1.7 CREATE FUNCTION Syntax

The `CREATE FUNCTION` statement is used to create stored functions and user-defined functions (UDFs):

- For information about creating stored functions, see Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.
- For information about creating user-defined functions, see Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”.

13.1.8 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_type]

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}
```

`CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See Section 13.1.4, “ALTER TABLE Syntax”. `CREATE INDEX` cannot be used to create a `PRIMARY KEY`; use `ALTER TABLE` instead. For more information about indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See Section 13.1.10, “CREATE TABLE Syntax”. `CREATE INDEX` enables you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index key values are formed by concatenating the values of the given columns.

For string columns, indexes can be created that use only the leading part of column values, using *`col_name(length)`* syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` column indexes.
- Prefixes *must* be specified for `BLOB` and `TEXT` column indexes.
- Prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements is interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.
- For spatial columns, prefix values can be given as described later in this section.

The statement shown here creates an index using the first 10 characters of the `name` column (assuming that `name` has a nonbinary string type):

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations.

Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 1000 bytes long for `MyISAM` tables, and 767 bytes for `InnoDB` tables. The `NDB` storage engine does not support prefixes (see Section 17.1.5.6, “Unsupported or Missing Features in MySQL Cluster”).

A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. This constraint does not apply to `NULL` values except for the `BDB` storage engine. For other engines, a `UNIQUE` index permits multiple `NULL` values for columns that can contain `NULL`. If you specify a prefix value for a column in a `UNIQUE` index, the column values must be unique within the prefix.

`FULLTEXT` indexes are supported only for `MyISAM` tables and can include only `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.9, “Full-Text Search Functions”, for details of operation.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The MyISAM, InnoDB, NDB, BDB, and ARCHIVE storage engines support spatial columns such as (POINT and GEOMETRY. (Section 11.5, “Extensions for Spatial Data”, describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and nonspatial indexes are available according to the following rules.

Spatial indexes (created using SPATIAL INDEX) have these characteristics:

- Available only for MyISAM tables. Specifying SPATIAL INDEX for other storage engines results in an error.
- Indexed columns must be NOT NULL.
- In MySQL 5.0, the full width of each column is indexed by default, but column prefix lengths are permitted. However, as of MySQL 5.0.40, the length is not displayed in SHOW CREATE TABLE output. mysqldump uses that statement. As of that version, if a table with SPATIAL indexes containing prefixed columns is dumped and reloaded, the index is created with no prefixes. (The full column width of each column is indexed.)

Characteristics of nonspatial indexes (created with INDEX, UNIQUE, or PRIMARY KEY):

- Permitted for any storage engine that supports spatial columns except ARCHIVE.
- Columns can be NULL unless the index is a primary key.
- For each spatial column in a non-SPATIAL index except POINT columns, a column prefix length must be specified. (This is the same requirement as for indexed BLOB columns.) The prefix length is given in bytes.
- The index type for a non-SPATIAL index depends on the storage engine. Currently, B-tree is used.
- You can add an index on a column that can have NULL values only for MyISAM, InnoDB, BDB, and MEMORY tables.
- You can add an index on a BLOB or TEXT column only for MyISAM, BDB, and InnoDB tables.

An *index_col_name* specification can end with ASC or DESC. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

Some storage engines permit you to specify an index type when creating an index. Table 13.1, “Index Types Per Storage Engine” shows the permissible index type values supported by different storage engines. Where multiple index types are listed, the first one is the default when no index type specifier is given.

Table 13.1 Index Types Per Storage Engine

Storage Engine	Permissible Index Types
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	BTREE, HASH (see note in text)

Example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Storage engines not listed in the table do not support an *`index_type`* clause in index definitions.

The *`index_type`* clause cannot be used for `FULLTEXT INDEX` or `SPATIAL INDEX` specifications. Full-text index implementation is storage engine dependent. Spatial indexes are implemented as R-tree indexes.

`BTREE` indexes are implemented by the `NDB` storage engine as T-tree indexes.

Note

For indexes on `NDB` table columns, the `USING` option can be specified only for a unique index or primary key. `USING HASH` prevents the creation of an ordered index; otherwise, creating a unique index or primary key on an `NDB` table automatically results in the creation of both an ordered index and a hash index, each of which indexes the same set of columns.

For unique indexes that include one or more `NULL` columns of an `NDB` table, the hash index can be used only to look up literal values, which means that `IS [NOT] NULL` conditions require a full scan of the table. One workaround is to make sure that a unique index using one or more `NULL` columns on such a table is always created in such a way that it includes the ordered index; that is, avoid employing `USING HASH` when creating the index.

If you specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes `RTREE` as a type name, but currently this cannot be specified for any storage engine.

Before MySQL 5.0.60, this option can be given only before the `ON` *`tbl_name`* clause. Use of the option in this position is deprecated as of 5.0.60 and support for it there will be removed in a future MySQL release. If an *`index_type`* option is given in both the earlier and later positions, the final option applies.

`TYPE` *`type_name`* is recognized as a synonym for `USING` *`type_name`*. However, `USING` is the preferred form.

13.1.9 CREATE PROCEDURE and CREATE FUNCTION Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

These statements create stored routines. By default, a routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as *`db_name.sp_name`* when you create it.

The `CREATE FUNCTION` statement is also used in MySQL to support UDFs (user-defined functions). See Section 21.2, “Adding New Functions to MySQL”. A UDF can be regarded as an external stored function. Stored functions share their namespace with UDFs. See Section 9.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

To invoke a stored procedure, use the `CALL` statement (see Section 13.2.1, “CALL Syntax”). To invoke a stored function, refer to it in an expression. The function returns a value during expression evaluation.

As of MySQL 5.0.3, `CREATE PROCEDURE` and `CREATE FUNCTION` require the `CREATE ROUTINE` privilege. They might also require the `SUPER` privilege, depending on the `DEFINER` value, as described later in this section. If binary logging is enabled, `CREATE FUNCTION` might require the `SUPER` privilege, as described in Section 18.6, “Binary Logging of Stored Programs”.

By default, MySQL automatically grants the `ALTER ROUTINE` and `EXECUTE` privileges to the routine creator. This behavior can be changed by disabling the `automatic_sp_privileges` system variable. See Section 18.2.2, “Stored Routines and MySQL Privileges”.

The `DEFINER` and `SQL SECURITY` clauses specify the security context to be used when checking access privileges at routine execution time, as described later in this section.

If the routine name is the same as the name of a built-in SQL function, a syntax error occurs unless you use a space between the name and the following parenthesis when defining the routine or invoking it later. For this reason, avoid using the names of existing SQL functions for your own stored routines.

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to stored routines. It is always permissible to have spaces after a stored routine name, regardless of whether `IGNORE_SPACE` is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used. Parameter names are not case sensitive.

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.

Note

Specifying a parameter as `IN`, `OUT`, or `INOUT` is valid only for a `PROCEDURE`. For a `FUNCTION`, parameters are always regarded as `IN` parameters.

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is `NULL` within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable in the `CALL` statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure

This documentation is for an older version. If you're
This documentation is for an older version. If you're

from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.

Routine parameters cannot be referenced in statements prepared within the routine; see Section C.1, “Restrictions on Stored Programs”.

The following example shows a simple stored procedure that uses an `OUT` parameter:

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

The example uses the `mysql` client `delimiter` command to change the statement delimiter from `;` to `//` while the procedure is being defined. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself. See Section 18.1, “Defining Stored Programs”.

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It indicates the return type of the function, and the function body must contain a `RETURN` *`value`* statement. If the `RETURN` statement returns a value of a different type, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` statement returns an integer, the value returned from the function is the string for the corresponding `ENUM` member of set of `SET` members.

The following example function takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

Parameter types and function return types can be declared to use any valid data type, except that the `COLLATE` attribute cannot be used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The *`routine_body`* consists of a valid SQL routine statement. This can be a simple statement such as `SELECT` or `INSERT`, or a compound statement written using `BEGIN` and `END`. Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described in Section 13.6, “MySQL Compound-Statement Syntax”.

MySQL permits routines to contain DDL statements, such as `CREATE` and `DROP`. MySQL also permits stored procedures (but not stored functions) to contain SQL transaction statements such as `COMMIT`. Stored functions may not contain statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

Statements that return a result set can be used within a stored procedure but not within a stored function. This prohibition includes `SELECT` statements that do not have an `INTO *var_list*` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. For statements that can be determined at function definition time to return a result set, a `Not allowed to return a result set from a function` error occurs (`ER_SP_NO_RETSET`). For statements that can be determined only at runtime to return a result set, a `PROCEDURE %s can't return a result set in the given context` error occurs (`ER_SP_BADSELECT`).

Note

Before MySQL 5.0.10, stored functions created with `CREATE FUNCTION` must not contain references to tables, with limited exceptions. They may include some `SET` statements that contain table references, for example `SET a:= (SELECT MAX(id) FROM t)`, and `SELECT` statements that fetch values directly into variables, for example `SELECT i INTO var1 FROM t`.

`USE` statements within stored routines are not permitted. When a routine is invoked, an implicit `USE *db_name*` is performed (and undone when the routine terminates). This causes the routine to have the given default database while it executes. References to objects in databases other than the routine default database should be qualified with the appropriate database name.

For additional information about statements that are not permitted in stored routines, see Section C.1, “Restrictions on Stored Programs”.

For information about invoking stored procedures from within programs written in a language that has a MySQL interface, see Section 13.2.1, “CALL Syntax”.

MySQL stores the `sql_mode` system variable setting in effect when a routine is created or altered, and always executes the routine with this setting in force, *regardless of the current server SQL mode when the routine begins executing*.

The switch from the SQL mode of the invoker to that of the routine occurs after evaluation of arguments and assignment of the resulting values to routine parameters. If you define a routine in strict SQL mode but invoke it in nonstrict mode, assignment of arguments to routine parameters does not take place in strict mode. If you require that expressions passed to a routine be assigned in strict SQL mode, you should invoke the routine with strict mode in effect.

The `COMMENT` characteristic is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

The `LANGUAGE` characteristic indicates the language in which the routine is written. The server ignores this characteristic; only SQL routines are supported.

A routine is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither `DETERMINISTIC` nor `NOT DETERMINISTIC` is given in the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

routine definition, the default is `NOT DETERMINISTIC`. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.

Assessment of the nature of a routine is based on the "honesty" of the creator: MySQL does not check that a routine declared `DETERMINISTIC` is free of statements that produce nondeterministic results. However, misdeclaring a routine might affect results or affect performance. Declaring a nondeterministic routine as `DETERMINISTIC` might lead to unexpected results by causing the optimizer to make incorrect execution plan choices. Declaring a deterministic routine as `NONDETERMINISTIC` might diminish performance by causing available optimizations not to be used. Prior to MySQL 5.0.44, the `DETERMINISTIC` characteristic is accepted, but not used by the optimizer.

If binary logging is enabled, the `DETERMINISTIC` characteristic affects which routine definitions MySQL accepts. See Section 18.6, "Binary Logging of Stored Programs".

A routine that contains the `NOW()` function (or its synonyms) or `RAND()` is nondeterministic, but it might still be replication-safe. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is called only a single time during the execution of a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

Several characteristics provide information about the nature of data use by the routine. In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be permitted to execute.

- `CONTAINS SQL` indicates that the routine does not contain statements that read or write data. This is the default if none of these characteristics is given explicitly. Examples of such statements are `SET @x = 1` or `DO RELEASE_LOCK('abc')`, which execute but neither read nor write data.
- `NO SQL` indicates that the routine contains no SQL statements.
- `READS SQL DATA` indicates that the routine contains statements that read data (for example, `SELECT`), but not statements that write data.
- `MODIFIES SQL DATA` indicates that the routine contains statements that may write data (for example, `INSERT` or `DELETE`).

The `SQL SECURITY` characteristic can be `DEFINER` or `INVOKER` to specify the security context; that is, whether the routine executes using the privileges of the account named in the routine `DEFINER` clause or the user who invokes it. This account must have permission to access the database with which the routine is associated. The default value is `DEFINER`. As of MySQL 5.0.3, the user who invokes the routine must have the `EXECUTE` privilege for it, as must the `DEFINER` account if the routine executes in definer security context.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the `SQL SECURITY DEFINER` characteristic. The `DEFINER` clause was added in MySQL 5.0.20.

If a *`user`* value is given for the `DEFINER` clause, it should be a MySQL account specified as *`'user_name'@'host_name'`* (the same format used in the `GRANT` statement), `CURRENT_USER`, or `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE PROCEDURE` or `CREATE FUNCTION` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the legal `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only legal *`user`* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If you have the SUPER privilege, you can specify any syntactically legal account name. If the account does not exist, a warning is generated.
- Although it is possible to create a routine with a nonexistent DEFINER account, an error occurs at routine execution time if the SQL SECURITY value is DEFINER but the definer account does not exist.

For more information about stored routine security, see Section 18.5, “Access Control for Stored Programs and Views”.

Within a stored routine that is defined with the SQL SECURITY DEFINER characteristic, CURRENT_USER returns the routine's DEFINER value. For information about user auditing within stored routines, see Section 6.3.9, “SQL-Based MySQL Account Activity Auditing”.

Consider the following procedure, which displays a count of the number of MySQL accounts listed in the mysql.user table:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure is assigned a DEFINER account of 'admin'@'localhost' no matter which user defines it. It executes with the privileges of that account no matter which user invokes it (because the default security characteristic is DEFINER). The procedure succeeds or fails depending on whether invoker has the EXECUTE privilege for it and 'admin'@'localhost' has the SELECT privilege for the mysql.user table.

Now suppose that the procedure is defined with the SQL SECURITY INVOKER characteristic:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
SQL SECURITY INVOKER
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure still has a DEFINER of 'admin'@'localhost', but in this case, it executes with the privileges of the invoking user. Thus, the procedure succeeds or fails depending on whether the invoker has the EXECUTE privilege for it and the SELECT privilege for the mysql.user table.

As of MySQL 5.0.18, the handles the data type of a routine parameter, local routine variable created with DECLARE, or function return value as follows:

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict SQL mode.
- Only scalar values can be assigned. For example, a statement such as SET x = (SELECT 1, 2) is invalid.
- For character data types, if there is a CHARACTER SET attribute in the declaration, the specified character set and its default collation are used. If there is no such attribute, as of MySQL 5.0.25, the database character set and collation that are in effect at the time the server loads the routine into the routine cache are used. (These are given by the values of the character_set_database and collation_database system variables.) If the database character set or collation change while the routine is in the cache, routine execution is unaffected by the change until the next time the server reloads the routine into the cache. The COLLATE attribute is not supported. (This includes use of BINARY, which in this context specifies the binary collation of the character set.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you change the database default character set or collation, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults.

Before MySQL 5.0.18, parameters, return values, and local variables are treated as items in expressions, and are subject to automatic (silent) conversion and truncation. Stored functions ignore the `sql_mode` setting.

13.1.10 CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    select_statement

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }

create_definition:
    col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
        [index_type]
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
        [index_type]
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
        [index_name] [index_type] (index_col_name,...)
        [index_type]
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
        [index_type]
  | [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (index_col_name,...) reference_definition
  | CHECK (expr)

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string'] [reference_definition]

data_type:
    BIT[(length)]
  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME
  | TIMESTAMP
  | DATETIME
  | YEAR
  | CHAR[(length)] [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | VARCHAR(length) [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  | BINARY[(length)]
  | VARBINARY(length)
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | TEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | MEDIUMTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | LONGTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | ENUM(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | SET(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_options:
    table_option [[,] table_option] ...

table_option:
    {ENGINE|TYPE} [=] engine_name
  | AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | CONNECTION [=] 'connect_string'
  | DATA DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | INDEX DIRECTORY [=] 'absolute path to directory'
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | UNION [=] (tbl_name[,tbl_name]...)

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

`CREATE TABLE` creates a table with the given name. You must have the `CREATE` privilege for the table.

Rules for permissible table names are given in Section 9.2, “Schema Object Names”. By default, the table is created in the default database. An error occurs if the table exists, if there is no default database, or if the database does not exist.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write `mydb`.`mytbl`, not `mydb.mytbl`.

Cloning or Copying a Table

Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

For more information, see Section 13.1.10.1, “CREATE TABLE ... LIKE Syntax”.

To create one table from another, add a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

For more information, see Section 13.1.10.2, “CREATE TABLE ... SELECT Syntax”.

Temporary Tables

You can use the TEMPORARY keyword when creating a table. A TEMPORARY table is visible only to the current session, and is dropped automatically when the session is closed. This means that two different sessions can use the same temporary table name without conflicting with each other or with an existing non-TEMPORARY table of the same name. (The existing table is hidden until the temporary table is dropped.) To create temporary tables, you must have the CREATE TEMPORARY TABLES privilege.

Note

CREATE TABLE does not automatically commit the current active transaction if you use the TEMPORARY keyword.

Note

TEMPORARY tables have a very loose relationship with databases (schemas). Dropping a database does not automatically drop any TEMPORARY tables created within that database. Also, you can create a TEMPORARY table in a nonexistent database if you qualify the table name with the database name in the CREATE TABLE statement. In this case, all subsequent references to the table must be qualified with the database name.

Existing Table with Same Name

The keywords IF NOT EXISTS prevent an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the CREATE TABLE statement.

Physical Representation

MySQL represents each table by an .frm table format (definition) file in the database directory. The storage engine for the table might create other files as well.

For InnoDB tables, the file storage is controlled by the innodb_file_per_table configuration option. For each InnoDB table created when this option is turned on, the table data and all associated indexes are

This documentation is for an older version. If you're
This documentation is for an older version. If you're

stored in a .ibd file located inside the database directory. When this option is turned off, all InnoDB tables and indexes are stored in the system tablespace, represented by one or more ibdata* files.

For MyISAM tables, the storage engine creates data and index files. Thus, for each MyISAM table *tbl_name*, there are three disk files.

File	Purpose
tbl_name.frm	Table format (definition) file
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Chapter 14, *Storage Engines*, describes what files each storage engine creates to represent tables.

Data Types and Attributes for Columns

data_type represents the data type in a column definition. *spatial_type* represents a spatial data type. The data type syntax shown is representative only. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see Chapter 11, *Data Types*, and Section 11.5, “Extensions for Spatial Data”.

Some attributes do not apply to all data types. AUTO_INCREMENT applies only to integer and floating-point types. DEFAULT does not apply to the BLOB or TEXT types.

- If neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.
- An integer or floating-point column can have the additional attribute AUTO_INCREMENT. When you insert a value of NULL (recommended) or 0 into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is *value*+1, where *value* is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1.

  To retrieve an AUTO_INCREMENT value after inserting a row, use the LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. See Section 12.13, “Information Functions”, and Section 20.6.7.37, “mysql_insert_id()”.

  If the NO_AUTO_VALUE_ON_ZERO SQL mode is enabled, you can store 0 in AUTO_INCREMENT columns as 0 without generating a new sequence value. See Section 5.1.7, “Server SQL Modes”.

  Note

  There can be only one AUTO_INCREMENT column per table, it must be indexed, and it cannot have a DEFAULT value. An AUTO_INCREMENT column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an AUTO_INCREMENT column that contains 0.

  For MyISAM and BDB tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. See Section 3.6.9, “Using AUTO_INCREMENT”.

  To make MySQL compatible with some ODBC applications, you can find the AUTO_INCREMENT value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This method requires that `sql_auto_is_null` variable is not set to 0. See Section 5.1.4, “Server System Variables”.

For information about `InnoDB` and `AUTO_INCREMENT`, see Section 14.2.3.3, “AUTO_INCREMENT Handling in InnoDB”.

- Character data types (`CHAR`, `VARCHAR`, `TEXT`) can include `CHARACTER SET` and `COLLATE` attributes to specify the character set and collation for the column. For details, see Section 10.1, “Character Set Support”. `CHARSET` is a synonym for `CHARACTER SET`. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

  MySQL 5.0 interprets length specifications in character column definitions in characters. Lengths for `BINARY` and `VARBINARY` are in bytes.

- The `DEFAULT` clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that you can specify `CURRENT_TIMESTAMP` as the default for a `TIMESTAMP` column. See Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP”.

  If a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as described in Section 11.6, “Data Type Default Values”.

  `BLOB` and `TEXT` columns cannot be assigned a default value.

  `CREATE TABLE` fails if a date-valued default is not correct according to the `NO_ZERO_IN_DATE` SQL mode, even if strict SQL mode is not enabled. For example, `c1 DATE DEFAULT '2010-00-00'` causes `CREATE TABLE` to fail with `Invalid default value for 'c1'`.

- A comment for a column can be specified with the `COMMENT` option, up to 255 characters long. The comment is displayed by the `SHOW CREATE TABLE` and `SHOW FULL COLUMNS` statements.

- `KEY` is normally a synonym for `INDEX`. The key attribute `PRIMARY KEY` can also be specified as just `KEY` when given in a column definition. This was implemented for compatibility with other database systems.

- A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. This constraint does not apply to `NULL` values except for the `BDB` storage engine. For other engines, a `UNIQUE` index permits multiple `NULL` values for columns that can contain `NULL`.

- A `PRIMARY KEY` is a unique index where all key columns must be defined as `NOT NULL`. If they are not explicitly declared as `NOT NULL`, MySQL declares them so implicitly (and silently). A table can have only one `PRIMARY KEY`. The name of a `PRIMARY KEY` is always `PRIMARY`, which thus cannot be used as the name for any other kind of index.

  If you do not have a `PRIMARY KEY` and an application asks for the `PRIMARY KEY` in your tables, MySQL returns the first `UNIQUE` index that has no `NULL` columns as the `PRIMARY KEY`.

  In `InnoDB` tables, having a long `PRIMARY KEY` wastes a lot of space. (See Section 14.2.10, “InnoDB Table and Index Structures”.)

- In the created table, a `PRIMARY KEY` is placed first, followed by all `UNIQUE` indexes, and then the nonunique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated `UNIQUE` keys.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate `PRIMARY KEY(index_col_name, ...)` clause.

- If a `PRIMARY KEY` or `UNIQUE` index consists of only one column that has an integer type, you can also refer to the column as `_rowid` in `SELECT` statements.

- In MySQL, the name of a `PRIMARY KEY` is `PRIMARY`. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (`_2`, `_3`, `...`) to make it unique. You can see index names for a table using `SHOW INDEX FROM tbl_name`. See Section 13.7.5.18, “SHOW INDEX Syntax”.

- Some storage engines permit you to specify an index type when creating an index. The syntax for the `index_type` specifier is `USING type_name`.

  Example:

  ```
  CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
  ```

  Before MySQL 5.0.60, `USING` can be given only before the index column list. As of 5.0.60, the preferred position is after the column list. Support for use of the option before the column list will be removed in a future MySQL release.

  `index_option` values specify additional options for an index. `USING` is one such option. For details about permissible `index_option` values, see Section 13.1.8, “CREATE INDEX Syntax”.

  For more information about indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

- In MySQL 5.0, only the `MyISAM`, `InnoDB`, `BDB`, and `MEMORY` storage engines support indexes on columns that can have `NULL` values. In other cases, you must declare indexed columns as `NOT NULL` or an error results.

- For `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns, indexes can be created that use only the leading part of column values, using `col_name(length)` syntax to specify an index prefix length. `BLOB` and `TEXT` columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first `length` characters of each column value for `CHAR`, `VARCHAR`, and `TEXT` columns, and the first `length` bytes of each column value for `BINARY`, `VARBINARY`, and `BLOB` columns. Indexing only a prefix of column values like this can make the index file much smaller. For additional information about index prefixes, see Section 13.1.8, “CREATE INDEX Syntax”.

  Only the `MyISAM`, `BDB`, and `InnoDB` storage engines support indexing on `BLOB` and `TEXT` columns. For example:

  ```
  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  ```

  Prefixes can be up to 1000 bytes long (767 bytes for `InnoDB` tables).

  Note

  Prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements is interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into

This documentation is for an older version. If you're
This documentation is for an older version. If you're

account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

- An *`index_col_name`* specification can end with `ASC` or `DESC`. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.
- When you use `ORDER BY` or `GROUP BY` on a column in a `SELECT`, the server sorts values using only the initial number of bytes indicated by the `max_sort_length` system variable.
- You can create special `FULLTEXT` indexes, which are used for full-text searches. Only the `MyISAM` storage engine supports `FULLTEXT` indexes. They can be created only from `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.9, “Full-Text Search Functions”, for details of operation.
- You can create `SPATIAL` indexes on spatial data types. Spatial types are supported only for `MyISAM` tables and indexed columns must be declared as `NOT NULL`. See Section 11.5, “Extensions for Spatial Data”.
- `InnoDB` tables support checking of foreign key constraints. The columns of the referenced table must always be explicitly named. Both `ON DELETE` and `ON UPDATE` actions on foreign keys. For more detailed information and examples, see Section 13.1.10.3, “Using FOREIGN KEY Constraints”. For information specific to foreign keys in `InnoDB`, see Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”.

  For other storage engines, MySQL Server parses and ignores the `FOREIGN KEY` and `REFERENCES` syntax in `CREATE TABLE` statements. The `CHECK` clause is parsed but ignored by all storage engines. See Section 1.8.2.4, “Foreign Key Differences”.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. `InnoDB` essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL requires that the referenced columns be indexed for performance. However, `InnoDB` does not enforce any requirement that the referenced columns be declared `UNIQUE` or `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only keys that are both `UNIQUE` (or `PRIMARY`) and `NOT NULL`.

MySQL parses but ignores “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column

This documentation is for an older version. If you're
This documentation is for an older version. If you're

specification. MySQL accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification.

- There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table and depends on the factors discussed in Section C.7.4, “Limits on Table Column Count and Row Size”.

Storage Engines

The `ENGINE` table option specifies the storage engine for the table. `TYPE` is a synonym, but `ENGINE` is the preferred option name.

The `ENGINE` table option specifies the storage engine for the table, using one of the names shown in the following table. The engine name can be unquoted or quoted. The quoted name `'DEFAULT'` is equivalent to specifying the default storage engine name.

Storage Engine	Description
`ARCHIVE`	The archiving storage engine. See Section 14.8, “The ARCHIVE Storage Engine”.
`BDB`	Transaction-safe tables with page locking. Also known as `BerkeleyDB`. See Section 14.5, “The BDB (BerkeleyDB) Storage Engine”.
`CSV`	Tables that store rows in comma-separated values format. See Section 14.9, “The CSV Storage Engine”.
`EXAMPLE`	An example engine. See Section 14.6, “The EXAMPLE Storage Engine”.
`FEDERATED`	Storage engine that accesses remote tables. See Section 14.7, “The FEDERATED Storage Engine”.
`HEAP`	This is a synonym for `MEMORY`.
`ISAM` (*OBSOLETE*)	Not available in MySQL 5.0. If you are upgrading to MySQL 5.0 from a previous version, you should convert any existing `ISAM` tables to `MyISAM` *before* performing the upgrade.
`InnoDB`	Transaction-safe tables with row locking and foreign keys. See Section 14.2, “The InnoDB Storage Engine”.
`MEMORY`	The data for this storage engine is stored only in memory. See Section 14.4, “The MEMORY (HEAP) Storage Engine”.
`MERGE`	A collection of `MyISAM` tables used as one table. Also known as `MRG_MyISAM`. See Section 14.3, “The MERGE Storage Engine”.
`MyISAM`	The binary portable storage engine that is the default storage engine used by MySQL. See Section 14.1, “The MyISAM Storage Engine”.
`NDBCLUSTER`	Clustered, fault-tolerant, memory-based tables. Also known as `NDB`. See Chapter 17, *MySQL Cluster*.

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is `MyISAM`. For example, if a table definition includes the `ENGINE=BDB` option but the MySQL server does not support `BDB` tables, the table is created as a `MyISAM` table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are nontransactional (to get more speed). In MySQL 5.0, a warning occurs if the storage engine specification is not honored.

Engine substitution can be controlled by the setting of the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in Section 5.1.7, “Server SQL Modes”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Optimizing Performance

The other table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use `ALTER TABLE` to convert the table to use a different storage engine.

- `AUTO_INCREMENT`

  The initial `AUTO_INCREMENT` value for the table. In MySQL 5.0, this works for `MyISAM` and `MEMORY` tables. It is also supported for `InnoDB` as of MySQL 5.0.3. To set the first auto-increment value for engines that do not support the `AUTO_INCREMENT` table option, insert a "dummy" row with a value one less than the desired value after creating the table, and then delete the dummy row.

  For engines that support the `AUTO_INCREMENT` table option in `CREATE TABLE` statements, you can also use `ALTER TABLE *tbl_name* AUTO_INCREMENT = *N*` to reset the `AUTO_INCREMENT` value. The value cannot be set lower than the maximum value currently in the column.

- `AVG_ROW_LENGTH`

  An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

  When you create a `MyISAM` table, MySQL uses the product of the `MAX_ROWS` and `AVG_ROW_LENGTH` options to decide how big the resulting table is. If you don't specify either option, the maximum size for `MyISAM` data and index table files is 256TB of data by default (4GB before MySQL 5.0.6). (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the `myisam_data_pointer_size` system variable, which was added in MySQL 4.1.2. (See Section 5.1.4, "Server System Variables".) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable. Setting the value to 7 permits table sizes up to 65,536TB.

- `[DEFAULT] CHARACTER SET`

  Specify a default character set for the table. `CHARSET` is a synonym for `CHARACTER SET`. If the character set name is `DEFAULT`, the database character set is used.

- `CHECKSUM`

  Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The `CHECKSUM TABLE` statement reports the checksum. (`MyISAM` only.)

- `[DEFAULT] COLLATE`

  Specify a default collation for the table.

- `COMMENT`

  A comment for the table, up to 60 characters long.

- `CONNECTION`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The connection string for a `FEDERATED` table. This option is available as of MySQL 5.0.13; before that, use a `COMMENT` option for the connection string.

- `DATA DIRECTORY`, `INDEX DIRECTORY`

  By using `DATA DIRECTORY='directory'` or `INDEX DIRECTORY='directory'` you can specify where the `MyISAM` storage engine should put a table's data file and index file. The directory must be the full path name to the directory, not a relative path.

  These options work only when you are not using the `--skip-symbolic-links` option. Your operating system must also have a working, thread-safe `realpath()` call. See Section 8.12.4.2, "Using Symbolic Links for MyISAM Tables on Unix", for more complete information.

  If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. As of MySQL 5.0.48, to suppress this behavior, start the server with the `--keep_files_on_create` option, in which case `MyISAM` will not overwrite existing files and returns an error instead.

  If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, MyISAM always returns an error. It will not overwrite a file in the specified directory.

  Important

  Beginning with MySQL 5.0.60, you cannot use path names that contain the MySQL data directory with `DATA DIRECTORY` or `INDEX DIRECTORY`. (See Bug #32167.)

- `DELAY_KEY_WRITE`

  Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the `delay_key_write` system variable in Section 5.1.4, "Server System Variables". (`MyISAM` only.)

- `INSERT_METHOD`

  If you want to insert data into a `MERGE` table, you must specify with `INSERT_METHOD` the table into which the row should be inserted. `INSERT_METHOD` is an option useful for `MERGE` tables only. Use a value of `FIRST` or `LAST` to have inserts go to the first or last table, or a value of `NO` to prevent inserts. See Section 14.3, "The MERGE Storage Engine".

- `MAX_ROWS`

  The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

  The `NDB` storage engine treats this value as a maximum. If you plan to create very large MySQL Cluster tables (containing millions of rows), you should use this option to insure that `NDB` allocates sufficient number of index slots in the hash table used for storing hashes of the table's primary keys by setting `MAX_ROWS = 2 * rows`, where *`rows`* is the number of rows that you expect to insert into the table.

  The maximum `MAX_ROWS` value is 4294967295; larger values are truncated to this limit.

- `MIN_ROWS`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The minimum number of rows you plan to store in the table. The `MEMORY` storage engine uses this option as a hint about memory use.

- `PACK_KEYS`

  `PACK_KEYS` takes effect only with `MyISAM` tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to `DEFAULT` tells the storage engine to pack only long `CHAR`, `VARCHAR`, `BINARY`, or `VARBINARY` columns.

  If you do not use `PACK_KEYS`, the default is to pack strings, but not numbers. If you use `PACK_KEYS=1`, numbers are packed as well.

  When packing binary number keys, MySQL uses prefix compression:

  - Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
  - The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

  This means that if you have many equal keys on two consecutive rows, all following “same” keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have `NULL` values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is `NULL`.)

- `PASSWORD`

  This option is unused. If you have a need to scramble your `.frm` files and make them unusable to any other MySQL server, please contact our sales department.

- `ROW_FORMAT`

  Defines how the rows should be stored. For `MyISAM` tables, the option value can be `FIXED` or `DYNAMIC` for static or variable-length row format. `myisampack` sets the type to `COMPRESSED`. See Section 14.1.3, “MyISAM Table Storage Formats”.

  Starting with MySQL 5.0.3, for `InnoDB` tables, rows are stored in compact format (`ROW_FORMAT=COMPACT`) by default. The noncompact format used in older versions of MySQL can still be requested by specifying `ROW_FORMAT=REDUNDANT`.

  Note

  When executing a `CREATE TABLE` statement, if you specify a row format which is not supported by the storage engine that is used for the table, the table is created using that storage engine's default row format. The information reported in this column in response to `SHOW TABLE STATUS` is the actual row format used. This may differ from the value in the `Create_options` column because the original `CREATE TABLE` definition is retained during creation.

- `RAID_TYPE`

  `RAID` support has been removed as of MySQL 5.0.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `UNION`

  `UNION` is used when you want to access a collection of identical `MyISAM` tables as one. This works only with `MERGE` tables. See Section 14.3, “The MERGE Storage Engine”.

  You must have `SELECT`, `UPDATE`, and `DELETE` privileges for the tables you map to a `MERGE` table.

Note

Formerly, all tables used had to be in the same database as the `MERGE` table itself. This restriction no longer applies.

Important

The original `CREATE TABLE` statement, including all specifications and table options are stored by MySQL when the table is created. The information is retained so that if you change storage engines, collations or other settings using an `ALTER TABLE` statement, the original table options specified are retained. This enables you to change between `InnoDB` and `MyISAM` table types even though the row formats supported by the two engines are different.

Because the text of the original statement is retained, but due to the way that certain values and options may be silently reconfigured (such as the `ROW_FORMAT`), the active table definition (accessible through `DESCRIBE` or with `SHOW TABLE STATUS`) and the table creation string (accessible through `SHOW CREATE TABLE`) will report different values.

13.1.10.1 CREATE TABLE ... LIKE Syntax

Use `CREATE TABLE ... LIKE` to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table. The `SELECT` privilege is required on the original table.

`LIKE` works only for base tables, not for views.

`CREATE TABLE ... LIKE` does not preserve any `DATA DIRECTORY` or `INDEX DIRECTORY` table options that were specified for the original table, or any foreign key definitions.

If the original table is a `TEMPORARY` table, `CREATE TABLE ... LIKE` does not preserve `TEMPORARY`. To create a `TEMPORARY` destination table, use `CREATE TEMPORARY TABLE ... LIKE`.

In MySQL 5.0, changes to the SQL mode do not affect `CREATE TABLE ... LIKE`. If the current SQL mode is different from the mode in effect when the original table was created, the statement succeeds even if the table definition is invalid for the new mode.

13.1.10.2 CREATE TABLE ... SELECT Syntax

You can create one table from another by adding a `SELECT` statement at the end of the `CREATE TABLE` statement:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the SELECT. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a MyISAM table with three columns, a, b, and c. The ENGINE option is part of the CREATE TABLE statement, and should not be used following the SELECT; this would result in a syntax error. The same is true for other CREATE TABLE options such as CHARSET.

Notice that the columns from the SELECT statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

For each row in table foo, a row is inserted in bar with the values from foo and default values for the new columns.

In a table resulting from CREATE TABLE ... SELECT, columns named only in the CREATE TABLE part come first. Columns named in both parts or only in the SELECT part come after that. The data type of SELECT columns can be overridden by also specifying the column in the CREATE TABLE part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

You can precede the SELECT by IGNORE or REPLACE to indicate how to handle rows that duplicate unique key values. With IGNORE, rows that duplicate an existing row on a unique key value are discarded. With REPLACE, new rows replace rows that have the same unique key value. If neither IGNORE nor REPLACE is specified, duplicate unique key values result in an error.

CREATE TABLE ... SELECT does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the SELECT statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the AUTO_INCREMENT attribute is not preserved, and VARCHAR columns can become CHAR columns. Retrained attributes are NULL (or NOT NULL) and, for those columns that have them, CHARACTER SET, COLLATION, COMMENT, and the DEFAULT clause.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When creating a table with `CREATE TABLE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a column in the created table:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

For `CREATE TABLE ... SELECT`, if `IF NOT EXISTS` is given and the destination table already exists, MySQL handles the statement as follows:

- The table definition given in the `CREATE TABLE` part is ignored. No error occurs, even if the definition does not match that of the existing table. MySQL attempts to insert the rows from the `SELECT` part anyway.
- If there is a mismatch between the number of columns in the table and the number of columns produced by the `SELECT` part, the selected values are assigned to the rightmost columns. For example, if the table contains *n* columns and the `SELECT` produces *m* columns, where $m < n$, the selected values are assigned to the *m* rightmost columns in the table. Each of the initial $n - m$ columns is assigned its default value, either that specified explicitly in the column definition or the implicit column data type default if the definition contains no default. If the `SELECT` part produces too many columns ($m > n$), an error occurs.
- If strict SQL mode is enabled and any of these initial columns do not have an explicit default value, the statement fails with an error.

The following example illustrates `IF NOT EXISTS` handling:

```
mysql> CREATE TABLE t1 (i1 INT DEFAULT 0, i2 INT, i3 INT, i4 INT);
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE IF NOT EXISTS t1 (c1 CHAR(10)) SELECT 1, 2;
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1;
+------+------+------+------+
| i1   | i2   | i3   | i4   |
+------+------+------+------+
|    0 | NULL |    1 |    2 |
+------+------+------+------+
1 row in set (0.00 sec)
```

To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts during `CREATE TABLE ... SELECT`.

13.1.10.3 Using FOREIGN KEY Constraints

MySQL supports foreign keys, which let you cross-reference related data across tables, and foreign key constraints, which help keep this spread-out data consistent. The essential syntax for a foreign key constraint definition in a `CREATE TABLE` or `ALTER TABLE` statement looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

index_name represents a foreign key ID. The *index_name* value is ignored if there is already an explicitly defined index on the child table that can support the foreign key. Otherwise, MySQL implicitly creates a foreign key index that is named according to the following rules:

- If defined, the `CONSTRAINT` *symbol* value is used. Otherwise, the `FOREIGN KEY` *index_name* value is used.
- If neither a `CONSTRAINT` *symbol* or `FOREIGN KEY` *index_name* is defined, the foreign key index name is generated using the name of the referencing foreign key column.

Foreign keys definitions are subject to the following conditions:

- Foreign key relationships involve a parent table that holds the central data values, and a child table with identical values pointing back to its parent. The `FOREIGN KEY` clause is specified in the child table. The parent and child tables must use the same storage engine. They must not be `TEMPORARY` tables.
- Corresponding columns in the foreign key and the referenced key must have similar data types. *The size and sign of integer types must be the same*. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.
- When `foreign_key_checks` is enabled, which is the default setting, character set conversion is not permitted on tables that include a character string column used in a foreign key constraint. The workaround is described in Section 13.1.4, “ALTER TABLE Syntax”.
- MySQL requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. This index might be silently dropped later, if you create another index that can be used to enforce the foreign key constraint. *index_name*, if given, is used as described previously.
- `InnoDB` permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- Index prefixes on foreign key columns are not supported. One consequence of this is that `BLOB` and `TEXT` columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.
- If the `CONSTRAINT` *symbol* clause is given, the *symbol* value, if used, must be unique in the database. A duplicate *symbol* will result in an error similar to: `ERROR 1005 (HY000): Can't create table 'test.#sql-211d_3' (errno: 121)`. If the clause is not given, or a *symbol* is not included following the `CONSTRAINT` keyword, a name for the constraint is created automatically.
- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This includes both parent and child tables.

Referential Actions

This section describes how foreign keys help guarantee referential integrity.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For storage engines supporting foreign keys, MySQL rejects any `INSERT` or `UPDATE` operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table.

When an `UPDATE` or `DELETE` operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified using `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. MySQL supports five options regarding the action to be taken, listed here:

- `CASCADE`: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table. Both `ON DELETE CASCADE` and `ON UPDATE CASCADE` are supported. Between two tables, do not define several `ON UPDATE CASCADE` clauses that act on the same column in the parent table or in the child table.

  Note

  Cascaded foreign key actions do not activate triggers.

- `SET NULL`: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to `NULL`. Both `ON DELETE SET NULL` and `ON UPDATE SET NULL` clauses are supported.

  If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as `NOT NULL`*.

- `RESTRICT`: Rejects the delete or update operation for the parent table. Specifying `RESTRICT` (or `NO ACTION`) is the same as omitting the `ON DELETE` or `ON UPDATE` clause.

- `NO ACTION`: A keyword from standard SQL. In MySQL, equivalent to `RESTRICT`. The MySQL Server rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` is the same as `RESTRICT`.

- `SET DEFAULT`: This action is recognized by the MySQL parser, but `InnoDB` rejects table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

For an `ON DELETE` or `ON UPDATE` that is not specified, the default action is always `RESTRICT`.

MySQL supports foreign key references between one column and another within a table. (A column cannot have a foreign key reference to itself.) In these cases, “child table records” really refers to dependent records within the same table.

Examples of Foreign Key Clauses

Here is a simple example that relates `parent` and `child` tables through a single-column foreign key:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON DELETE CASCADE
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
) ENGINE=INNODB;
```

A more complex example in which a `product_order` table has foreign keys for two other tables. One foreign key references a two-column index in the `product` table. The other references a single-column index in the `customer` table:

```
CREATE TABLE product (
    category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)
)   ENGINE=INNODB;

CREATE TABLE customer (
    id INT NOT NULL,
    PRIMARY KEY (id)
)   ENGINE=INNODB;

CREATE TABLE product_order (
    no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,

    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    INDEX (customer_id),

    FOREIGN KEY (product_category, product_id)
      REFERENCES product(category, id)
      ON UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY (customer_id)
      REFERENCES customer(id)
)   ENGINE=INNODB;
```

Adding foreign keys

You can add a new foreign key constraint to an existing table by using `ALTER TABLE`. The syntax relating to foreign keys for this statement is shown here:

```
ALTER TABLE tbl_name
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using `ALTER TABLE`, *remember to create the required indexes first.*

Dropping Foreign Keys

You can also use `ALTER TABLE` to drop foreign keys, using the syntax shown here:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the `FOREIGN KEY` clause included a `CONSTRAINT` name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the *`fk_symbol`* value is generated internally when the foreign key is created. To find out the symbol value when you want to drop a foreign key, use a `SHOW CREATE TABLE` statement, as shown here:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SHOW CREATE TABLE ibtest11c\G
*************************** 1. row ***************************
       Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

Adding and dropping a foreign key in separate clauses of a single `ALTER TABLE` statement may be problematic in some cases and is therefore unsupported. Use separate statements for each operation.

If an `ALTER TABLE` statement results in changes to column values (for example, because a column is truncated), MySQL's foreign key constraint checks do not notice possible violations caused by changing the values.

Foreign Keys and Other MySQL Statements

Table and column identifiers in a `FOREIGN KEY ... REFERENCES ...` clause can be quoted within backticks (`` ` ``). Alternatively, double quotation marks (`"`) can be used if the `ANSI_QUOTES` SQL mode is enabled. The setting of the `lower_case_table_names` system variable is also taken into account.

You can view a child table's foreign key definitions as part of the output of the `SHOW CREATE TABLE` statement:

```
SHOW CREATE TABLE tbl_name;
```

You can also obtain information about foreign keys by querying the `INFORMATION_SCHEMA.KEY_COLUMN_USAGE` table.

`mysqldump` produces correct definitions of tables in the dump file, including the foreign keys for child tables.

To make it easier to reload dump files for tables that have foreign key relationships, `mysqldump` automatically includes a statement in the dump output to set `foreign_key_checks` to 0. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. It is also possible to set this variable manually:

```
mysql> SET foreign_key_checks = 0;
mysql> SOURCE dump_file_name;
mysql> SET foreign_key_checks = 1;
```

This enables you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting `foreign_key_checks` to 0 can also be useful for ignoring foreign key constraints during `LOAD DATA` and `ALTER TABLE` operations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

However, even if `foreign_key_checks = 0`, MySQL does not permit the creation of a foreign key constraint where a column references a nonmatching column type. Also, if a table has foreign key constraints, `ALTER TABLE` cannot be used to alter the table to use another storage engine. To change the storage engine, you must drop any foreign key constraints first.

You cannot issue `DROP TABLE` for a table that is referenced by a `FOREIGN KEY` constraint, unless you do `SET foreign_key_checks = 0`. When you drop a table, any constraints that were defined in the statement used to create that table are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the correct column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns Error 1005 and refers to Error 150 in the error message, which means that a foreign key constraint was not correctly formed. Similarly, if an `ALTER TABLE` fails due to Error 150, this means that a foreign key definition would be incorrectly formed for the altered table.

For `InnoDB` tables, you can obtain a detailed explanation of the most recent `InnoDB` foreign key error in the MySQL Server, by checking the output of `SHOW ENGINE INNODB STATUS`.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential-integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. MySQL essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL requires that the referenced columns be indexed for performance reasons. However, the system does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only `UNIQUE` (including `PRIMARY`) and `NOT NULL` keys.

Furthermore, MySQL parses but ignores “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification. For storage engines that do not support foreign keys (such as `MyISAM`), MySQL Server parses and ignores foreign key specifications.

13.1.10.4 Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a `CREATE TABLE` or `ALTER TABLE` statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

All changes are subject to the internal row-size limit of 65,535 bytes, which may cause some attempts at data type changes to fail. See Section C.7.4, “Limits on Table Column Count and Row Size”.

Some silent column specification changes include modifications to attribute or index specifications:

- `TIMESTAMP` display sizes are discarded.

  Also note that `TIMESTAMP` columns are `NOT NULL` by default.

- Columns that are part of a `PRIMARY KEY` are made `NOT NULL` even if not declared that way.

- Trailing spaces are automatically deleted from `ENUM` and `SET` member values when the table is created.

- MySQL maps certain data types used by other SQL database vendors to MySQL types. See Section 11.9, “Using Data Types from Other Database Engines”.

- If you include a `USING` clause to specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

Possible data type changes are given in the following list. If a version number is given, the change occurs only up to the versions listed. After that, an error occurs if a column cannot be created using the specified data type.

- Before MySQL 5.0.3, `VARCHAR` columns with a length less than four are changed to `CHAR`.

- Before MySQL 5.0.3, if any column in a table has a variable length, the entire row becomes variable-length as a result. Therefore, if a table contains any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB`), all `CHAR` columns longer than three characters are changed to `VARCHAR` columns. This does not affect how you use the columns in any way; in MySQL, `VARCHAR` is just a different way to store characters. MySQL performs this conversion because it saves space and makes table operations faster. See Chapter 14, *Storage Engines*.

- Before MySQL 5.0.3, a `CHAR` or `VARCHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `VARCHAR(500)` is converted to `TEXT`, and `VARCHAR(200000)` is converted to `MEDIUMTEXT`. Similar conversions occur for `BINARY` and `VARBINARY`, except that they are converted to a `BLOB` type.

  Note that these conversions result in a change in behavior with regard to treatment of trailing spaces.

  As of MySQL 5.0.3, a `CHAR` or `BINARY` column with a length specification greater than 255 is not silently converted. Instead, an error occurs. From MySQL 5.0.6 on, silent conversion of `VARCHAR` and `VARBINARY` columns with a length specification greater than 65535 does not occur if strict SQL mode is enabled. Instead, an error occurs.

- Before MySQL 5.0.10, for a specification of `DECIMAL(M,D)`, if `M` is not larger than `D`, it is adjusted upward. For example, `DECIMAL(10,10)` becomes `DECIMAL(11,10)`. As of MySQL 5.0.10, `DECIMAL(10,10)` is created as specified.

- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

To see whether MySQL used a data type other than the one you specified, issue a `DESCRIBE` or `SHOW CREATE TABLE` statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using `myisampack`. See Section 14.1.3.3, “Compressed Table Characteristics”.

13.1.11 CREATE TRIGGER Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named *`tbl_name`*, which must refer to a permanent table. You cannot associate a trigger with a `TEMPORARY` table or a view. `CREATE TRIGGER` was added in MySQL 5.0.2.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

This section describes `CREATE TRIGGER` syntax. For additional discussion, see Section 18.3.1, “Trigger Syntax and Examples”.

In MySQL 5.0 `CREATE TRIGGER` requires the `SUPER` privilege.

The `DEFINER` clause determines the security context to be used when checking access privileges at trigger activation time. It was added in MySQL 5.0.17. See later in this section for more information.

`trigger_time` is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after each row to be modified.

`trigger_event` indicates the kind of operation that activates the trigger. These *`trigger_event`* values are permitted:

- `INSERT`: The trigger activates whenever a new row is inserted into the table; for example, through `INSERT`, `LOAD DATA`, and `REPLACE` statements.
- `UPDATE`: The trigger activates whenever a row is modified; for example, through `UPDATE` statements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `DELETE`: The trigger activates whenever a row is deleted from the table; for example, through `DELETE` and `REPLACE` statements. `DROP TABLE` and `TRUNCATE TABLE` statements on the table do *not* activate this trigger, because they do not use `DELETE`.

The *`trigger_event`* does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an `INSERT` trigger activates not only for `INSERT` statements but also `LOAD DATA` statements because both statements insert rows into a table.

A potentially confusing example of this is the `INSERT INTO ... ON DUPLICATE KEY UPDATE ...` syntax: a `BEFORE INSERT` trigger activates for every row, followed by either an `AFTER INSERT` trigger or both the `BEFORE UPDATE` and `AFTER UPDATE` triggers, depending on whether there was a duplicate key for the row.

Note

Cascaded foreign key actions do not activate triggers.

There cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. But you can have a `BEFORE UPDATE` and a `BEFORE INSERT` trigger, or a `BEFORE UPDATE` and an `AFTER UPDATE` trigger.

`trigger_body` is the statement to execute when the trigger activates. To execute multiple statements, use the `BEGIN ... END` compound statement construct. This also enables you to use the same statements that are permissible within stored routines. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”. Some statements are not permitted in triggers; see Section C.1, “Restrictions on Stored Programs”.

Within the trigger body, you can refer to columns in the subject table (the table associated with the trigger) by using the aliases `OLD` and `NEW`. `OLD.`*`col_name`* refers to a column of an existing row before it is updated or deleted. `NEW.`*`col_name`* refers to the column of a new row to be inserted or an existing row after it is updated.

MySQL stores the `sql_mode` system variable setting in effect when a trigger is created, and always executes the trigger body with this setting in force, *regardless of the current server SQL mode when the trigger begins executing*.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If a *`user`* value is given, it should be a MySQL account specified as *`'user_name'@'host_name'`* (the same format used in the `GRANT` statement), `CURRENT_USER`, or `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE TRIGGER` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the legal `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only legal *`user`* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.
- If you have the `SUPER` privilege, you can specify any syntactically legal account name. If the account does not exist, a warning is generated.
- Although it is possible to create a trigger with a nonexistent `DEFINER` account, it is not a good idea for such triggers to be activated until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.

Note: Because MySQL currently requires the `SUPER` privilege for the use of `CREATE TRIGGER`, only the second of the preceding rules applies. (MySQL 5.1.6 implements the `TRIGGER` privilege and requires

This documentation is for an older version. If you're
This documentation is for an older version. If you're

that privilege for trigger creation, so at that point both rules come into play and SUPER is required only for specifying a DEFINER value other than your own account.)

From MySQL 5.0.17 on, MySQL takes the DEFINER user into account when checking trigger privileges as follows:

- At CREATE TRIGGER time, the user who issues the statement must have the SUPER privilege.
- At trigger activation time, privileges are checked against the DEFINER user. This user must have these privileges:
  - The SUPER privilege.
  - The SELECT privilege for the subject table if references to table columns occur using OLD.*col_name* or NEW.*col_name* in the trigger body.
  - The UPDATE privilege for the subject table if table columns are targets of SET NEW.*col_name* = *value* assignments in the trigger body.
  - Whatever other privileges normally are required for the statements executed by the trigger.

Before MySQL 5.0.17, DEFINER is not available and MySQL checks trigger privileges like this:

- At CREATE TRIGGER time, the user who issues the statement must have the SUPER privilege.
- At trigger activation time, privileges are checked against the user whose actions cause the trigger to be activated. This user must have whatever privileges normally are required for the statements executed by the trigger.

For more information about trigger security, see Section 18.5, “Access Control for Stored Programs and Views”.

Within a trigger body, the CURRENT_USER() function returns the account used to check privileges at trigger activation time. Consistent with the privilege-checking rules just given, CURRENT_USER() returns the DEFINER user from MySQL 5.0.17 on. Before 5.0.17, CURRENT_USER() returns the user whose actions caused the trigger to be activated. For information about user auditing within triggers, see Section 6.3.9, “SQL-Based MySQL Account Activity Auditing”.

If you use LOCK TABLES to lock a table that has triggers, the tables used within the trigger are also locked, as described in Section 13.3.5.2, “LOCK TABLES and Triggers”.

For additional discussion of trigger use, see Section 18.3.1, “Trigger Syntax and Examples”.

13.1.12 CREATE VIEW Syntax

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

The CREATE VIEW statement creates a new view, or replaces an existing view if the OR REPLACE clause is given. This statement was added in MySQL 5.0.1. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW is the same as ALTER VIEW.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The *select_statement* is a SELECT statement that provides the definition of the view. (Selecting from the view selects, in effect, using the SELECT statement.) The *select_statement* can select from base tables or other views.

The view definition is “frozen” at creation time and is not affected by subsequent changes to the definitions of the underlying tables. For example, if a view is defined as SELECT * on a table, new columns added to the table later do not become part of the view, and columns dropped from the table will result in an error when selecting from the view.

The ALGORITHM clause affects how MySQL processes the view. The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at view invocation time. The WITH CHECK OPTION clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The CREATE VIEW statement requires the CREATE VIEW privilege for the view, and some privilege for each column selected by the SELECT statement. For columns used elsewhere in the SELECT statement, you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view. CREATE VIEW might also require the SUPER privilege, depending on the DEFINER value, as described later in this section.

When a view is referenced, privilege checking occurs as described later in this section.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, use *db_name.view_name* syntax to qualify the view name with the database name:

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Within a database, base tables and views share the same namespace, so a base table and a view cannot have the same name.

Columns retrieved by the SELECT statement can be simple references to table columns, or expressions that use functions, constant values, operators, and so forth.

A view must have unique column names with no duplicates, just like a base table. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, the optional *column_list* clause can be given as a list of comma-separated identifiers. The number of names in *column_list* must be the same as the number of columns retrieved by the SELECT statement.

Note

Prior to MySQL 5.0.72, when you modify an existing view, the server saves a backup of the current view definition under the view database directory, in a subdirectory named arc. The backup file for a view v is named v.frm-00001. If you alter the view again, the next backup is named v.frm-00002. The three latest view backup definitions are stored.

Backed up view definitions are not preserved by mysqldump, or any other such programs, but you can retain them using a file copy operation. However, they are not needed for anything but to provide you with a backup of your previous view definition.

It is safe to remove these backup definitions, but only while mysqld is not running. If you delete the arc subdirectory or its files while mysqld is running, an error occurs the next time you try to alter the view:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> ALTER VIEW v AS SELECT * FROM t;
ERROR 6 (HY000): Error on delete of '.\test\arc/v.frm-0004' (Errcode:
2)
```

Unqualified table or view names in the `SELECT` statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the appropriate database name.

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables.

The following example defines a view that selects two columns from another table as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

A view definition is subject to the following restrictions:

- The `SELECT` statement cannot contain a subquery in the `FROM` clause.
- The `SELECT` statement cannot refer to system variables or user-defined variables.
- Within a stored program, the `SELECT` statement cannot refer to program parameters or local variables.
- The `SELECT` statement cannot refer to prepared statement parameters.
- Any table or view referred to in the definition must exist. After the view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the `CHECK TABLE` statement.
- The definition cannot refer to a `TEMPORARY` table, and you cannot create a `TEMPORARY` view.
- You cannot associate a trigger with a view.
- As of MySQL 5.0.52, aliases for column names in the `SELECT` statement are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

`ORDER BY` is permitted in a view definition, but it is ignored if you select from a view using a statement that has its own `ORDER BY`.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a `LIMIT` clause, and you select from the view using a statement that has its own `LIMIT` clause, it is undefined which limit applies. This same principle applies to options such as `ALL`, `DISTINCT`, or `SQL_SMALL_RESULT` that follow the `SELECT` keyword, and to clauses such as `INTO`, `FOR UPDATE`, `LOCK IN SHARE MODE`, and `PROCEDURE`.

If you create a view and then change the query processing environment by changing system variables, that may affect the results you get from the view:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| mycol |
+-------+
1 row in set (0.01 sec)

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| abc   |
+-------+
1 row in set (0.00 sec)
```

The `DEFINER` and `SQL SECURITY` clauses determine which MySQL account to use when checking access privileges for the view when a statement is executed that references the view. These clauses were addded in MySQL 5.0.13, but have no effect until MySQL 5.0.16. The legal `SQL SECURITY` characteristic values are `DEFINER` (the default) and `INVOKER`. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively.

If a *`user`* value is given for the `DEFINER` clause, it should be a MySQL account specified as *`'user_name'@'host_name'`* (the same format used in the `GRANT` statement), `CURRENT_USER`, or `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE VIEW` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only valid *`user`* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.
- If you have the `SUPER` privilege, you can specify any syntactically valid account name. If the account does not exist, a warning is generated.
- Although it is possible to create a view with a nonexistent `DEFINER` account, an error occurs when the view is referenced if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.

For more information about view security, see Section 18.5, “Access Control for Stored Programs and Views”.

Within a view definition, `CURRENT_USER` returns the view's `DEFINER` value by default as of MySQL 5.0.24. For older versions, and for views defined with the `SQL SECURITY INVOKER` characteristic, `CURRENT_USER` returns the account for the view's invoker. For information about user auditing within views, see Section 6.3.9, “SQL-Based MySQL Account Activity Auditing”.

Within a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, `CURRENT_USER` returns the routine's `DEFINER` value. This also affects a view defined within such a routine, if the view definition contains a `DEFINER` value of `CURRENT_USER`.

As of MySQL 5.0.16 (when the `DEFINER` and `SQL SECURITY` clauses were implemented), view privileges are checked like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have some privilege for each column in the select list of the definition, and the `SELECT` privilege for each column used elsewhere in the definition. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required at function invocation time can be checked only as it executes: For different invocations, different execution paths within the function might be taken.
- The user who references a view must have appropriate privileges to access it (`SELECT` to select from it, `INSERT` to insert into it, and so forth.)
- When a view has been referenced, privileges for objects accessed by the view are checked against the privileges held by the view `DEFINER` account or invoker, depending on whether the `SQL SECURITY` characteristic is `DEFINER` or `INVOKER`, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function `SQL SECURITY` characteristic is `DEFINER` or `INVOKER`. If the security characteristic is `DEFINER`, the function runs with the privileges of the `DEFINER` account. If the characteristic is `INVOKER`, the function runs with the privileges determined by the view's `SQL SECURITY` characteristic.

Prior to MySQL 5.0.16 (before the `DEFINER` and `SQL SECURITY` clauses were implemented), privileges required for objects used in a view are checked at view creation time.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function `f()`:

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that `f()` contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within `f()` need to be checked when `f()` executes. This might mean that privileges are needed for `p1()` or `p2()`, depending on the execution path within `f()`. Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the `SQL SECURITY` values of the view `v` and the function `f()`.

The `DEFINER` and `SQL SECURITY` clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for `SQL SECURITY DEFINER`. The standard says that the definer of the view, which is the same as the owner of the view's schema, gets applicable privileges on the view (for example, `SELECT`) and may grant them. MySQL has no concept of a schema “owner”, so MySQL adds a clause to identify the definer. The `DEFINER` clause is an extension where the intent is to have what the standard has; that is, a permanent record of who defined the view. This is why the default `DEFINER` value is the account of the view creator.

If you invoke a view that was created before MySQL 5.0.13, it is treated as though it was created with a `SQL SECURITY DEFINER` characteristic and with a `DEFINER` value that is the same as your account. However, because the actual definer is unknown, MySQL issues a warning. To eliminate the warning, it is sufficient to re-create the view so that the view definition includes a `DEFINER` clause.

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. The default algorithm

This documentation is for an older version. If you're
This documentation is for an older version. If you're

is UNDEFINED if no ALGORITHM clause is present. For more information, see Section 18.4.2, “View Processing Algorithms”.

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable.

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts or updates to rows except those for which the WHERE clause in the *select_statement* is true. The WITH CHECK OPTION clause was implemented in MySQL 5.0.2.

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED.

For more information about updatable views and the WITH CHECK OPTION clause, see Section 18.4.3, “Updatable and Insertable Views”, and Section 18.4.4, “The View WITH CHECK OPTION Clause”.

13.1.13 DROP DATABASE Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE drops all tables in the database and deletes the database. Be *very* careful with this statement! To use DROP DATABASE, you need the DROP privilege on the database. DROP SCHEMA is a synonym for DROP DATABASE as of MySQL 5.0.2.

Important

When a database is dropped, user privileges on the database are *not* automatically dropped. See Section 13.7.1.3, “GRANT Syntax”.

IF EXISTS is used to prevent an error from occurring if the database does not exist.

If the default database is dropped, the default database is unset (the DATABASE() function returns NULL).

If you use DROP DATABASE on a symbolically linked database, both the link and the original database are deleted.

DROP DATABASE returns the number of tables that were removed. This corresponds to the number of .frm files removed.

The DROP DATABASE statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with the following extensions.

.BAK	.DAT	.HSH	.MRG
.MYD	.MYI	.TRG	.TRN
.db	.frm	.ibd	.ndb

- All subdirectories with names that consist of two hex digits 00-ff. These are subdirectories used for RAID tables. (These directories are not removed as of MySQL 5.0, when support for RAID tables was

This documentation is for an older version. If you're
This documentation is for an older version. If you're

removed. You should convert any existing `RAID` tables and remove these directories manually before upgrading to MySQL 5.0. See Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”.)

- The `db.opt` file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

Dropping a database does not remove any `TEMPORARY` tables that were created in that database. `TEMPORARY` tables are automatically removed when the session that created them ends. See Temporary Tables.

You can also drop databases with `mysqladmin`. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

13.1.14 DROP FUNCTION Syntax

The `DROP FUNCTION` statement is used to drop stored functions and user-defined functions (UDFs):

- For information about dropping stored functions, see Section 13.1.16, “DROP PROCEDURE and DROP FUNCTION Syntax”.
- For information about dropping user-defined functions, see Section 13.7.3.2, “DROP FUNCTION Syntax”.

13.1.15 DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
```

`DROP INDEX` drops the index named *`index_name`* from the table *`tbl_name`*. This statement is mapped to an `ALTER TABLE` statement to drop the index. See Section 13.1.4, “ALTER TABLE Syntax”.

To drop a primary key, the index name is always `PRIMARY`, which must be specified as a quoted identifier because `PRIMARY` is a reserved word:

```
DROP INDEX `PRIMARY` ON t;
```

13.1.16 DROP PROCEDURE and DROP FUNCTION Syntax

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. As of MySQL 5.0.3, you must have the `ALTER ROUTINE` privilege for the routine. (If the `automatic_sp_privileges` system variable is enabled, that privilege and `EXECUTE` are granted automatically to the routine creator when the routine is created and dropped from the creator when the routine is dropped. See Section 18.2.2, “Stored Routines and MySQL Privileges”.)

The `IF EXISTS` clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with `SHOW WARNINGS`.

Note

`DROP PROCEDURE IF EXISTS` and `DROP FUNCTION IF EXISTS` are not written to the binary log (and thus not replicated) if the stored procedure or function named

This documentation is for an older version. If you're
This documentation is for an older version. If you're

in the DROP statement does not exist on the master. This is a known issue, which is resolved in MySQL 5.1 and later. (Bug #13684)

DROP FUNCTION is also used to drop user-defined functions (see Section 13.7.3.2, "DROP FUNCTION Syntax").

13.1.17 DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

DROP TABLE removes one or more tables. You must have the DROP privilege for each table. All table data and the table definition are *removed*, so *be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name which nonexisting tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important

When a table is dropped, user privileges on the table are *not* automatically dropped. See Section 13.7.1.3, "GRANT Syntax".

Use IF EXISTS to prevent an error from occurring for tables that do not exist. A NOTE is generated for each nonexistent table when using IF EXISTS. See Section 13.7.5.37, "SHOW WARNINGS Syntax".

RESTRICT and CASCADE are permitted to make porting easier. In MySQL 5.0, they do nothing.

Note

DROP TABLE automatically commits the current active transaction, unless you use the TEMPORARY keyword.

The TEMPORARY keyword has the following effects:

- The statement drops only TEMPORARY tables.
- The statement does not end an ongoing transaction.
- No access rights are checked. (A TEMPORARY table is visible only to the session that created it, so no check is necessary.)

Using TEMPORARY is a good way to ensure that you do not accidentally drop a non-TEMPORARY table.

13.1.18 DROP TRIGGER Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. DROP TRIGGER was added in MySQL 5.0.2. Its use requires the SUPER privilege.

Use IF EXISTS to prevent an error from occurring for a trigger that does not exist. A NOTE is generated for a nonexistent trigger when using IF EXISTS. See Section 13.7.5.37, "SHOW WARNINGS Syntax". The IF EXISTS clause was added in MySQL 5.0.32.

Triggers for a table are also dropped if you drop the table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

Prior to MySQL 5.0.10, the table name was required instead of the schema name (*`table_name.trigger_name`*). When upgrading from a previous version of MySQL 5.0 to MySQL 5.0.10 or newer, you must drop all triggers and re-create them. Otherwise, `DROP TRIGGER` does not work for older triggers after the upgrade. See Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”, for a suggested upgrade procedure.

In addition, triggers created in MySQL 5.0.16 or later cannot be dropped following a downgrade to MySQL 5.0.15 or earlier. If you wish to perform such a downgrade, you must also in this case drop all triggers *prior to* the downgrade, and then re-create them afterward.

(For more information about these two issues, see Bug #15921 and Bug #18588.)

13.1.19 DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

`DROP VIEW` removes one or more views. You must have the `DROP` privilege for each view. If any of the views named in the argument list do not exist, MySQL returns an error indicating by name which nonexisting views it was unable to drop, but it also drops all of the views in the list that do exist.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each nonexistent view. See Section 13.7.5.37, “SHOW WARNINGS Syntax”.

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

This statement was added in MySQL 5.0.1.

13.1.20 RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables. The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running.

For example, a table named `old_table` can be renamed to `new_table` as shown here:

```
RENAME TABLE old_table TO new_table;
```

This statement is equivalent to the following `ALTER TABLE` statement:

```
ALTER TABLE old_table RENAME new_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that `tmp_table` does not already exist):

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

MySQL checks the destination table name before checking whether the source table exists. For example, if `new_table` already exists and `old_table` does not, the following statement fails as shown here:

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_mydb |
+----------------+
| table_a        |
+----------------+
1 row in set (0.00 sec)

mysql> RENAME TABLE table_b TO table_a;
ERROR 1050 (42S01): Table 'table_a' already exists
```

As long as two databases are on the same file system, you can use `RENAME TABLE` to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

You can use this method to move all tables from one database to a different one, in effect renaming the database. (MySQL has no single statement to perform this task.)

Foreign keys that point to the renamed table are not automatically updated. In such cases, you must drop and re-create the foreign keys in order for them to function properly.

As of MySQL 5.0.14, `RENAME TABLE` also works for views, as long as you do not try to rename a view into a different database.

Any privileges granted specifically for the renamed table or view are not migrated to the new name. They must be changed manually.

When you execute `RENAME TABLE`, you cannot have any locked tables or active transactions. You must also have the `ALTER` and `DROP` privileges on the original table, and the `CREATE` and `INSERT` privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse rename for all renamed tables to return everything to its original state.

You cannot use `RENAME TABLE` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` with temporary tables.

Like `RENAME TABLE`, `ALTER TABLE ... RENAME` can also be used to move a table to a different database. Regardless of the statement used to perform the rename, if the rename operation would move the table to a database located on a different file system, the success of the outcome is platform specific and depends on the underlying operating system calls used to move the table files.

13.1.21 TRUNCATE TABLE Syntax

```
TRUNCATE [TABLE] tbl_name
```

`TRUNCATE TABLE` empties a table completely. Logically, this is equivalent to a `DELETE` statement that deletes all rows, but there are practical differences under some circumstances.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For an InnoDB table before version 5.0.3, InnoDB processes TRUNCATE TABLE by deleting rows one by one. As of MySQL 5.0.3, row by row deletion is used only if there are any FOREIGN KEY constraints that reference the table. If there are no FOREIGN KEY constraints, InnoDB performs fast truncation by dropping the original table and creating an empty one with the same definition, which is much faster than deleting rows one by one. (When fast truncation is used, it resets any AUTO_INCREMENT counter to zero. From MySQL 5.0.13 on, the AUTO_INCREMENT counter is reset to zero by TRUNCATE TABLE, regardless of whether there is a foreign key constraint.)

In the case that FOREIGN KEY constraints reference the table, InnoDB deletes rows one by one and processes the constraints on each one. If the FOREIGN KEY constraint specifies DELETE CASCADE, rows from the child (referenced) table are deleted, and the truncated table becomes empty. If the FOREIGN KEY constraint does *not* specify CASCADE, the TRUNCATE TABLE statement deletes rows one by one and stops if it encounters a parent row that is referenced by the child, returning this error:

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign
key constraint fails (`test`.`child`, CONSTRAINT `child_ibfk_1`
FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`))
```

This is the same as a DELETE statement with no WHERE clause.

The count of rows affected by TRUNCATE TABLE is accurate only when it is mapped to a DELETE statement.

For other storage engines, TRUNCATE TABLE differs from DELETE in the following ways in MySQL 5.0:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- As of MySQL 5.0.8, truncate operations cause an implicit commit. Before 5.0.8, truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction.
- Truncation operations cannot be performed if the session holds an active table lock.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is “0 rows affected,” which should be interpreted as “no information.”
- As long as the table format file *tbl_name*.frm is valid, the table can be re-created as an empty table with TRUNCATE TABLE, even if the data or index files have become corrupted.
- The table handler does not remember the last used AUTO_INCREMENT value, but starts counting from the beginning. This is true even for MyISAM and InnoDB, which normally do not reuse sequence values.
- Since truncation of a table does not make any use of DELETE, the TRUNCATE TABLE statement does not invoke ON DELETE triggers.

13.2 Data Manipulation Statements

13.2.1 CALL Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

The CALL statement invokes a stored procedure that was defined previously with CREATE PROCEDURE.

As of MySQL 5.0.30, stored procedures that take no arguments can be invoked without parentheses. That is, CALL p() and CALL p are equivalent.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` function; from the C API, call the `mysql_affected_rows()` function.

To get back a value from a procedure using an `OUT` or `INOUT` parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.) For an `INOUT` parameter, initialize its value before passing it to the procedure. The following procedure has an `OUT` parameter that the procedure sets to the current server version, and an `INOUT` value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the `INOUT` parameter. After calling the procedure, the values of the two variables will have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+------------+------------+
| @version   | @increment |
+------------+------------+
| 5.0.25-log | 11         |
+------------+------------+
```

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholder support is available in MySQL 5.0 for `IN` parameters, but not for `OUT` or `INOUT` parameters. To work around this limitation for `OUT` and `INOUT` parameters, to forgo the use of placeholders: Refer to user variables in the `CALL` statement itself and do not specify them in the `EXECUTE` statement:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(@version, @increment)';
mysql> EXECUTE s;
mysql> SELECT @version, @increment;
+-----------------+------------+
| @version        | @increment |
+-----------------+------------+
| 6.0.7-alpha-log | 11         |
+-----------------+------------+
```

To write C programs that use the `CALL` SQL statement to execute stored procedures that produce result sets, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. `CLIENT_MULTI_RESULTS` must also be enabled if `CALL` is used to execute any stored procedure that contains prepared statements. It cannot be determined when such a procedure is loaded whether those statements will produce result sets, so it is necessary to assume that they will.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To process the result of a `CALL` statement executed using `mysql_query()` or `mysql_real_query()`, use a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 20.6.16, “C API Support for Multiple Statement Execution”.

For programs written in a language that provides a MySQL interface, there is no native method for directly retrieving the results of `OUT` or `INOUT` parameters from `CALL` statements. To get the parameter values, pass user-defined variables to the procedure in the `CALL` statement and then execute a `SELECT` statement to produce a result set containing the variable values. To handle an `INOUT` parameter, execute a statement prior to the `CALL` that sets the corresponding user variable to the value to be passed to the procedure.

The following example illustrates the technique (without error checking) for the stored procedure `p` described earlier that has an `OUT` parameter and an `INOUT` parameter:

```
mysql_query(mysql, "SET @increment = 10");
mysql_query(mysql, "CALL p(@version, @increment)");
mysql_query(mysql, "SELECT @version, @increment");
result = mysql_store_result(mysql);
row = mysql_fetch_row(result);
mysql_free_result(result);
```

After the preceding code executes, `row[0]` and `row[1]` contain the values of `@version` and `@increment`, respectively.

13.2.2 DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

For the single-table syntax, the `DELETE` statement deletes rows from *`tbl_name`* and returns a count of the number of deleted rows. This count can be obtained by calling the `ROW_COUNT()` function (see Section 12.13, “Information Functions”). The `WHERE` clause, if given, specifies the conditions that identify which rows to delete. With no `WHERE` clause, all rows are deleted. If the `ORDER BY` clause is specified, the rows are deleted in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, `DELETE` deletes from each *`tbl_name`* the rows that satisfy the conditions. In this case, `ORDER BY` and `LIMIT` cannot be used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

where_condition is an expression that evaluates to true for each row to be deleted. It is specified as described in Section 13.2.8, “SELECT Syntax”.

You cannot delete from a table and select from the same table in a subquery.

You need the DELETE privilege on a table to delete rows from it. You need only the SELECT privilege for any columns that are only read, such as those named in the WHERE clause.

As stated, a DELETE statement with no WHERE clause deletes all rows. A faster way to do this, when you do not need to know the number of deleted rows, is to use TRUNCATE TABLE. However, within a transaction or if you have a lock on the table, TRUNCATE TABLE cannot be used whereas DELETE can. See Section 13.1.21, “TRUNCATE TABLE Syntax”, and Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”.

If you delete the row containing the maximum value for an AUTO_INCREMENT column, the value is reused later for a BDB table, but not for a MyISAM or InnoDB table. If you delete all rows in the table with DELETE FROM *tbl_name* (without a WHERE clause) in autocommit mode, the sequence starts over for all storage engines except InnoDB and MyISAM. There are some exceptions to this behavior for InnoDB tables, as discussed in Section 14.2.3.3, “AUTO_INCREMENT Handling in InnoDB”.

For MyISAM and BDB tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See Section 3.6.9, “Using AUTO_INCREMENT”.

The DELETE statement supports the following modifiers:

- If you specify LOW_PRIORITY, the server delays execution of the DELETE until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).
- For MyISAM tables, if you use the QUICK keyword, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The IGNORE keyword causes MySQL to ignore errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of IGNORE are returned as warnings.

The speed of delete operations may also be affected by factors discussed in Section 8.2.2.3, “Speed of DELETE Statements”.

In MyISAM tables, deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. To reclaim unused space and reduce file sizes, use the OPTIMIZE TABLE statement or the myisamchk utility to reorganize tables. OPTIMIZE TABLE is easier to use, but myisamchk is faster. See Section 13.7.2.5, “OPTIMIZE TABLE Syntax”, and Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

The QUICK modifier affects whether index leaves are merged for delete operations. DELETE QUICK is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

DELETE QUICK is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of QUICK can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed AUTO_INCREMENT column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

3. Delete a block of rows at the low end of the column range using `DELETE QUICK`.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of `QUICK`. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use `DELETE` without `QUICK`, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use `OPTIMIZE TABLE`.

If you are going to delete many rows from a table, it might be faster to use `DELETE QUICK` followed by `OPTIMIZE TABLE`. This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific `LIMIT row_count` option to `DELETE` tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a given `DELETE` statement does not take too much time. You can simply repeat the `DELETE` statement until the number of affected rows is less than the `LIMIT` value.

If the `DELETE` statement includes an `ORDER BY` clause, rows are deleted in the order specified by the clause. This is useful primarily in conjunction with `LIMIT`. For example, the following statement finds rows matching the `WHERE` clause, sorts them by `timestamp_column`, and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

`ORDER BY` may also be useful in some cases to delete rows in an order required to avoid referential integrity violations.

If you are deleting many rows from a large table, you may exceed the lock table size for an `InnoDB` table. To avoid this problem, or simply to minimize the time that the table remains locked, the following strategy (which does not use `DELETE` at all) might be helpful:

1. Select the rows *not* to be deleted into an empty table that has the same structure as the original table:

   ```
   INSERT INTO t_copy SELECT * FROM t WHERE ... ;
   ```

2. Use `RENAME TABLE` to atomically move the original table out of the way and rename the copy to the original name:

   ```
   RENAME TABLE t TO t_old, t_copy TO t;
   ```

3. Drop the original table:

   ```
   DROP TABLE t_old;
   ```

No other sessions can access the tables involved while `RENAME TABLE` executes, so the rename operation is not subject to concurrency problems. See Section 13.1.20, “RENAME TABLE Syntax”.

You can specify multiple tables in a `DELETE` statement to delete rows from one or more tables depending on the particular condition in the `WHERE` clause. However, you cannot use `ORDER BY` or `LIMIT` in a multiple-table `DELETE`. The `table_references` clause lists the tables involved in the join. Its syntax is described in Section 13.2.8.2, “JOIN Syntax”.

For the first multiple-table syntax, only matching rows from the tables listed before the `FROM` clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the `FROM` clause

This documentation is for an older version. If you're
This documentation is for an older version. If you're

(before the USING clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables t1 and t2.

The preceding examples use INNER JOIN, but multiple-table DELETE statements can use other types of join permitted in SELECT statements, such as LEFT JOIN. For example, to delete rows that exist in t1 that have no match in t2, use a LEFT JOIN:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

The syntax permits .* after each *tbl_name* for compatibility with Access.

If you use a multiple-table DELETE statement involving InnoDB tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the ON DELETE capabilities that InnoDB provides to cause the other tables to be modified accordingly.

Note

If you declare an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Table aliases in a multiple-table DELETE should be declared only in the *table_references* part of the statement. Declaration of aliases other than in the *table_references* part should be avoided because that can lead to ambiguous statements that have unexpected results such as deleting rows from the wrong table. This is such a statement:

```
DELETE t1 AS a2 FROM t1 AS a1 INNER JOIN t2 AS a2;
```

For alias references in the list of tables from which to delete rows in a multiple-table delete, the default database is used unless one is specified explicitly. For example, if the default database is db1, the following statement does not work because the unqualified alias reference a2 is interpreted as having a database of db1:

```
DELETE a1, a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

To correctly match an alias that refers to a table outside the default database, you must explicitly qualify the reference with the name of the proper database:

```
DELETE a1, db2.a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

13.2.3 DO Syntax

```
DO expr [, expr] ...
```

`DO` executes the expressions but does not return any results. In most respects, `DO` is shorthand for `SELECT expr, ...`, but has the advantage that it is slightly faster when you do not care about the result.

`DO` is useful primarily with functions that have side effects, such as `RELEASE_LOCK()`.

Example: This `SELECT` statement pauses, but also produces a result set:

```
mysql> SELECT SLEEP(5);
+----------+
| SLEEP(5) |
+----------+
|        0 |
+----------+
1 row in set (5.02 sec)
```

`DO`, on the other hand, pauses without producing a result set.:

```
mysql> DO SLEEP(5);
Query OK, 0 rows affected (4.99 sec)
```

This could be useful, for example in a stored function or trigger, which prohibit statements that produce result sets.

`DO` only executes expressions. It cannot be used in all cases where `SELECT` can be used. For example, `DO id FROM t1` is invalid because it references a table.

13.2.4 HANDLER Syntax

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

The `HANDLER` statement provides direct access to table storage engine interfaces. It is available for `MyISAM` and `InnoDB` tables.

The `HANDLER ... OPEN` statement opens a table, making it accessible using subsequent `HANDLER ... READ` statements. This table object is not shared by other sessions and is not closed until the session calls `HANDLER ... CLOSE` or the session terminates.

If you open the table using an alias, further references to the open table with other `HANDLER` statements must use the alias rather than the table name. If you do not use an alias, but open the table using a table name qualified by the database name, further references must use the unqualified table name. For example, for a table opened using `mydb.mytable`, further references must use `mytable`.

The first `HANDLER ... READ` syntax fetches a row where the index specified satisfies the given values and the `WHERE` condition is met. If you have a multiple-column index, specify the index column values as a

This documentation is for an older version. If you're
This documentation is for an older version. If you're

comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index `my_idx` includes three columns named `col_a`, `col_b`, and `col_c`, in that order. The `HANDLER` statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the `HANDLER` interface to refer to a table's `PRIMARY KEY`, use the quoted identifier `` `PRIMARY` ``:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second `HANDLER ... READ` syntax fetches a row from the table in index order that matches the `WHERE` condition.

The third `HANDLER ... READ` syntax fetches a row from the table in natural row order that matches the `WHERE` condition. It is faster than `HANDLER tbl_name READ index_name` when a full table scan is desired. Natural row order is the order in which rows are stored in a `MyISAM` table data file. This statement works for `InnoDB` tables as well, but there is no such concept because there is no separate data file.

Without a `LIMIT` clause, all forms of `HANDLER ... READ` fetch a single row if one is available. To return a specific number of rows, include a `LIMIT` clause. It has the same syntax as for the `SELECT` statement. See Section 13.2.8, “SELECT Syntax”.

`HANDLER ... CLOSE` closes a table that was opened with `HANDLER ... OPEN`.

There are several reasons to use the `HANDLER` interface instead of normal `SELECT` statements:

- `HANDLER` is faster than `SELECT`:
  - A designated storage engine handler object is allocated for the `HANDLER ... OPEN`. The object is reused for subsequent `HANDLER` statements for that table; it need not be reinitialized for each one.
  - There is less parsing involved.
  - There is no optimizer or query-checking overhead.
  - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are permitted), so the storage engine can use optimizations that `SELECT` does not normally permit.
- `HANDLER` makes it easier to port to MySQL applications that use a low-level `ISAM`-like interface.
- `HANDLER` enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with `SELECT`. The `HANDLER` interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

`HANDLER` is a somewhat low-level statement. For example, it does not provide consistency. That is, `HANDLER ... OPEN` does *not* take a snapshot of the table, and does *not* lock the table. This means that after a `HANDLER ... OPEN` statement is issued, table data can be modified (by the current session or other sessions) and these modifications might be only partially visible to `HANDLER ... NEXT` or `HANDLER ... PREV` scans.

An open handler can be closed and marked for reopen, in which case the handler loses its position in the table. This occurs when both of the following circumstances are true:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Any session executes `FLUSH TABLES` or DDL statements on the handler's table.
- The session in which the handler is open executes non-`HANDLER` statements that use tables.

13.2.5 INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. `INSERT ... SELECT` is discussed further in Section 13.2.5.1, “INSERT ... SELECT Syntax”.

You can use `REPLACE` instead of `INSERT` to overwrite old rows. `REPLACE` is the counterpart to `INSERT IGNORE` in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded. See Section 13.2.7, “REPLACE Syntax”.

`tbl_name` is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- You can provide a comma-separated list of column names following the table name. In this case, a value for each named column must be provided by the `VALUES` list or the `SELECT` statement.
- If you do not specify a list of column names for `INSERT ... VALUES` or `INSERT ... SELECT`, values for every column in the table must be provided by the `VALUES` list or the `SELECT` statement. If you do not know the order of the columns in the table, use `DESCRIBE tbl_name` to find out.
- The `SET` clause indicates the column names explicitly.

Column values can be given in several ways:

- If you are not running in strict SQL mode, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 11.6, “Data Type Default Values”. See also Section 1.8.3.3, “Constraints on Invalid Data”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you want an INSERT statement to generate an error unless you explicitly specify values for all columns that do not have a default value, you should use strict mode. See Section 5.1.7, “Server SQL Modes”.

- Use the keyword DEFAULT to set a column explicitly to its default value. This makes it easier to write INSERT statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete VALUES list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the VALUES list.

  You can also use DEFAULT(*col_name*) as a more general form that can be used in expressions to produce a given column's default value.

- If both the column list and the VALUES list are empty, INSERT creates a row with each column set to its default value:

```
INSERT INTO tbl_name () VALUES();
```

  In strict mode, an error occurs if any column doesn't have a default value. Otherwise, MySQL uses the implicit default value for any column that does not have an explicitly defined default.

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the data type. For example, inserting the string '1999.0e-2' into an INT, FLOAT, DECIMAL(10,6), or YEAR column results in the values 1999, 19.9921, 19.992100, and 1999 being inserted, respectively. The reason the value stored in the INT and YEAR columns is 1999 is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string a valid floating-point value.

  An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for col2 refers to col1, which has previously been assigned:

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

  But the following is not legal, because the value for col1 refers to col2, which is assigned after col1:

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

  One exception involves columns that contain AUTO_INCREMENT values. Because the AUTO_INCREMENT value is generated after other value assignments, any reference to an AUTO_INCREMENT column in the assignment returns a 0.

INSERT statements that use VALUES syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses. The following statement is illegal because the number of values in the list does not match the number of column names:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

VALUE is a synonym for VALUES in this context. Neither implies anything about the number of values lists, and either may be used whether there is a single values list or multiple lists.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The affected-rows value for an INSERT can be obtained using the ROW_COUNT() function (see Section 12.13, “Information Functions”), or the mysql_affected_rows() C API function (see Section 20.6.7.1, “mysql_affected_rows()”).

If you use an INSERT ... VALUES statement with multiple value lists or INSERT ... SELECT, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because Duplicates can be nonzero.) Duplicates indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. Warnings indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting NULL into a column that has been declared NOT NULL. For multiple-row INSERT statements or INSERT INTO ... SELECT statements, the column is set to the implicit default value for the column data type. This is 0 for numeric types, the empty string ('') for string types, and the “zero” value for date and time types. INSERT INTO ... SELECT statements are handled the same way as multiple-row inserts because the server does not examine the result set from the SELECT to see whether it returns a single row. (For a single-row INSERT, no warning occurs when NULL is inserted into a NOT NULL column. Instead, the statement fails with an error.)
- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as '10.34 a' to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to 0.
- Inserting a string into a string column (CHAR, VARCHAR, TEXT, or BLOB) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the mysql_info() function. See Section 20.6.7.35, “mysql_info()”.

If INSERT inserts a row into a table that has an AUTO_INCREMENT column, you can find the value used for that column by using the SQL LAST_INSERT_ID() function. From within the C API, use the mysql_insert_id() function. However, you should note that the two functions do not always behave identically. The behavior of INSERT statements with respect to AUTO_INCREMENT columns is discussed further in Section 12.13, “Information Functions”, and Section 20.6.7.37, “mysql_insert_id()”.

The INSERT statement supports the following modifiers:

- If you use the DELAYED keyword, the server puts the row or rows to be inserted into a buffer, and the client issuing the INSERT DELAYED statement can then continue immediately. If the table is in use, the server holds the rows. When the table is free, the server begins inserting rows, checking periodically to see whether there are any new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See Section 13.2.5.2, “INSERT DELAYED Syntax”.

  DELAYED is ignored with INSERT ... SELECT or INSERT ... ON DUPLICATE KEY UPDATE.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Beginning with MySQL 5.0.42, `DELAYED` is also disregarded for an `INSERT` that uses functions accessing tables or triggers, or that is called from a function or a trigger.

- If you use the `LOW_PRIORITY` keyword, execution of the `INSERT` is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the `INSERT LOW_PRIORITY` statement is waiting. It is possible, therefore, for a client that issues an `INSERT LOW_PRIORITY` statement to wait for a very long time (or even forever) in a read-heavy environment. (This is in contrast to `INSERT DELAYED`, which lets the client continue at once.)

  Note

  `LOW_PRIORITY` should normally not be used with `MyISAM` tables because doing so disables concurrent inserts. See Section 8.11.3, “Concurrent Inserts”.

  If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used. See Section 8.11.3, “Concurrent Inserts”.

  `LOW_PRIORITY` and `HIGH_PRIORITY` affect only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- If you use the `IGNORE` keyword, errors that occur while executing the `INSERT` statement are ignored. For example, without `IGNORE`, a row that duplicates an existing `UNIQUE` index or `PRIMARY KEY` value in the table causes a duplicate-key error and the statement is aborted. With `IGNORE`, the row is discarded and no error occurs. Ignored errors may generate warnings instead, although duplicate-key errors do not.

  Data conversions that would trigger errors abort the statement if `IGNORE` is not specified. With `IGNORE`, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the `mysql_info()` C API function how many rows were actually inserted into the table.

- If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row is performed. The affected-rows value per row is 1 if the row is inserted as a new row and 2 if an existing row is updated. See Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

Inserting into a table requires the `INSERT` privilege for the table. If the `ON DUPLICATE KEY UPDATE` clause is used and a duplicate key causes an `UPDATE` to be performed instead, the statement requires the `UPDATE` privilege for the columns to be updated. For columns that are read but not modified you need only the `SELECT` privilege (such as for a column referenced only on the right hand side of an *`col_name`*=*`expr`* assignment in an `ON DUPLICATE KEY UPDATE` clause).

13.2.5.1 INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

With `INSERT ... SELECT`, you can quickly insert many rows into a table from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for a `INSERT ... SELECT` statements:

- Specify `IGNORE` to ignore rows that would cause duplicate-key violations.
- `DELAYED` is ignored with `INSERT ... SELECT`.
- The target table of the `INSERT` statement may appear in the `FROM` clause of the `SELECT` part of the query. (This was not possible in some older versions of MySQL.) However, you cannot insert into a table and select from the same table in a subquery.

  When selecting from and inserting into a table at the same time, MySQL creates a temporary table to hold the rows from the `SELECT` and then inserts those rows into the target table. However, it remains true that you cannot use `INSERT INTO t ... SELECT ... FROM t` when `t` is a `TEMPORARY` table, because `TEMPORARY` tables cannot be referred to twice in the same statement (see Section B.5.6.2, “TEMPORARY Table Problems”).
- `AUTO_INCREMENT` columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for `INSERT ... SELECT` statements.
- To avoid ambiguous column reference problems when the `SELECT` and the `INSERT` refer to the same table, provide a unique alias for each table used in the `SELECT` part, and qualify column names in that part with the appropriate alias.

In the values part of `ON DUPLICATE KEY UPDATE`, you can refer to columns in other tables, as long as you do not use `GROUP BY` in the `SELECT` part. One side effect is that you must qualify nonunique column names in the values part.

The order in which rows are returned by a `SELECT` statement with no `ORDER BY` clause is not determined. This means that, when using replication, there is no guarantee that such a `SELECT` returns rows in the same order on the master and the slave; this can lead to inconsistencies between them. To prevent this from occurring, you should always write `INSERT ... SELECT` statements that are to be replicated as `INSERT ... SELECT ... ORDER BY` *`column`*. The choice of *`column`* does not matter as long as the same order for returning the rows is enforced on both the master and the slave. See also Section 16.4.1.10, “Replication and LIMIT”.

13.2.5.2 INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The `DELAYED` option for the `INSERT` statement is a MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the `INSERT` to complete. This is a common situation when you use MySQL for logging and you also periodically run `SELECT` and `UPDATE` statements that take a long time to complete.

When a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that `INSERT DELAYED` is slower than a normal `INSERT` if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are

This documentation is for an older version. If you're
This documentation is for an older version. If you're

delayed rows. This means that you should use `INSERT DELAYED` only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate `mysqld` forcibly (for example, with `kill -9`) or if `mysqld` dies unexpectedly, *any queued rows that have not been written to disk are lost*.

There are some constraints on the use of `DELAYED`:

- `INSERT DELAYED` works only with `MyISAM`, `MEMORY`, and `ARCHIVE` tables. For engines that do not support `DELAYED`, an error occurs.
- An error occurs for `INSERT DELAYED` if used with a table that has been locked with `LOCK TABLES` because the insert must be handled by a separate thread, not by the session that holds the lock.
- For `MyISAM` tables, if there are no free blocks in the middle of the data file, concurrent `SELECT` and `INSERT` statements are supported. Under these circumstances, you very seldom need to use `INSERT DELAYED` with `MyISAM`.
- `INSERT DELAYED` should be used only for `INSERT` statements that specify value lists. The server ignores `DELAYED` for `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE` statements.
- Because the `INSERT DELAYED` statement returns immediately, before the rows are inserted, you cannot use `LAST_INSERT_ID()` to get the `AUTO_INCREMENT` value that the statement might generate.
- `DELAYED` rows are not visible to `SELECT` statements until they actually have been inserted.
- `INSERT DELAYED` is treated as a normal `INSERT` if the statement inserts multiple rows and binary logging is enabled.
- `DELAYED` is ignored on slave replication servers, so that `INSERT DELAYED` is treated as a normal `INSERT` on slaves. This is because `DELAYED` could cause the slave to have different data than the master.
- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- `INSERT DELAYED` is not supported for views.

The following describes in detail what happens when you use the `DELAYED` option to `INSERT` or `REPLACE`. In this description, the “thread” is the thread that received an `INSERT DELAYED` statement and “handler” is the thread that handles all `INSERT DELAYED` statements for a particular table.

- When a thread executes a `DELAYED` statement for a table, a handler thread is created to process all `DELAYED` statements for the table, if no such handler already exists.
- The thread checks whether the handler has previously acquired a `DELAYED` lock; if not, it tells the handler thread to do so. The `DELAYED` lock can be obtained even if other threads have a `READ` or `WRITE` lock on the table. However, the handler waits for all `ALTER TABLE` locks or `FLUSH TABLES` statements to finish, to ensure that the table structure is up to date.
- The thread executes the `INSERT` statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client cannot obtain from the server the number of duplicate rows or the `AUTO_INCREMENT` value for the resulting row, because the `INSERT` returns before the insert operation has been completed. (If you use the C API, the `mysql_info()` function does not return anything meaningful, for the same reason.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- Each time that `delayed_insert_limit` rows are written, the handler checks whether any `SELECT` statements are still pending. If so, it permits these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` statements are received within `delayed_insert_timeout` seconds, the handler terminates.
- If more than `delayed_queue_size` rows are pending in a specific handler queue, the thread requesting `INSERT DELAYED` waits until there is room in the queue. This is done to ensure that `mysqld` does not use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with `delayed_insert` in the `Command` column. It is killed if you execute a `FLUSH TABLES` statement or kill it with `KILL *thread_id*`. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new `INSERT` statements from other threads. If you execute an `INSERT DELAYED` statement after this, a new handler thread is created.

  Note that this means that `INSERT DELAYED` statements have higher priority than normal `INSERT` statements if there is an `INSERT DELAYED` handler running. Other update statements have to wait until the `INSERT DELAYED` queue is empty, someone terminates the handler thread (with `KILL *thread_id*`), or someone executes a `FLUSH TABLES`.
- The following status variables provide information about `INSERT DELAYED` statements.

Status Variable	Meaning
`Delayed_insert_threads`	Number of handler threads
`Delayed_writes`	Number of rows written with `INSERT DELAYED`
`Not_flushed_delayed_rows`	Number of rows waiting to be written

  You can view these variables by issuing a `SHOW STATUS` statement or by executing a `mysqladmin extended-status` command.

13.2.5.3 INSERT ... ON DUPLICATE KEY UPDATE Syntax

If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row is performed. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have identical effect:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=c+1;

UPDATE table SET c=c+1 WHERE a=1;
```

The `ON DUPLICATE KEY UPDATE` clause can contain multiple column assignments, separated by commas.

With `ON DUPLICATE KEY UPDATE`, the affected-rows value per row is 1 if the row is inserted as a new row, and 2 if an existing row is updated.

If column `b` is also unique, the `INSERT` is equivalent to this `UPDATE` statement instead:

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If `a=1 OR b=2` matches several rows, only *one* row is updated. In general, you should try to avoid using an `ON DUPLICATE KEY UPDATE` clause on tables with multiple unique indexes.

You can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the `INSERT ... UPDATE` statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in `INSERT ... UPDATE` statements and returns `NULL` otherwise. Example:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
  ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
  ON DUPLICATE KEY UPDATE c=9;
```

If a table contains an `AUTO_INCREMENT` column and `INSERT ... UPDATE` inserts a row, the `LAST_INSERT_ID()` function returns the `AUTO_INCREMENT` value. If the statement updates a row instead, `LAST_INSERT_ID()` is not meaningful. However, you can work around this by using `LAST_INSERT_ID(expr)`. Suppose that `id` is the `AUTO_INCREMENT` column. To make `LAST_INSERT_ID()` meaningful for updates, insert rows as follows:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE id=LAST_INSERT_ID(id), c=3;
```

The `DELAYED` option is ignored when you use `ON DUPLICATE KEY UPDATE`.

13.2.6 LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number LINES]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

The `LOAD DATA INFILE` statement reads rows from a text file into a table at a very high speed. `LOAD DATA INFILE` is the complement of `SELECT ... INTO OUTFILE`. (See Section 13.2.8.1, “SELECT ... INTO Syntax”.) To write data from a table to a file, use `SELECT ... INTO OUTFILE`. To read the file back into a table, use `LOAD DATA INFILE`. The syntax of the `FIELDS` and `LINES` clauses is the same for both statements. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

You can also load data files by using the `mysqlimport` utility; it operates by sending a `LOAD DATA INFILE` statement to the server. The `--local` option causes `mysqlimport` to read data files from the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

client host. You can specify the `--compress` option to get better performance over slow networks if the client and server support the compressed protocol. See Section 4.5.5, “`mysqlimport` — A Data Import Program”.

For more information about the efficiency of `INSERT` versus `LOAD DATA INFILE` and speeding up `LOAD DATA INFILE`, see Section 8.2.2.1, “Speed of INSERT Statements”.

The file name must be given as a literal string. On Windows, specify backslashes in path names as forward slashes or doubled backslashes. As of MySQL 5.0.19, the `character_set_filesystem` system variable controls the interpretation of the file name.

The server uses the character set indicated by the `character_set_database` system variable to interpret the information in the file. `SET NAMES` and the setting of `character_set_client` do not affect interpretation of input. If the contents of the input file use a character set that differs from the default, it is usually preferable to specify the character set of the file by using the `CHARACTER SET` clause, which is available as of MySQL 5.0.38. A character set of `binary` specifies “no conversion.”

`LOAD DATA INFILE` interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you must ensure that it was written with the correct character set. For example, if you write a data file with `mysqldump -T` or by issuing a `SELECT ... INTO OUTFILE` statement in `mysql`, be sure to use a `--default-character-set` option so that output is written in the character set to be used when the file is loaded with `LOAD DATA INFILE`.

Note that it is currently not possible to load data files that use the `ucs2` character set.

If you use `LOW_PRIORITY`, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

If you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while `LOAD DATA` is executing. This option affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

`CONCURRENT` is not replicated. See Section 16.4.1.11, “Replication and LOAD Operations”, for more information.

The `LOCAL` keyword affects expected location of the file and error handling, as described later. `LOCAL` works only if your server and your client both have been configured to permit it. For example, if `mysqld` was started with `--local-infile=0`, `LOCAL` does not work. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

The `LOCAL` keyword affects where the file is expected to be found:

- If `LOCAL` is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

  When using `LOCAL` with `LOAD DATA`, a copy of the file is created in the server's temporary directory. This is *not* the directory determined by the value of `tmpdir` or `slave_load_tmpdir`, but rather the operating system's temporary directory, and is not configurable in the MySQL Server. (Typically the system temporary directory is `/tmp` on Linux systems and `C:\WINDOWS\TEMP` on Windows.) Lack of sufficient space for the copy in this directory can cause the `LOAD DATA LOCAL` statement to fail.

- If `LOCAL` is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If the file name is an absolute path name, the server uses it as given.
- If the file name is a relative path name with one or more leading components, the server searches for the file relative to the server's data directory.
- If a file name with no leading components is given, the server looks for the file in the database directory of the default database.

In the non-`LOCAL` case, these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the file named as `myfile.txt` is read from the database directory of the default database. For example, if `db1` is the default database, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1`, even though the statement explicitly loads the file into a table in the `db2` database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use `LOAD DATA INFILE` on server files, you must have the `FILE` privilege. See Section 6.2.1, “Privileges Provided by MySQL”. For non-`LOCAL` load operations, if the `secure_file_priv` system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

Using `LOCAL` is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the `FILE` privilege to load local files.

`LOCAL` also affects error handling:

- With `LOAD DATA INFILE`, data-interpretation and duplicate-key errors terminate the operation.
- With `LOAD DATA LOCAL INFILE`, data-interpretation and duplicate-key errors become warnings and the operation continues because the server has no way to stop transmission of the file in the middle of the operation. For duplicate-key errors, this is the same as if `IGNORE` is specified. `IGNORE` is explained further later in this section.

The `REPLACE` and `IGNORE` keywords control handling of input rows that duplicate existing rows on unique key values:

- If you specify `REPLACE`, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See Section 13.2.7, “REPLACE Syntax”.
- If you specify `IGNORE`, rows that duplicate an existing row on a unique key value are discarded.
- If you do not specify either option, the behavior depends on whether the `LOCAL` keyword is specified. Without `LOCAL`, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With `LOCAL`, the default behavior is the same as if `IGNORE` is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

To ignore foreign key constraints during the load operation, issue a `SET foreign_key_checks = 0` statement before executing `LOAD DATA`.

If you use `LOAD DATA INFILE` on an empty `MyISAM` table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). Normally, this makes `LOAD DATA INFILE` much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with `ALTER TABLE ... DISABLE KEYS` before loading the file into the table and using `ALTER`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`TABLE ... ENABLE KEYS` to re-create the indexes after loading the file. See Section 8.2.2.1, "Speed of INSERT Statements".

For both the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements, the syntax of the `FIELDS` and `LINES` clauses is the same. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

If you specify a `FIELDS` clause, each of its subclauses (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY`, and `ESCAPED BY`) is also optional, except that you must specify at least one of them.

If you specify no `FIELDS` or `LINES` clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

(Backslash is the MySQL escape character within strings in SQL statements, so to specify a literal backslash, you must specify two backslashes for the value to be interpreted as a single backslash. The escape sequences `'\t'` and `'\n'` specify tab and newline characters, respectively.)

In other words, the defaults cause `LOAD DATA INFILE` to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret characters preceded by the escape character "`\`" as escape sequences. For example, "`\t`", "`\n`", and "`\\`" signify tab, newline, and backslash, respectively. See the discussion of `FIELDS ESCAPED BY` later for the full list of escape sequences.

Conversely, the defaults cause `SELECT ... INTO OUTFILE` to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use "`\`" to escape instances of tab, newline, or "`\`" that occur within field values.
- Write newlines at the ends of lines.

Note

If you have generated the text file on a Windows system, you might have to use `LINES TERMINATED BY '\r\n'` to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the lines you want to read in have a common prefix that you want to ignore, you can use `LINES STARTING BY '*prefix_string*'` to skip over the prefix, *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
```


This documentation is for an older version. If you're

```
  FIELDS TERMINATED BY ','  LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be `("abc",1)` and `("def",2)`. The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA INFILE` will not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
  FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA INFILE` can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotation marks, with an initial line of column names. If the lines in such a file are terminated by carriage return/newline pairs, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

If the input values are not necessarily enclosed within quotation marks, use `OPTIONALLY` before the `ENCLOSED BY` keywords.

Any of the field- or line-handling options can specify an empty string (`''`). If not empty, the `FIELDS [OPTIONALLY] ENCLOSED BY` and `FIELDS ESCAPED BY` values must be a single character. The

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`FIELDS TERMINATED BY`, `LINES STARTING BY`, and `LINES TERMINATED BY` values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a `LINES TERMINATED BY '\r\n'` clause.

To read a file containing jokes that are separated by lines consisting of `%%`, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ''
  LINES TERMINATED BY '\n%%\n' (joke);
```

`FIELDS [OPTIONALLY] ENCLOSED BY` controls quoting of fields. For output (`SELECT ... INTO OUTFILE`), if you omit the word `OPTIONALLY`, all fields are enclosed by the `ENCLOSED BY` character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify `OPTIONALLY`, the `ENCLOSED BY` character is used only to enclose values from columns that have a string data type (such as `CHAR`, `BINARY`, `TEXT`, or `ENUM`):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Occurrences of the `ENCLOSED BY` character within a field value are escaped by prefixing them with the `ESCAPED BY` character. Also note that if you specify an empty `ESCAPED BY` value, it is possible to inadvertently generate output that cannot be read properly by `LOAD DATA INFILE`. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the `ENCLOSED BY` character, if present, is stripped from the ends of field values. (This is true regardless of whether `OPTIONALLY` is specified; `OPTIONALLY` has no effect on input interpretation.) Occurrences of the `ENCLOSED BY` character preceded by the `ESCAPED BY` character are interpreted as part of the current field value.

If the field begins with the `ENCLOSED BY` character, instances of that character are recognized as terminating a field value only if followed by the field or line `TERMINATED BY` sequence. To avoid ambiguity, occurrences of the `ENCLOSED BY` character within a field value can be doubled and are interpreted as a single instance of the character. For example, if `ENCLOSED BY '"'` is specified, quotation marks are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
The ""BIG"" boss    -> The ""BIG"" boss
```

`FIELDS ESCAPED BY` controls how to read or write special characters:

- For input, if the `FIELDS ESCAPED BY` character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using "`\`" for the escape character). The rules for `NULL` handling are described later in this section.

Character	Escape Sequence
`\0`	An ASCII NUL (`X'00'`) character
`\b`	A backspace character
`\n`	A newline (linefeed) character
`\r`	A carriage return character
`\t`	A tab character.
`\Z`	ASCII 26 (Control+Z)
`\N`	NULL

  For more information about "`\`"-escape syntax, see Section 9.1.1, "String Literals".

  If the `FIELDS ESCAPED BY` character is empty, escape-sequence interpretation does not occur.

- For output, if the `FIELDS ESCAPED BY` character is not empty, it is used to prefix the following characters on output:

  - The `FIELDS ESCAPED BY` character
  - The `FIELDS [OPTIONALLY] ENCLOSED BY` character
  - The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
  - ASCII `0` (what is actually written following the escape character is ASCII "`0`", not a zero-valued byte)

  If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

In certain cases, field- and line-handling options interact:

- If `LINES TERMINATED BY` is an empty string and `FIELDS TERMINATED BY` is nonempty, lines are also terminated with `FIELDS TERMINATED BY`.

- If the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values are both empty (`''`), a fixed-row (nondelimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

  `LINES TERMINATED BY` is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to `''`. In this case, the text file must contain all fields for each row.

  Fixed-row format also affects handling of `NULL` values, as described later.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

Fixed-size format does not work if you are using a multibyte character set.

Note

Before MySQL 5.0.6, fixed-row format used the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. However, if the column contained wider values, they were dumped to their full width, leading to the possibility of a “ragged” field holding values of different widths. Using a field wide enough to hold all values in the field prevents this problem. However, data files written before this change was made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 5.0.6 and up. This change also affects data files read by `mysqlimport` and written by `mysqldump --tab`, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

Handling of `NULL` values varies according to the `FIELDS` and `LINES` options in use:

- For the default `FIELDS` and `LINES` values, `NULL` is written as a field value of `\N` for output, and a field value of `\N` is read as `NULL` for input (assuming that the `ESCAPED BY` character is “`\`”).
- If `FIELDS ENCLOSED BY` is not empty, a field containing the literal word `NULL` as its value is read as a `NULL` value. This differs from the word `NULL` enclosed within `FIELDS ENCLOSED BY` characters, which is read as the string `'NULL'`.
- If `FIELDS ESCAPED BY` is empty, `NULL` is written as the word `NULL`.
- With fixed-row format (which is used when `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` are both empty), `NULL` is written as an empty string. This causes both `NULL` values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load `NULL` into a `NOT NULL` column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in Section 11.6, “Data Type Default Values”.

Some cases are not supported by `LOAD DATA INFILE`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA INFILE` cannot interpret the input properly. For example, the following `FIELDS` clause would cause problems:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value causes `LOAD DATA INFILE` to stop reading a field or line too early. This happens because `LOAD DATA INFILE` cannot properly determine where the field or line value ends.

The following example loads all columns of the `persondata` table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

By default, when no column list is provided at the end of the `LOAD DATA INFILE` statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

Before MySQL 5.0.3, the column list must contain only names of columns in the table being loaded, and the `SET` clause is not supported. As of MySQL 5.0.3, the column list can contain either column names or user variables. With user variables, the `SET` clause enables you to perform transformations on their values before assigning the result to columns.

User variables in the `SET` clause can be used in several ways. The following example uses the first input column directly for the value of `t1.column1`, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of `t1.column2`:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

The `SET` clause can be used to supply values not derived from the input file. The following statement sets `column3` to the current date and time:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and `SET` clause is subject to the following restrictions:

- Assignments in the `SET` clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of `SET` assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an `IGNORE` clause are not processed for the column/variable list or `SET` clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, `LOAD DATA` splits it into fields and uses the values according to the column/variable list and the `SET` clause, if they are present. Then the resulting row is inserted into the table. If there are `BEFORE INSERT` or `AFTER INSERT` triggers for the table, they are activated before or after inserting the row, respectively.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in Section 11.6, “Data Type Default Values”.

An empty field value is interpreted different from a missing field:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to `0`.
- For date and time types, the column is set to the appropriate “zero” value for the type. See Section 11.3, “Date and Time Types”.

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an `INSERT` or `UPDATE` statement.

Treatment of empty or incorrect field values differs from that just described if the SQL mode is set to a restrictive value. For example, if `sql_mode` is set to `TRADITIONAL`, conversion of an empty value or a value such as `'x'` for a numeric column results in an error, not conversion to 0. (With `LOCAL` or `IGNORE`, warnings occur rather than errors, even with a restrictive `sql_mode` value, and the row is inserted using the same closest-value behavior used for nonrestrictive SQL modes. This occurs because the server has no way to stop transmission of the file in the middle of the operation.)

`TIMESTAMP` columns are set to the current date and time only if there is a `NULL` value for the column (that is, `\N`) and the column is not declared to permit `NULL` values, or if the `TIMESTAMP` column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

`LOAD DATA INFILE` regards all input as strings, so you cannot use numeric values for `ENUM` or `SET` columns the way you can with `INSERT` statements. All `ENUM` and `SET` values must be specified as strings.

`BIT` values cannot be loaded using binary notation (for example, `b'011010'`). To work around this, specify the values as regular integers and use the `SET` clause to convert them so that MySQL performs a numeric type conversion and loads them into the `BIT` column properly:

```
shell> cat /tmp/bit_test.txt
2
127
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
    -> INTO TABLE bit_test (@var1) SET b = CAST(@var1 AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+----------+
| bin(b+0) |
+----------+
| 10       |
| 1111111  |
+----------+
2 rows in set (0.00 sec)
```

On Unix, if you need `LOAD DATA` to read from a pipe, you can use the following technique (the example loads a listing of the `/` directory into the table `db1.t1`):

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Here you must run the command that generates the data to be loaded and the `mysql` commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe will block until data is read by the `mysql` process.

When the `LOAD DATA INFILE` statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted using the `INSERT` statement (see Section 13.2.5, “INSERT Syntax”), except that `LOAD DATA INFILE` also generates warnings when there are too few or too many fields in the input row.

You can use `SHOW WARNINGS` to get a list of the first `max_error_count` warnings as information about what went wrong. See Section 13.7.5.37, “SHOW WARNINGS Syntax”.

If you are using the C API, you can get information about the statement by calling the `mysql_info()` function. See Section 20.6.7.35, “mysql_info()”.

13.2.7 REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

`REPLACE` works exactly like `INSERT`, except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. See Section 13.2.5, “INSERT Syntax”.

`REPLACE` is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL—that either inserts or *updates*—see Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

Note

`REPLACE` makes sense only if a table has a `PRIMARY KEY` or `UNIQUE` index. Otherwise, it becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the `REPLACE` statement. Any missing columns are set to their default values, just as happens for `INSERT`. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `SET col_name = col_name + 1`, the reference to the column name on the right hand side is treated as `DEFAULT(col_name)`, so the assignment is equivalent to `SET col_name = DEFAULT(col_name) + 1`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To use REPLACE, you must have both the INSERT and DELETE privileges for the table.

The REPLACE statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row REPLACE, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether REPLACE only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the mysql_affected_rows() function.

You cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for REPLACE (and LOAD DATA ... REPLACE):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
   a. Delete from the table the conflicting row that has the duplicate key value
   b. Try again to insert the new row into the table

It is possible that in the case of a duplicate-key error, a storage engine may perform the REPLACE as an update rather than a delete plus insert, but the semantics are the same. There are no user-visible effects other than a possible difference in how the storage engine increments Handler_*xxx* status variables.

13.2.8 SELECT Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries. See Section 13.2.8.3, “UNION Syntax”, and Section 13.2.9, “Subquery Syntax”.

The most commonly used clauses of SELECT statements are these:

- Each *select_expr* indicates a column that you want to retrieve. There must be at least one *select_expr*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in Section 13.2.8.2, “JOIN Syntax”.

- The WHERE clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *where_condition* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause.

  In the WHERE expression, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See Section 9.5, “Expression Syntax”, and Chapter 12, *Functions and Operators*.

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
        -> 2
```

You are permitted to specify DUAL as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
        -> 2
```

DUAL is purely for the convenience of people who require that all SELECT statements should have FROM and possibly other clauses. MySQL may ignore the clauses. MySQL does not require FROM DUAL if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a HAVING clause must come after any GROUP BY clause and before any ORDER BY clause. The exception is that the INTO clause can appear either as shown in the syntax description or immediately following the *select_expr* list. For more information about INTO, see Section 13.2.8.1, “SELECT ... INTO Syntax”.

The list of *select_expr* terms comprises the select list that indicates which columns to retrieve. Terms specify a column or expression or can use *-shorthand:

- A select list consisting only of a single unqualified * can be used as shorthand to select all columns from all tables:

  ```
  SELECT * FROM t1 INNER JOIN t2 ...
  ```

- *tbl_name*.* can be used as a qualified shorthand to select all columns from the named table:

  ```
  SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
  ```

- Use of an unqualified * with other items in the select list may produce a parse error. To avoid this problem, use a qualified *tbl_name*.* reference

  ```
  SELECT AVG(score), t1.* FROM t1 ...
  ```

The following list provides additional information about other SELECT clauses:

- A *select_expr* can be given an alias using AS *alias_name*. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses. For example:

  ```
  SELECT CONCAT(last_name,', ',first_name) AS full_name
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  FROM mytable ORDER BY full_name;
```

The AS keyword is optional when aliasing a *select_expr* with an identifier. The preceding example could have been written like this:

```
SELECT CONCAT(last_name,', ',first_name) full_name
  FROM mytable ORDER BY full_name;
```

However, because the AS is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, columnb is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using AS explicitly when specifying column aliases.

It is not permissible to refer to a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed. See Section B.5.4.4, “Problems with Column Aliases”.

- The FROM *table_references* clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 13.2.8.2, “JOIN Syntax”. For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias] [index_hint]
```

The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see Section 8.9.2, “Index Hints”.

You can use SET max_seeks_for_key=*value* as an alternative way to force MySQL to prefer key scans instead of table scans. See Section 5.1.4, “Server System Variables”.

- You can refer to a table within the default database as *tbl_name*, or as *db_name*.*tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name*.*col_name*, or *db_name*.*tbl_name*.*col_name*. You need not specify a *tbl_name* or *db_name*.*tbl_name* prefix for a column reference unless the reference would be ambiguous. See Section 9.2.1, “Identifier Qualifiers”, for examples of ambiguity that require the more explicit column reference forms.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name* *alias_name*:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;

SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
  ORDER BY region, seed;

SELECT college, region AS r, seed AS s FROM tournament
  ORDER BY r, s;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT college, region, seed FROM tournament
  ORDER BY 2, 3;
```

To sort in reverse order, add the `DESC` (descending) keyword to the name of the column in the `ORDER BY` clause that you are sorting by. The default is ascending order; this can be specified explicitly using the `ASC` keyword.

If `ORDER BY` occurs within a subquery and also is applied in the outer query, the outermost `ORDER BY` takes precedence. For example, results for the following statement are sorted in descending order, not ascending order:

```
(SELECT ... ORDER BY a) ORDER BY a DESC;
```

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use `GROUP BY`, output rows are sorted according to the `GROUP BY` columns as if you had an `ORDER BY` for the same columns. To avoid the overhead of sorting that `GROUP BY` produces, add `ORDER BY NULL`:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- MySQL extends the `GROUP BY` clause so that you can also specify `ASC` and `DESC` after columns named in the clause:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- MySQL extends the use of `GROUP BY` to permit selecting fields that are not mentioned in the `GROUP BY` clause. If you are not getting the results that you expect from your query, please read the description of `GROUP BY` found in Section 12.16, “GROUP BY (Aggregate) Functions”.
- `GROUP BY` permits a `WITH ROLLUP` modifier. See Section 12.16.2, “GROUP BY Modifiers”.
- The `HAVING` clause is applied nearly last, just before items are sent to the client, with no optimization. (`LIMIT` is applied after `HAVING`.)

  A `HAVING` clause can refer to any column or alias named in a *`select_expr`* in the `SELECT` list or in outer subqueries, and to aggregate functions. However, the SQL standard requires that `HAVING` must reference only columns in the `GROUP BY` clause or columns used in aggregate functions. To accommodate both standard SQL and the MySQL-specific behavior of being able to refer columns in the `SELECT` list, MySQL 5.0.2 and up permit `HAVING` to refer to columns in the `SELECT` list, columns in the `GROUP BY` clause, columns in outer subqueries, and to aggregate functions.

  For example, the following statement works in MySQL 5.0.2 but produces an error for earlier versions:

```
mysql> SELECT COUNT(*) FROM t GROUP BY col1 HAVING col1 = 2;
```

  If the `HAVING` clause refers to a column that is ambiguous, a warning occurs. In the following statement, `col2` is ambiguous because it is used as both an alias and a column name:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

  Preference is given to standard SQL behavior, so if a `HAVING` column name is used both in `GROUP BY` and as an aliased column in the output column list, preference is given to the column in the `GROUP BY` column.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Do not use `HAVING` for items that should be in the `WHERE` clause. For example, do not write the following:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

  Write this instead:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- The `HAVING` clause can refer to aggregate functions, which the `WHERE` clause cannot:

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

  (This did not work in some older versions of MySQL.)

- MySQL permits duplicate column names. That is, there can be more than one *`select_expr`* with the same name. This is an extension to standard SQL. Because MySQL also permits `GROUP BY` and `HAVING` to refer to *`select_expr`* values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

  In that statement, both columns have the name `a`. To ensure that the correct column is used for grouping, use different names for each *`select_expr`*.

- MySQL resolves unqualified column or alias references in `ORDER BY` clauses by searching in the *`select_expr`* values, then in the columns of the tables in the `FROM` clause. For `GROUP BY` or `HAVING` clauses, it searches the `FROM` clause before searching in the *`select_expr`* values. (For `GROUP BY` and `HAVING`, this differs from the pre-MySQL 5.0 behavior that used the same rules as for `ORDER BY`.)

- The `LIMIT` clause can be used to constrain the number of rows returned by the `SELECT` statement. `LIMIT` takes one or two numeric arguments, which must both be nonnegative integer constants (except when using prepared statements).

  With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

```
SELECT * FROM tbl LIMIT 5,10;  # Retrieve rows 6-15
```

  To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

  With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
SELECT * FROM tbl LIMIT 5;     # Retrieve first 5 rows
```

  In other words, `LIMIT` *`row_count`* is equivalent to `LIMIT 0,` *`row_count`*.

  For prepared statements, you can use placeholders (supported as of MySQL version 5.0.7). The following statements will return one row from the `tbl` table:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
EXECUTE STMT USING @a;
```

The following statements will return the second to sixth row from the `tbl` table:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

If `LIMIT` occurs within a subquery and also is applied in the outer query, the outermost `LIMIT` takes precedence. For example, the following statement produces two rows, not one:

```
(SELECT ... LIMIT 1) LIMIT 2;
```

- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see Section 8.4.2.4, “Using PROCEDURE ANALYSE”, which describes `ANALYSE`, a procedure that can be used to obtain suggestions for optimal column data types that may help reduce table sizes.

  A `PROCEDURE` clause is not permitted in a `UNION` statement.

- The `SELECT ... INTO` form of `SELECT` enables the query result to be written to a file or stored in variables. For more information, see Section 13.2.8.1, “SELECT ... INTO Syntax”.

- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that permits other transactions to read the examined rows but not to update or delete them. See Section 14.2.8.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”.

Following the `SELECT` keyword, you can use a number of options that affect the operation of the statement. `HIGH_PRIORITY`, `STRAIGHT_JOIN`, and options beginning with `SQL_` are MySQL extensions to standard SQL.

- The `ALL` and `DISTINCT` options specify whether duplicate rows should be returned. `ALL` (the default) specifies that all matching rows should be returned, including duplicates. `DISTINCT` specifies removal of duplicate rows from the result set. It is an error to specify both options. `DISTINCTROW` is a synonym for `DISTINCT`.

- `HIGH_PRIORITY` gives the `SELECT` higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A `SELECT HIGH_PRIORITY` query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

  `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`.

- `STRAIGHT_JOIN` forces the optimizer to join the tables in the order in which they are listed in the `FROM` clause. You can use this to speed up a query if the optimizer joins the tables in nonoptimal order. `STRAIGHT_JOIN` also can be used in the *`table_references`* list. See Section 13.2.8.2, “JOIN Syntax”.

  `STRAIGHT_JOIN` does not apply to any table that the optimizer treats as a `const` or `system` table. Such a table produces a single row, is read during the optimization phase of query execution, and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

references to its columns are replaced with the appropriate column values before query execution proceeds. These tables will appear first in the query plan displayed by `EXPLAIN`. See Section 8.8.1, "Optimizing Queries with EXPLAIN". This exception may not apply to `const` or `system` tables that are used on the `NULL`-complemented side of an outer join (that is, the right-side table of a `LEFT JOIN` or the left-side table of a `RIGHT JOIN`.

- `SQL_BIG_RESULT` or `SQL_SMALL_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set has many rows or is small, respectively. For `SQL_BIG_RESULT`, MySQL directly uses disk-based temporary tables if needed, and prefers sorting to using a temporary table with a key on the `GROUP BY` elements. For `SQL_SMALL_RESULT`, MySQL uses fast temporary tables to store the resulting table instead of using sorting. This should not normally be needed.

- `SQL_BUFFER_RESULT` forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client. This option can be used only for top-level `SELECT` statements, not for subqueries or following `UNION`.

- `SQL_CALC_FOUND_ROWS` tells MySQL to calculate how many rows there would be in the result set, disregarding any `LIMIT` clause. The number of rows can then be retrieved with `SELECT FOUND_ROWS()`. See Section 12.13, "Information Functions".

- The `SQL_CACHE` and `SQL_NO_CACHE` options affect caching of query results in the query cache (see Section 8.10.3, "The MySQL Query Cache"). `SQL_CACHE` tells MySQL to store the result in the query cache if it is cacheable and the value of the `query_cache_type` system variable is `2` or `DEMAND`. With `SQL_NO_CACHE`, the server does not use the query cache. It neither checks the query cache to see whether the result is already cached, nor does it cache the query result. (Due to a limitation in the parser, a space character must precede and follow the `SQL_NO_CACHE` keyword; a nonspace such as a newline causes the server to check the query cache to see whether the result is already cached.)

  For a query that uses `UNION`, subqueries, or views, the following rules apply:

  - `SQL_NO_CACHE` applies if it appears in any `SELECT` in the query.
  - For a cacheable query, `SQL_CACHE` applies if it appears in the first `SELECT` of the query, or in the first `SELECT` of a view referred to by the query.

13.2.8.1 SELECT ... INTO Syntax

The `SELECT ... INTO` form of `SELECT` enables a query result to be stored in variables or written to a file:

- `SELECT ... INTO` *`var_list`* selects column values and stores them into variables.
- `SELECT ... INTO OUTFILE` writes the selected rows to a file. Column and line terminators can be specified to produce a specific output format.
- `SELECT ... INTO DUMPFILE` writes a single row to a file without any formatting.

The `SELECT` syntax description (see Section 13.2.8, "SELECT Syntax") shows the `INTO` clause near the end of the statement. It is also possible to use `INTO` immediately following the *`select_expr`* list.

An `INTO` clause should not be used in a nested `SELECT` because such a `SELECT` must return its result to the outer context.

The `INTO` clause can name a list of one or more variables, which can be user-defined variables, stored procedure or function parameters, or stored program local variables. (Within a prepared `SELECT ... INTO OUTFILE` statement, only user-defined variables are permitted;see Section 13.6.4.2, "Local Variable Scope and Resolution".)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The selected values are assigned to the variables. The number of variables must match the number of columns. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). If it is possible that the statement may retrieve multiple rows, you can use `LIMIT 1` to limit the result set to a single row.

```
SELECT id, data INTO @x, @y FROM test.t1 LIMIT 1;
```

User variable names are not case sensitive. See Section 9.4, “User-Defined Variables”.

The `SELECT ... INTO OUTFILE 'file_name'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, so you must have the `FILE` privilege to use this syntax. *`file_name`* cannot be an existing file, which among other things prevents files such as `/etc/passwd` and database tables from being destroyed. As of MySQL 5.0.19, the `character_set_filesystem` system variable controls the interpretation of the file name.

The `SELECT ... INTO OUTFILE` statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some other host than the server host, you normally cannot use `SELECT ... INTO OUTFILE` since there is no way to write a path to the file relative to the server host's file system.

However, if the MySQL client software is installed on the remote machine, you can instead use a client command such as `mysql -e "SELECT ..." > file_name` to generate the file on the client host.

It is also possible to create the resulting file on a different host other than the server host, if the location of the file on the remote host can be accessed using a network-mapped path on the server's file system. In this case, the presence of `mysql` (or some other MySQL client program) is not required on the target host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`. Column values are dumped using the `binary` character set. In effect, there is no character set conversion. If a result set contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

The syntax for the *`export_options`* part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See Section 13.2.6, “LOAD DATA INFILE Syntax”, for information about the `FIELDS` and `LINES` clauses, including their default values and permissible values.

`FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used when necessary to avoid ambiguity as a prefix that precedes following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII `NUL` (the zero-valued byte; what is actually written following the escape character is ASCII “`0`”, not a zero-valued byte)

The `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters *must* be escaped so that you can read the file back in reliably. ASCII `NUL` is escaped to make it easier to view with some pagers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  FROM test_table;
```

If you use INTO DUMPFILE instead of INTO OUTFILE, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a BLOB value in a file.

Note

Any file created by INTO OUTFILE or INTO DUMPFILE is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run mysqld as root for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.

If the secure_file_priv system variable is set to a nonempty directory name, the file to be written must be located in that directory.

13.2.8.2 JOIN Syntax

MySQL supports the following JOIN syntaxes for the *table_references* part of SELECT statements and multiple-table DELETE and UPDATE statements:

```
table_references:
    escaped_table_reference [, escaped_table_reference] ...

escaped_table_reference:
    table_reference
  | { OJ table_reference }

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [[AS] alias] [index_hint]
  | table_subquery [AS] alias
  | ( table_references )

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor

join_condition:
    ON conditional_expr
  | USING (column_list)

index_hint:
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    USE {INDEX|KEY} [FOR JOIN] (index_list)
  | IGNORE {INDEX|KEY} [FOR JOIN] (index_list)
  | FORCE {INDEX|KEY} [FOR JOIN] (index_list)

index_list:
    index_name [, index_name] ...
```

A table reference is also known as a join expression.

The syntax of *`table_factor`* is extended in comparison with the SQL Standard. The latter accepts only *`table_reference`*, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of *`table_reference`* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, `JOIN`, `CROSS JOIN`, and `INNER JOIN` are syntactic equivalents (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause, `CROSS JOIN` is used otherwise.

In versions of MySQL prior to 5.0.1, parentheses in *`table_references`* were just omitted and all join operations were grouped to the left. In general, parentheses can be ignored in join expressions containing only inner join operations. As of 5.0.1, nested joins are permitted (see Section 8.2.1.9, “Nested Join Optimization”).

Further changes in join processing were made in 5.0.12 to make MySQL more compliant with standard SQL. These charges are described later in this section.

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see Section 8.9.2, “Index Hints”.

The following list describes general factors to take into account when writing joins.

- A table reference can be aliased using *`tbl_name`* `AS` *`alias_name`* or *`tbl_name alias_name`*:

  ```
  SELECT t1.name, t2.salary
    FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

  SELECT t1.name, t2.salary
    FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
  ```

- A *`table_subquery`* is also known as a subquery in the `FROM` clause. Such subqueries *must* include an alias to give the subquery result a table name. A trivial example follows; see also Section 13.2.9.8, “Subqueries in the FROM Clause”.

  ```
  SELECT * FROM (SELECT 1, 2, 3) AS t1;
  ```

- `INNER JOIN` and `,` (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

However, the precedence of the comma operator is less than of `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and so on. If you mix comma joins with the other join types when there is a join condition, an error of the form `Unknown column 'col_name' in 'on clause'` may occur. Information about dealing with this problem is given later in this section.

- The *`conditional_expr`* used with `ON` is any conditional expression of the form that can be used in a `WHERE` clause. Generally, you should use the `ON` clause for conditions that specify how to join tables, and the `WHERE` clause to restrict which rows you want in the result set.
- If there is no matching row for the right table in the `ON` or `USING` part in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

```
SELECT left_tbl.*
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
  WHERE right_tbl.id IS NULL;
```

  This example finds all rows in `left_tbl` with an `id` value that is not present in `right_tbl` (that is, all rows in `left_tbl` with no corresponding row in `right_tbl`). This assumes that `right_tbl.id` is declared `NOT NULL`. See Section 8.2.1.7, “LEFT JOIN and RIGHT JOIN Optimization”.
- The `USING(`*`column_list`*`)` clause names a list of columns that must exist in both tables. If tables `a` and `b` both contain columns `c1`, `c2`, and `c3`, the following join compares corresponding columns from the two tables:

```
a LEFT JOIN b USING (c1,c2,c3)
```

- The `NATURAL [LEFT] JOIN` of two tables is defined to be semantically equivalent to an `INNER JOIN` or a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `RIGHT JOIN` works analogously to `LEFT JOIN`. To keep code portable across databases, it is recommended that you use `LEFT JOIN` instead of `RIGHT JOIN`.
- The `{ OJ ... }` syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

```
SELECT left_tbl.*
  FROM { OJ left_tbl LEFT OUTER JOIN right_tbl ON left_tbl.id = right_tbl.id }
  WHERE right_tbl.id IS NULL;
```

- `STRAIGHT_JOIN` is similar to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Some join examples:

```
SELECT * FROM table1, table2;

SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  LEFT JOIN table3 ON table2.id=table3.id;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Join Processing Changes in MySQL 5.0.12

Beginning with MySQL 5.0.12, natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard. The goal was to align the syntax and semantics of MySQL with respect to `NATURAL JOIN` and `JOIN ... USING` according to SQL:2003. However, these changes in join processing can result in different output columns for some joins. Also, some queries that appeared to work correctly in older versions must be rewritten to comply with the standard.

These changes have five main aspects:

- The way that MySQL determines the result columns of `NATURAL` or `USING` join operations (and thus the result of the entire `FROM` clause).
- Expansion of `SELECT *` and `SELECT tbl_name.*` into a list of selected columns.
- Resolution of column names in `NATURAL` or `USING` joins.
- Transformation of `NATURAL` or `USING` joins into `JOIN ... ON`.
- Resolution of column names in the `ON` condition of a `JOIN ... ON`.

The following list provides more detail about several effects of the 5.0.12 change in join processing. The term “previously” means “prior to MySQL 5.0.12.”

- The columns of a `NATURAL` join or a `USING` join may be different from previously. Specifically, redundant output columns no longer appear, and the order of columns for `SELECT *` expansion may be different from before.

  Consider this set of statements:

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

  Previously, the statements produced this output:

```
+------+------+------+------+
| i    | j    | k    | j    |
+------+------+------+------+
|    1 |    1 |    1 |    1 |
+------+------+------+------+
+------+------+------+------+
| i    | j    | k    | j    |
+------+------+------+------+
|    1 |    1 |    1 |    1 |
+------+------+------+------+
```

  In the first `SELECT` statement, column `j` appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second SELECT statement, column `j` is named in the `USING` clause and should appear only once in the output, not twice. But in both cases, the redundant column is not eliminated. Also, the order of the columns is not correct according to standard SQL.

  Now the statements produce this output:

```
+------+------+------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
```

The redundant column is eliminated and the column order is correct according to standard SQL:

- First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
- Second, columns unique to the first table, in order in which they occur in that table
- Third, columns unique to the second table, in order in which they occur in that table

The single result column that replaces two common columns is defined using the coalesce operation. That is, for two `t1.a` and `t2.a` the resulting single join column `a` is defined as `a = COALESCE(t1.a, t2.a)`, where:

```
COALESCE(x, y) = (CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END)
```

If the join operation is any other join, the result columns of the join consists of the concatenation of all columns of the joined tables. This is the same as previously.

A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-`NULL` column if one of the two columns is always `NULL`. If neither or both columns are `NULL`, both common columns have the same value, so it doesn't matter which one is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a `JOIN`. Suppose that the tables `t1(a,b)` and `t2(a,c)` have the following contents:

```
t1    t2
----  ----
1 x   2 z
2 y   3 w
```

Then:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 | x    | NULL |
|    2 | y    | z    |
+------+------+------+
```

Here column `a` contains the values of `t1.a`.

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+------+------+------+
| a    | c    | b    |
+------+------+------+
|    2 | z    | y    |
|    3 | w    | NULL |
```


This documentation is for an older version. If you're

```
+------+------+------+
```

Here column `a` contains the values of `t2.a`.

Compare these results to the otherwise equivalent queries with `JOIN ... ON`:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    1 | x    | NULL | NULL |
|    2 | y    |    2 | z    |
+------+------+------+------+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    2 | y    |    2 | z    |
| NULL | NULL |    3 | w    |
+------+------+------+------+
```

- Previously, a `USING` clause could be rewritten as an `ON` clause that compares corresponding columns. For example, the following two clauses were semantically identical:

  ```
  a LEFT JOIN b USING (c1,c2,c3)
  a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
  ```

  Now the two clauses no longer are quite the same:

  - With respect to determining which rows satisfy the join condition, both joins remain semantically identical.
  - With respect to determining which columns to display for `SELECT *` expansion, the two joins are not semantically identical. The `USING` join selects the coalesced value of corresponding columns, whereas the `ON` join selects all columns from all tables. For the preceding `USING` join, `SELECT *` selects these values:

    ```
    COALESCE(a.c1,b.c1), COALESCE(a.c2,b.c2), COALESCE(a.c3,b.c3)
    ```

    For the `ON` join, `SELECT *` selects these values:

    ```
    a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
    ```

    With an inner join, `COALESCE(a.c1,b.c1)` is the same as either `a.c1` or `b.c1` because both columns will have the same value. With an outer join (such as `LEFT JOIN`), one of the two columns can be `NULL`. That column will be omitted from the result.

- The evaluation of multi-way natural joins differs in a very important way that affects the result of `NATURAL` or `USING` joins and that can require query rewriting. Suppose that you have three tables `t1(a,b)`, `t2(c,b)`, and `t3(a,c)` that each have one row: `t1(1,2)`, `t2(10,2)`, and `t3(7,10)`. Suppose also that you have this `NATURAL JOIN` on the three tables:

  ```
  SELECT ... FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Previously, the left operand of the second join was considered to be `t2`, whereas it should be the nested join `(t1 NATURAL JOIN t2)`. As a result, the columns of `t3` are checked for common columns only in `t2`, and, if `t3` has common columns with `t1`, these columns are not used as equi-join columns. Thus, previously, the preceding query was transformed to the following equi-join:

```
SELECT ... FROM t1, t2, t3
  WHERE t1.b = t2.b AND t2.c = t3.c;
```

That join is missing one more equi-join predicate `(t1.a = t3.a)`. As a result, it produces one row, not the empty result that it should. The correct equivalent query is this:

```
SELECT ... FROM t1, t2, t3
  WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
```

If you require the same query result in current versions of MySQL as in older versions, rewrite the natural join as the first equi-join.

- Previously, the comma operator (`,`) and `JOIN` both had the same precedence, so the join expression `t1, t2 JOIN t3` was interpreted as `((t1, t2) JOIN t3)`. Now `JOIN` has higher precedence, so the expression is interpreted as `(t1, (t2 JOIN t3))`. This change affects statements that use an `ON` clause, because that clause can refer only to columns in the operands of the join, and the change in precedence changes interpretation of what those operands are.

  Example:

  ```
  CREATE TABLE t1 (i1 INT, j1 INT);
  CREATE TABLE t2 (i2 INT, j2 INT);
  CREATE TABLE t3 (i3 INT, j3 INT);
  INSERT INTO t1 VALUES(1,1);
  INSERT INTO t2 VALUES(1,1);
  INSERT INTO t3 VALUES(1,1);
  SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
  ```

  Previously, the `SELECT` was legal due to the implicit grouping of `t1,t2` as `(t1,t2)`. Now the `JOIN` takes precedence, so the operands for the `ON` clause are `t2` and `t3`. Because `t1.i1` is not a column in either of the operands, the result is an `Unknown column 't1.i1' in 'on clause'` error. To allow the join to be processed, group the first two tables explicitly with parentheses so that the operands for the `ON` clause are `(t1,t2)` and `t3`:

  ```
  SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
  ```

  Alternatively, avoid the use of the comma operator and use `JOIN` instead:

  ```
  SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
  ```

  This change also applies to statements that mix the comma operator with `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and `RIGHT JOIN`, all of which now have higher precedence than the comma operator.

- Previously, the `ON` clause could refer to columns in tables named to its right. Now an `ON` clause can refer only to its operands.

  Example:

  ```
  CREATE TABLE t1 (i1 INT);
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

Previously, the `SELECT` statement was legal. Now the statement fails with an `Unknown column 'i3' in 'on clause'` error because `i3` is a column in `t3`, which is not an operand of the `ON` clause. The statement should be rewritten as follows:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- Resolution of column names in `NATURAL` or `USING` joins is different than previously. For column names that are outside the `FROM` clause, MySQL now handles a superset of the queries compared to previously. That is, in cases when MySQL formerly issued an error that some column is ambiguous, the query now is handled correctly. This is due to the fact that MySQL now treats the common columns of `NATURAL` or `USING` joins as a single column, so when a query refers to such columns, the query compiler does not consider them as ambiguous.

  Example:

  ```
  SELECT * FROM t1 NATURAL JOIN t2 WHERE b > 1;
  ```

  Previously, this query would produce an error `ERROR 1052 (23000): Column 'b' in where clause is ambiguous`. Now the query produces the correct result:

  ```
  +------+------+------+
  | b    | c    | y    |
  +------+------+------+
  |    4 |    2 |    3 |
  +------+------+------+
  ```

  One extension of MySQL compared to the SQL:2003 standard is that MySQL enables you to qualify the common (coalesced) columns of `NATURAL` or `USING` joins (just as previously), while the standard disallows that.

13.2.8.3 UNION Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

`UNION` is used to combine the result from multiple `SELECT` statements into a single result set.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each `SELECT` statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding `SELECT` columns do not match, the types and lengths of the columns in the `UNION` result take into account the values retrieved by all of the `SELECT` statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| bbbbbbbbbb    |
+---------------+
```

(In some earlier versions of MySQL, only the type and length from the first `SELECT` would have been used and the second row would have been truncated to a length of 1.)

The `SELECT` statements are normal select statements, but with the following restrictions:

- Only the last `SELECT` statement can use `INTO OUTFILE`. (However, the entire `UNION` result is written to the file.)
- `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`. If you specify it for the first `SELECT`, it has no effect. If you specify it for any subsequent `SELECT` statements, a syntax error results.

The default behavior for `UNION` is that duplicate rows are removed from the result. The optional `DISTINCT` keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional `ALL` keyword, duplicate-row removal does not occur and the result includes all matching rows from all the `SELECT` statements.

You can mix `UNION ALL` and `UNION DISTINCT` in the same query. Mixed `UNION` types are treated such that a `DISTINCT` union overrides any `ALL` union to its left. A `DISTINCT` union can be produced explicitly by using `UNION DISTINCT` or implicitly by using `UNION` with no following `DISTINCT` or `ALL` keyword.

To apply `ORDER BY` or `LIMIT` to an individual `SELECT`, place the clause inside the parentheses that enclose the `SELECT`:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

However, use of `ORDER BY` for individual `SELECT` statements implies nothing about the order in which the rows appear in the final result because `UNION` by default produces an unordered set of rows. Therefore, the use of `ORDER BY` in this context is typically in conjunction with `LIMIT`, so that it is used to determine the subset of the selected rows to retrieve for the `SELECT`, even though it does not necessarily affect the order of those rows in the final `UNION` result. If `ORDER BY` appears without `LIMIT` in a `SELECT`, it is optimized away because it will have no effect anyway.

To use an `ORDER BY` or `LIMIT` clause to sort or limit the entire `UNION` result, parenthesize the individual `SELECT` statements and place the `ORDER BY` or `LIMIT` after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

A statement without parentheses is equivalent to one parenthesized as just shown.

This kind of `ORDER BY` cannot use column references that include a table name (that is, names in *`tbl_name.col_name`* format). Instead, provide a column alias in the first `SELECT` statement and refer to the alias in the `ORDER BY`. (Alternatively, refer to the column in the `ORDER BY` using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the `ORDER BY` clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an `Unknown column 'a' in 'order clause'` error:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To cause rows in a `UNION` result to consist of the sets of rows retrieved by each `SELECT` one after the other, select an additional column in each `SELECT` to use as a sort column and add an `ORDER BY` following the last `SELECT`:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual `SELECT` results, add a secondary column to the `ORDER BY` clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

Use of an additional column also enables you to determine which `SELECT` each row comes from. Extra columns can provide other identifying information as well, such as a string that indicates a table name.

13.2.9 Subquery Syntax

A subquery is a `SELECT` statement within another statement.

All subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
  WHERE NOT EXISTS
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  (SELECT * FROM t3
   WHERE ROW(5*t2.s1,77)=
    (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
     (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain many of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.

In MySQL, you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and (because subqueries can be used in the `SET` clause) `LOAD DATA INFILE`.

For information about how the optimizer handles subqueries, see Section 8.2.1.14, “Optimizing Subqueries with EXISTS Strategy”. For a discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, see Section C.3, “Restrictions on Subqueries”.

13.2.9.1 The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication that it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` returns a single value (`'abcde'`) that has a data type of `CHAR`, a length of 5, a character set and collation equal to the defaults in effect at `CREATE TABLE` time, and an indication that the value in the column can be `NULL`. Nullability of the value selected by a scalar subquery is not copied because if the subquery result is empty, the result is `NULL`. For the subquery just shown, if `t1` were empty, the result would be `NULL` even though `s2` is `NOT NULL`.

There are a few contexts in which a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, `LIMIT` requires literal integer arguments, and `LOAD DATA INFILE` requires a literal string file name. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct `(SELECT column1 FROM t1)`, imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Then perform a `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is `2` because there is a row in `t2` containing a column `s1` that has a value of `2`.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.9.2 Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *`comparison_operator`* is one of these operators:

```
=  >  <  >=  <=  <>  !=  <=>
```

For example:

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the rows in table `t1` for which the `column1` value is equal to a maximum value in table `t2`:

```
SELECT * FROM t1
  WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table `t1` containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
  WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison of the subquery to a scalar, the subquery must return a scalar. For a comparison of the subquery to a row constructor, the subquery must be a row subquery that returns a row with the same number of values as the row constructor. See Section 13.2.9.5, “Row Subqueries”.

13.2.9.3 Subqueries with ANY, IN, or SOME

Syntax:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where *`comparison_operator`* is one of these operators:

```
=  >  <  >=  <=  <>  !=
```

The `ANY` keyword, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ANY` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(21,14,7)` because there is a value `7` in `t2` that is less than `10`. The expression is `FALSE` if table `t2` contains `(20,10)`, or if table `t2` is empty. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(NULL,NULL,NULL)`.

When used with a subquery, the word `IN` is an alias for `= ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

`IN` and `= ANY` are not synonyms when used with an expression list. `IN` can take an expression list, but `= ANY` cannot. See Section 12.3.2, “Comparison Functions and Operators”.

`NOT IN` is not an alias for `<> ANY`, but for `<> ALL`. See Section 13.2.9.4, “Subqueries with ALL”.

The word `SOME` is an alias for `ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word `SOME` is rare, but this example shows why it might be useful. To most people, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

13.2.9.4 Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word `ALL`, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ALL` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(-5,0,+5)` because `10` is greater than all three values in `t2`. The expression is `FALSE` if table `t2`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

contains `(12,6,NULL,-100)` because there is a single value `12` in table `t2` that is greater than `10`. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(0,NULL,1)`.

Finally, the expression is `TRUE` if table `t2` is empty. So, the following expression is `TRUE` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing `NULL` values* and *empty tables* are “edge cases.” When writing subqueries, always consider whether you have taken those two possibilities into account.

`NOT IN` is an alias for `<> ALL`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.9.5 Row Subqueries

Scalar or column subqueries return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Legal operators for row subquery comparisons are:

```
=  >  <  >=  <=  <>  !=  <=>
```

Here are two examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

For both queries, if the table `t2` contains a single row with `id = 10`, the subquery returns a single row. If this row has `col3` and `col4` values equal to the `col1` and `col2` values of any rows in `t1`, the `WHERE` expression is `TRUE` and each query returns those `t1` rows. If the `t2` row `col3` and `col4` values are not equal the `col1` and `col2` values of any `t1` row, the expression is `FALSE` and the query returns an empty result set. The expression is *unknown* (that is, `NULL`) if the subquery produces no rows. An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

For information about how each operator works for row comparisons, see Section 12.3.2, “Comparison Functions and Operators”.

The expressions `(1,2)` and `ROW(1,2)` are sometimes called *row constructors*. The two are equivalent. The row constructor and the row returned by the subquery must contain the same number of values.

A row constructor is used for comparisons with subqueries that return two or more columns. When a subquery returns a single column, this is regarded as a scalar value and not as a row, so a row constructor

This documentation is for an older version. If you're
This documentation is for an older version. If you're

cannot be used with a subquery that does not return at least two columns. Thus, the following query fails with a syntax error:

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

Row constructors are legal in other contexts. For example, the following two statements are semantically equivalent:

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

Prior to MySQL 5.0.26, only the second of the preceding two expressions could be optimized. (Bug #16081)

The following query answers the request, “find all rows in table `t1` that also exist in table `t2`”:

```
SELECT column1,column2,column3
  FROM t1
  WHERE (column1,column2,column3) IN
         (SELECT column1,column2,column3 FROM t2);
```

For more information about the optimizer and row constructors, see Section 8.2.1.16, “Row Constructor Expression Optimization”

13.2.9.6 Subqueries with EXISTS or NOT EXISTS

If a subquery returns any rows at all, `EXISTS subquery` is `TRUE`, and `NOT EXISTS subquery` is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it makes no difference.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, the `EXISTS` condition is `TRUE`. This is actually an unlikely example because a `[NOT] EXISTS` subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
  WHERE NOT EXISTS (
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    SELECT * FROM cities WHERE NOT EXISTS (
      SELECT * FROM cities_stores
       WHERE cities_stores.city = cities.city
       AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question “does a city exist with a store that is not in `Stores`”? But it is easier to say that a nested `NOT EXISTS` answers the question “is *`x`* `TRUE` for all *`y`*?”

13.2.9.7 Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1
  WHERE column1 = ANY (SELECT column1 FROM t2
                       WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause does not mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the expression as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
  WHERE x.column1 = (SELECT column1 FROM t2 AS x
    WHERE x.column1 = (SELECT column1 FROM t3
      WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

13.2.9.8 Subqueries in the FROM Clause

Subqueries are legal in a `SELECT` statement's `FROM` clause. The actual syntax is:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT ... FROM (subquery) [AS] name ...
```

The `[AS] name` clause is mandatory, because every table in a `FROM` clause must have a name. Any columns in the `subquery` select list must have unique names.

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
  WHERE sb1 > 1;
```

Result: `2, '2', 4.0`.

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
  FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

Subqueries in the `FROM` clause can return a scalar, column, row, or table. Subqueries in the `FROM` clause cannot be correlated subqueries, unless used within the `ON` clause of a `JOIN` operation.

Subqueries in the `FROM` clause are executed even for the `EXPLAIN` statement (that is, derived temporary tables are materialized). This occurs because upper-level queries need information about all tables during the optimization phase, and the table represented by a subquery in the `FROM` clause is unavailable unless the subquery is executed.

It is possible under certain circumstances to modify table data using `EXPLAIN SELECT`. This can occur if the outer query accesses any tables and an inner query invokes a stored function that changes one or more rows of a table. Suppose that there are two tables `t1` and `t2` in database `d1`, created as shown here:

```
mysql> CREATE DATABASE d1;
Query OK, 1 row affected (0.00 sec)

mysql> USE d1;
Database changed

mysql> CREATE TABLE t1 (c1 INT);
Query OK, 0 rows affected (0.15 sec)

mysql> CREATE TABLE t2 (c1 INT);
Query OK, 0 rows affected (0.08 sec)
```

Now we create a stored function `f1` which modifies `t2`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> DELIMITER //
mysql> CREATE FUNCTION f1(p1 INT) RETURNS INT
mysql>   BEGIN
mysql>     INSERT INTO t2 VALUES (p1);
mysql>     RETURN p1;
mysql>   END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

Referencing the function directly in an `EXPLAIN SELECT` does not have any effect on `t2`, as shown here:

```
mysql> SELECT * FROM t2;
Empty set (0.00 sec)

mysql> EXPLAIN SELECT f1(5);
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
|  1 | SIMPLE      | NULL  | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

This is because the `SELECT` statement did not reference any tables, as can be seen in the `table` and `Extra` columns of the output. This is also true of the following nested `SELECT`:

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2;
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
|  1 | PRIMARY     | NULL  | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1249 | Select 2 was reduced during optimization |
+-------+------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

However, if the outer `SELECT` references any tables, the optimizer executes the statement in the subquery as well:

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2;
+----+-------------+------------+--------+---------------+------+---------+------+------+-----------------------
| id | select_type | table      | type   | possible_keys | key  | key_len | ref  | rows | Extra
+----+-------------+------------+--------+---------------+------+---------+------+------+-----------------------
|  1 | PRIMARY     | a1         | system | NULL          | NULL | NULL    | NULL |    0 | const row not found
|  1 | PRIMARY     | <derived2> | system | NULL          | NULL | NULL    | NULL |    1 |
|  2 | DERIVED     | NULL       | NULL   | NULL          | NULL | NULL    | NULL | NULL | No tables used
+----+-------------+------------+--------+---------------+------+---------+------+------+-----------------------
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+
| c1   |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

This also means that an `EXPLAIN SELECT` statement such as the one shown here may take a long time to execute because the `BENCHMARK()` function is executed once for each row in `t1`:

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

13.2.9.9 Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

  ```
  ERROR 1235 (ER_NOT_SUPPORTED_YET)
  SQLSTATE = 42000
  Message = "This version of MySQL doesn't yet support
  'LIMIT & IN/ALL/ANY/SOME subquery'"
  ```

  This means that MySQL does not support statements of the following form:

  ```
  SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
  ```

- Incorrect number of columns from subquery:

  ```
  ERROR 1241 (ER_OPERAND_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  This error occurs in cases like this:

  ```
  SELECT (SELECT column1, column2 FROM t2) FROM t1;
  ```

  You may use a subquery that returns multiple columns, if the purpose is row comparison. In other contexts, the subquery must be a scalar operand. See Section 13.2.9.5, “Row Subqueries”.

- Incorrect number of rows from subquery:

  ```
  ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
  SQLSTATE = 21000
  Message = "Subquery returns more than 1 row"
  ```

  This error occurs for statements where the subquery must return at most one row but returns multiple rows. Consider the following example:

  ```
  SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
  ```

  If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

  ```
  SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

  This error occurs in cases such as the following, which attempts to modify a table and select from the same table in the subquery:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

  You can use a subquery for assignment within an `UPDATE` statement because subqueries are legal in `UPDATE` and `DELETE` statements as well as in `SELECT` statements. However, you cannot use the same table (in this case, table `t1`) for both the subquery `FROM` clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For nontransactional storage engines, data modifications made before the error was encountered are preserved.

13.2.9.10 Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

  Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See Section 13.2.9.11, “Rewriting Subqueries as Joins”.

- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

  Instead of this query:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
                AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.
- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.
- Use `= ANY` rather than `EXISTS`.
- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

```
SELECT * FROM t1
  WHERE t1.col_name = (SELECT a FROM t2 WHERE b = some_const);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation. See Section 12.13, “Information Functions”.

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is uncorrelated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` or `index_subquery`):

```
... IN (SELECT indexed_column FROM single_table ...)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

```
value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
```

For example, this `WHERE` clause:

```
WHERE 5 > ALL (SELECT x FROM t)
```

might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

See also MySQL Internals: How MySQL Transforms Subqueries.

13.2.9.11 Rewriting Subqueries as Joins

Sometimes there are other ways to test membership in a set of values than by using a subquery. Also, on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the `IN()` construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better—a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported. See Section 13.2.2, “DELETE Syntax”, and Section 13.2.10, “UPDATE Syntax”.

13.2.10 UPDATE Syntax

Single-table syntax:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

For the single-table syntax, the `UPDATE` statement updates columns of existing rows in the named table with new values. The `SET` clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value. The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated. If the `ORDER BY` clause is specified, the rows are updated in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, `UPDATE` updates rows in each table named in *`table_references`* that satisfy the conditions. Each matching row is updated once, even if it matches the conditions multiple times. For multiple-table syntax, `ORDER BY` and `LIMIT` cannot be used.

`where_condition` is an expression that evaluates to true for each row to be updated. For expression syntax, see Section 9.5, “Expression Syntax”.

`table_references` and *`where_condition`* are is specified as described in Section 13.2.8, “SELECT Syntax”.

You need the `UPDATE` privilege only for columns referenced in an `UPDATE` that are actually updated. You need only the `SELECT` privilege for any columns that are read but not modified.

The `UPDATE` statement supports the following modifiers:

- With the `LOW_PRIORITY` keyword, execution of the `UPDATE` is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- With the `IGNORE` keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur on a unique key value are not updated. Rows updated to values that would cause data conversion errors are updated to the closest valid values instead.

If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. For example, the following statement sets `col1` to one more than its current value:

```
UPDATE t1 SET col1 = col1 + 1;
```

The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

Single-table `UPDATE` assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared `NOT NULL` by setting to `NULL`, an error occurs if strict SQL mode is enabled; otherwise, the column is set to the implicit default value for the column data type and the warning count is incremented. The implicit default value is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. See Section 11.6, “Data Type Default Values”.

`UPDATE` returns the number of rows that were actually changed. The `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

You can use `LIMIT row_count` to restrict the scope of the `UPDATE`. A `LIMIT` clause is a rows-matched restriction. The statement stops as soon as it has found *`row_count`* rows that satisfy the `WHERE` clause, whether or not they actually were changed.

If an `UPDATE` statement includes an `ORDER BY` clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table `t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform `UPDATE` operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table `UPDATE`. The *`table_references`* clause lists the tables involved in the join. Its syntax is described in Section 13.2.8.2, “JOIN Syntax”. Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table `UPDATE` statements can use any type of join permitted in `SELECT` statements, such as `LEFT JOIN`.

If you use a multiple-table `UPDATE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the `ON UPDATE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly. See Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”.

You cannot update a table and select from the same table in a subquery.

Index hints (see Section 8.9.2, “Index Hints”) are accepted but ignored for `UPDATE` statements.

13.3 MySQL Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as `SET autocommit`, `START TRANSACTION`, `COMMIT`, and `ROLLBACK`. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”. Beginning with MySQL 5.0, XA transaction support is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

available, which enables MySQL to participate in distributed transactions as well. See Section 13.3.7, “XA Transactions”.

13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION [WITH CONSISTENT SNAPSHOT]
BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

These statements provide control over use of transactions:

- `START TRANSACTION` or `BEGIN` start a new transaction
- `COMMIT` commits the current transaction, making its changes permanent
- `ROLLBACK` rolls back the current transaction, canceling its changes
- `SET autocommit` disables or enables the default autocommit mode for the current session

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent.

To disable autocommit mode implicitly for a single series of statements, use the `START TRANSACTION` statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With `START TRANSACTION`, autocommit remains disabled until you end the transaction with `COMMIT` or `ROLLBACK`. The autocommit mode then reverts to its previous state.

You can also begin a transaction like this:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

The `WITH CONSISTENT SNAPSHOT` option starts a consistent read for storage engines that are capable of it. This applies only to `InnoDB`. The effect is the same as issuing a `START TRANSACTION` followed by a `SELECT` from any `InnoDB` table. See Section 14.2.8.4, “Consistent Nonlocking Reads”. The `WITH CONSISTENT SNAPSHOT` option does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation level is one that permits consistent read (`REPEATABLE READ` or `SERIALIZABLE`).

Important

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a `START TRANSACTION` statement from the client. See Chapter 20, *Connectors and APIs*, or the documentation for your API, for more information.

To disable autocommit mode explicitly, use the following statement:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SET autocommit=0;
```

After disabling autocommit mode by setting the `autocommit` variable to zero, changes to transaction-safe tables (such as those for `InnoDB`, `BDB`, or `NDBCLUSTER`) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

`autocommit` is a session variable and must be set for each session. To disable autocommit mode for each new connection, see the description of the `autocommit` system variable at Section 5.1.4, “Server System Variables”.

`BEGIN` and `BEGIN WORK` are supported as aliases of `START TRANSACTION` for initiating a transaction. `START TRANSACTION` is standard SQL syntax and is the recommended way to start an ad-hoc transaction.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not begin a transaction. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”.

Note

Within all stored programs (stored procedures and functions, and triggers), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. Begin a transaction in this context with `START TRANSACTION` instead.

Beginning with MySQL 5.0.3, the optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the `completion_type` system variable determines the default completion behavior. See Section 5.1.4, “Server System Variables”.

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client session after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` system variable is set to cause chaining or release completion by default.

Beginning a transaction causes any pending transaction to be committed. See Section 13.3.3, “Statements That Cause an Implicit Commit”, for more information.

Beginning a transaction also causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as `InnoDB` and `BDB`), and the transaction isolation level is not `SERIALIZABLE`, it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use `SET TRANSACTION ISOLATION LEVEL` to set the isolation level to `SERIALIZABLE` on a per-transaction basis as necessary.)
- If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If you issue a `ROLLBACK` statement after updating a nontransactional table within a transaction, an `ER_WARNING_NOT_COMPLETE_ROLLBACK` warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon `COMMIT`. Transactions that are rolled back are not logged. (**Exception**: Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that modifications to the nontransactional tables are replicated.) See Section 5.4.3, "The Binary Log".

You can change the isolation level for transactions with the `SET TRANSACTION` statement. See Section 13.3.6, "SET TRANSACTION Syntax".

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the session, not only for explicit rollbacks performed with the `ROLLBACK` statement but also for implicit rollbacks.

Note

Beginning with MySQL 5.0.84, `BEGIN`, `COMMIT`, and `ROLLBACK` are no longer affected by `--replicate-do-db` or `--replicate-ignore-db` rules. (Bug #43263)

13.3.2 Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a `ROLLBACK` statement.

13.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a `COMMIT` before executing the statement.

- **Data definition language (DDL) statements that define or modify database objects.** `ALTER TABLE`, `CREATE INDEX`, `DROP INDEX`, `DROP TABLE`, `RENAME TABLE`.

  `CREATE TABLE` and `DROP TABLE` statements do not commit a transaction if the `TEMPORARY` keyword is used. (This does not apply to other operations on temporary tables such as `ALTER TABLE` and `CREATE INDEX`, which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back, which means that the use of such statements causes transactional atomicity to be violated. For example, if you use `CREATE TEMPORARY TABLE` and then roll back the transaction, the table remains in existence.

  The `CREATE TABLE` statement in `InnoDB` is processed as a single transaction. This means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.

  Beginning with MySQL 5.0.8, `CREATE TABLE`, `CREATE DATABASE DROP DATABASE`, and `TRUNCATE TABLE` cause an implicit commit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Beginning with MySQL 5.0.13, `ALTER PROCEDURE`, `CREATE PROCEDURE`, and `DROP PROCEDURE` cause an implicit commit.

Also beginning with MySQL 5.0.13, `ALTER FUNCTION`, `CREATE FUNCTION` and `DROP FUNCTION` cause an implicit commit when used with stored functions, but not with user-defined functions. (`ALTER FUNCTION` can only be used with stored functions.)

Beginning with MySQL 5.0.15, `ALTER VIEW`, `CREATE TRIGGER`, `CREATE VIEW`, `DROP TRIGGER`, and `DROP VIEW` cause an implicit commit.

- **Statements that implicitly use or modify tables in the `mysql` database.** Beginning with MySQL 5.0.15, `CREATE USER`, `DROP USER`, and `RENAME USER` cause an implicit commit.
- **Transaction-control and locking statements.** `BEGIN`, `LOCK TABLES`, `SET autocommit = 1` (if the value is not already 1), `START TRANSACTION`, `UNLOCK TABLES`.

  `UNLOCK TABLES` commits a transaction only if any tables currently have been locked with `LOCK TABLES`. This does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table-level locks.

  Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a `START TRANSACTION` statement or one of its synonyms.

  Statements that cause an implicit commit cannot be used in an XA transaction while the transaction is in an `ACTIVE` state.

  The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not cause an implicit commit. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”.

- **Data loading statements.** `LOAD MASTER DATA`, `LOAD DATA INFILE`. Before MySQL 5.0.26, `LOAD DATA INFILE` caused an implicit commit for all storage engines. As of MySQL 5.0.26, it causes an implicit commit only for tables using the `NDB` storage engine. For more information, see Bug #11151.

13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT, and Syntax

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

`InnoDB` supports the SQL statements `SAVEPOINT` and `ROLLBACK TO SAVEPOINT`. Starting from MySQL 5.0.3, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK` are supported as well.

The `SAVEPOINT` statement sets a named transaction savepoint with a name of *`identifier`*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The `ROLLBACK TO SAVEPOINT` statement rolls back a transaction to the named savepoint without terminating the transaction. (The `SAVEPOINT` keyword is optional as of MySQL 5.0.3.) Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but `InnoDB` does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the `ROLLBACK TO SAVEPOINT` statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

The `RELEASE SAVEPOINT` statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a `COMMIT`, or a `ROLLBACK` that does not name a savepoint.

Beginning with MySQL 5.0.17, a new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

13.3.5 LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail later in this section.

`LOCK TABLES` explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or (as of MySQL 5.0.6) views. You must have the `LOCK TABLES` privilege, and the `SELECT` privilege for each object to be locked.

For view locking, `LOCK TABLES` adds all base tables used in the view to the set of tables to be locked and locks them automatically. If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly, as described in Section 13.3.5.2, “LOCK TABLES and Triggers”.

`UNLOCK TABLES` explicitly releases any table locks held by the current session. `LOCK TABLES` implicitly releases any table locks held by the current session before acquiring new locks.

Another use for `UNLOCK TABLES` is to release the global read lock acquired with the `FLUSH TABLES WITH READ LOCK` statement, which enables you to lock all tables in all databases. See Section 13.7.6.2, “FLUSH Syntax”. (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. The session holding the lock, even a read lock, can perform table-level operations such as `DROP TABLE`. Truncate operations are not transaction-safe, so an error occurs if the session attempts one during an active transaction or while holding a table lock.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following discussion applies only to non-`TEMPORARY` tables. `LOCK TABLES` is permitted (but ignored) for a `TEMPORARY` table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

For information about other conditions on the use of `LOCK TABLES` and statements that cannot be used while `LOCK TABLES` is in effect, see Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

Rules for Lock Acquisition

To acquire table locks within the current session, use the `LOCK TABLES` statement. The following lock types are available:

`READ [LOCAL]` lock:

- The session that holds the lock can read the table (but not write it).
- Multiple sessions can acquire a `READ` lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a `READ` lock.
- The `LOCAL` modifier enables nonconflicting `INSERT` statements (concurrent inserts) by other sessions to execute while the lock is held. (See Section 8.11.3, “Concurrent Inserts”.) However, `READ LOCAL` cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For `InnoDB` tables, `READ LOCAL` is the same as `READ` as of MySQL 5.0.13. (Before that, `READ LOCAL` essentially does nothing: It does not lock the table at all, so for `InnoDB` tables, the use of `READ LOCAL` is deprecated because a plain consistent-read `SELECT` does the same thing, and no locks are needed.)

`[LOW_PRIORITY] WRITE` lock:

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.
- Lock requests for the table by other sessions block while the `WRITE` lock is held.
- The `LOW_PRIORITY` modifier affects lock scheduling if the `WRITE` lock request must wait, as described later.

If the `LOCK TABLES` statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single `LOCK TABLES` statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access `t2` because it was not locked in the `LOCK TABLES` statement:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Tables in the INFORMATION_SCHEMA database are an exception. They can be accessed without being locked explicitly even while a session holds table locks obtained with LOCK TABLES.

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

The error occurs for the first INSERT because there are two references to the same name for a locked table. The second INSERT succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

WRITE locks normally have higher priority than READ locks to ensure that updates are processed as soon as possible. This means that if one session obtains a READ lock and then another session requests a WRITE lock, subsequent READ lock requests wait until the session that requested the WRITE lock has obtained the lock and released it. A request for a LOW_PRIORITY WRITE lock, by contrast, permits subsequent READ lock requests by other sessions to be satisfied first if they occur while the LOW_PRIORITY WRITE request is waiting. You should use LOW_PRIORITY WRITE locks only if you are sure that eventually there will be a time when no sessions have a READ lock. For InnoDB tables in transactional mode (autocommit = 0), a waiting LOW_PRIORITY WRITE lock acts like a regular WRITE lock and causes subsequent READ lock requests to wait.

LOCK TABLES acquires locks as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.
2. If a table is to be locked with a read and a write lock, put the write lock request before the read lock request.
3. Lock one table at a time until the session gets all locks.

This policy ensures that table locking is deadlock free. There are, however, other things you need to be aware of about this policy: If you are using a LOW_PRIORITY WRITE lock for a table, it means only that MySQL waits for this particular lock until there are no other sessions that want a READ lock. When the session has gotten the WRITE lock and is waiting to get the lock for the next table in the lock table list, all other sessions wait for the WRITE lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transaction-safe tables.

Rules for Lock Release

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

- A session can release its locks explicitly with `UNLOCK TABLES`.
- If a session issues a `LOCK TABLES` statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
- If a session begins a transaction (for example, with `START TRANSACTION`), an implicit `UNLOCK TABLES` is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see Section 13.3.5.1, “Interaction of Table Locking and Transactions”.)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks will no longer be in effect. In addition, if the client had an active transaction, the server rolls back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction will have been lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See Section 20.6.15, “Controlling Automatic Reconnection Behavior”.

Note

If you use `ALTER TABLE` on a locked table, it may become unlocked. For example, if you attempt a second `ALTER TABLE` operation, the result may be an error `Table 'tbl_name' was not locked with LOCK TABLES`. To handle this, lock the table again prior to the second alteration. See also Section B.5.6.1, “Problems with ALTER TABLE”.

13.3.5.1 Interaction of Table Locking and Transactions

`LOCK TABLES` and `UNLOCK TABLES` interact with the use of transactions as follows:

- `LOCK TABLES` is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
- `UNLOCK TABLES` implicitly commits any active transaction, but only if `LOCK TABLES` has been used to acquire table locks. For example, in the following set of statements, `UNLOCK TABLES` releases the global read lock but does not commit the transaction because no table locks are in effect:

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
SELECT ... ;
UNLOCK TABLES;
```

- Beginning a transaction (for example, with `START TRANSACTION`) implicitly commits any current transaction and releases existing table locks.
- `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits. For example, `START TRANSACTION` does not release the global read lock. See Section 13.7.6.2, “FLUSH Syntax”.
- Other statements that implicitly cause transactions to be committed do not release existing table locks. For a list of such statements, see Section 13.3.3, “Statements That Cause an Implicit Commit”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The correct way to use `LOCK TABLES` and `UNLOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

  When you call `LOCK TABLES`, `InnoDB` internally takes its own table lock, and MySQL takes its own table lock. `InnoDB` releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call `UNLOCK TABLES`. You should not have `autocommit = 1`, because then `InnoDB` releases its internal table lock immediately after the call of `LOCK TABLES`, and deadlocks can very easily happen. `InnoDB` does not acquire the internal table lock at all if `autocommit = 1`, to help old applications avoid unnecessary deadlocks.

- `ROLLBACK` does not release table locks.

13.3.5.2 LOCK TABLES and Triggers

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly:

- The locks are taken as the same time as those acquired explicitly with the `LOCK TABLES` statement.
- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.
- If a table is locked explicitly for reading with `LOCK TABLES`, but needs to be locked for writing because it might be modified within a trigger, a write lock is taken rather than a read lock. (That is, an implicit write lock needed due to the table's appearance within a trigger causes an explicit read lock request for the table to be converted to a write lock request.)

Suppose that you lock two tables, `t1` and `t2`, using this statement:

```
LOCK TABLES t1 WRITE, t2 READ;
```

If `t1` or `t2` have any triggers, tables used within the triggers will also be locked. Suppose that `t1` has a trigger defined like this:

```
CREATE TRIGGER t1_a_ins AFTER INSERT ON t1 FOR EACH ROW
BEGIN
  UPDATE t4 SET count = count+1
      WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);
  INSERT INTO t2 VALUES(1, 2);
END;
```

The result of the `LOCK TABLES` statement is that `t1` and `t2` are locked because they appear in the statement, and `t3` and `t4` are locked because they are used within the trigger:

- `t1` is locked for writing per the `WRITE` lock request.
- `t2` is locked for writing, even though the request is for a `READ` lock. This occurs because `t2` is inserted into within the trigger, so the `READ` request is converted to a `WRITE` request.
- `t3` is locked for reading because it is only read from within the trigger.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `t4` is locked for writing because it might be updated within the trigger.

13.3.5.3 Table-Locking Restrictions and Conditions

You can safely use `KILL` to terminate a session that is waiting for a table lock. See Section 13.7.6.3, “KILL Syntax”.

You should *not* lock any tables that you are using with `INSERT DELAYED`. An `INSERT DELAYED` in this case results in an error because the insert must be handled by a separate thread, not by the session which holds the lock.

`LOCK TABLES` and `UNLOCK TABLES` cannot be used within stored programs.

Normally, you do not need to lock tables, because all single `UPDATE` statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of `MyISAM` tables, it is much faster to lock the tables you are going to use. Locking `MyISAM` tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until `UNLOCK TABLES` is called. Normally, the key cache is flushed after each SQL statement.

  The downside to locking the tables is that no session can update a `READ`-locked table (including the one holding the lock) and no session can access a `WRITE`-locked table other than the one holding the lock.

- If you are using tables for a nontransactional storage engine, you must use `LOCK TABLES` if you want to ensure that no other session modifies the tables between a `SELECT` and an `UPDATE`. The example shown here requires `LOCK TABLES` to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

  Without `LOCK TABLES`, it is possible that another session might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

You can avoid using `LOCK TABLES` in many cases by using relative updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` function. See Section 1.8.2.3, “Transactions and Atomic Operations”.

You can also avoid locking tables in some cases by using the user-level advisory lock functions `GET_LOCK()` and `RELEASE_LOCK()`. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See Section 12.15, “Miscellaneous Functions”.

See Section 8.11.1, “Internal Locking Methods”, for more information on locking policy.

13.3.6 SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
  {
       REPEATABLE READ
     | READ COMMITTED
     | READ UNCOMMITTED
     | SERIALIZABLE
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  }
```

This statement sets the transaction isolation level, used for operations on `InnoDB` tables.

Scope of the Isolation Level

You can set the isolation level globally, for the current session, or for the next transaction:

- With the `GLOBAL` keyword, the statement sets the default transaction level globally for all subsequent sessions. Existing sessions are unaffected.
- With the `SESSION` keyword, the statement sets the default transaction level for all subsequent transactions performed within the current session.
- Without any `SESSION` or `GLOBAL` keyword, the statement sets the isolation level for the next (not started) transaction performed within the current session. Subsequent transactions revert to using the `SESSION` isolation level.

A change to the global default isolation level requires the `SUPER` privilege. Any session is free to change its session isolation level (even in the middle of a transaction), or the isolation level for its next transaction.

To set the global default isolation level at server startup, use the `--transaction-isolation=level` option to `mysqld` on the command line or in an option file. Values of *`level`* for this option use dashes rather than spaces, so the permissible values are `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. For example, to set the default isolation level to `REPEATABLE READ`, use these lines in the `[mysqld]` section of an option file:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
```

It is possible to check or set the global and session transaction isolation levels at runtime by using the `tx_isolation` system variable:

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
SET GLOBAL tx_isolation='REPEATABLE-READ';
SET SESSION tx_isolation='SERIALIZABLE';
```

Details and Usage of Isolation Levels

`InnoDB` supports each of the transaction isolation levels described here using different locking strategies. You can enforce a high degree of consistency with the default `REPEATABLE READ` level, for operations on crucial data where ACID compliance is important. Or you can relax the consistency rules with `READ COMMITTED` or even `READ UNCOMMITTED`, in situations such as bulk reporting where precise consistency and repeatable results are less important than minimizing the amount of overhead for locking. `SERIALIZABLE` enforces even stricter rules than `REPEATABLE READ`, and is used mainly in specialized situations, such as with XA transactions and for troubleshooting issues with concurrency and deadlocks.

For full information about how these isolation levels work with `InnoDB` transactions, see Section 14.2.8, “InnoDB Transaction Model and Locking”. In particular, for additional information about `InnoDB` record-level locks and how it uses them to execute various types of statements, see Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks” and Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”.

The following list describes how MySQL supports the different transaction levels. The list goes from the most commonly used level to the least used.

- `REPEATABLE READ`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This is the default isolation level for `InnoDB`. For consistent reads, there is an important difference from the `READ COMMITTED` isolation level: All consistent reads within the same transaction read the snapshot established by the first read. This convention means that if you issue several plain (nonlocking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See Section 14.2.8.4, “Consistent Nonlocking Reads”.

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it. For other search conditions, `InnoDB` locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range.

- `READ COMMITTED`

  A somewhat Oracle-like isolation level with respect to consistent (nonlocking) reads: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See Section 14.2.8.4, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `InnoDB` locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked records. For `UPDATE` and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition (such as `WHERE id = 100`), or a range-type search condition (such as `WHERE id > 100`). For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it. For range-type searches, `InnoDB` locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range. This is necessary because “phantom rows” must be blocked for MySQL replication and recovery to work.

- `READ UNCOMMITTED`

  `SELECT` statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a “dirty read.” Otherwise, this isolation level works like `READ COMMITTED`.

- `SERIALIZABLE`

  This level is like `REPEATABLE READ`, but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT ... LOCK IN SHARE MODE` if `autocommit` is disabled. If `autocommit` is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions. (To force a plain `SELECT` to block if other transactions have modified the selected rows, disable `autocommit`.)

13.3.7 XA Transactions

As of MySQL 5.0.3, support for XA transactions is available for the `InnoDB` storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at http://www.opengroup.org/public/pubs/catalog/c193.htm. Limitations of the current XA implementation are described in Section C.5, “Restrictions on XA Transactions”.

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the `XA` keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also will work.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Among the MySQL Connectors, MySQL Connector/J 5.0.0 supports XA directly (by means of a class interface that handles the XA SQL statement interface for you).

XA supports distributed transactions, that is, the ability to permit multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (However, for a distributed transaction, you must use the `SERIALIZABLE` isolation level to achieve ACID properties. It is enough to use `REPEATABLE READ` for a nondistributed transaction, but not for a distributed transaction.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”
- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
- A bank keeps account information in an RDBMS and distributes and receives money through automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA MySQL enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable

This documentation is for an older version. If you're
This documentation is for an older version. If you're

storage. The branches indicate whether they are able to do this, and these results are used for the second phase.

2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they will be able to commit, all branches are told to commit. If any branch indicated when it was prepared that it will not be able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

13.3.7.1 XA Transaction SQL Syntax

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]

XA END xid [SUSPEND [FOR MIGRATE]]

XA PREPARE xid

XA COMMIT xid [ONE PHASE]

XA ROLLBACK xid

XA RECOVER
```

For `XA START`, the `JOIN` and `RESUME` clauses are not supported.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is not supported.

Each XA statement begins with the `XA` keyword, and most of them require an *`xid`* value. An *`xid`* is an XA transaction identifier. It indicates which transaction the statement applies to. *`xid`* values are supplied by the client, or generated by the MySQL server. An *`xid`* value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

`gtrid` is a global transaction identifier, *`bqual`* is a branch qualifier, and *`formatID`* is a number that identifies the format used by the *`gtrid`* and *`bqual`* values. As indicated by the syntax, *`bqual`* and *`formatID`* are optional. The default *`bqual`* value is `''` if not given. The default *`formatID`* value is 1 if not given.

`gtrid` and *`bqual`* must be string literals, each up to 64 bytes (not characters) long. *`gtrid`* and *`bqual`* can be specified in several ways. You can use a quoted string (`'ab'`), hex string (`X'6162'`, `0x6162`), or bit value (`b'nnnn'`).

`formatID` is an unsigned integer.

The *`gtrid`* and *`bqual`* values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write *`gtrid`* and *`bqual`* as hex strings.

`xid` values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own *`xid`* values in a list of values returned by the `XA RECOVER` statement.

For `XA START` *`xid`* starts an XA transaction with the given *`xid`* value. Each XA transaction must have a unique *`xid`* value, so the value must not currently be used by another XA transaction. Uniqueness is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the XA START statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA specification.)

The XA RECOVER statement returns information for those XA transactions on the MySQL server that are in the PREPARED state. (See Section 13.3.7.2, "XA Transaction States".) The output includes a row for each such XA transaction on the server, regardless of which client started it.

XA RECOVER output rows look like this (for an example *xid* value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        7 |            3 |            3 | abcdef |
+----------+--------------+--------------+--------+
```

The output columns have the following meanings:

- formatID is the *formatID* part of the transaction *xid*
- gtrid_length is the length in bytes of the *gtrid* part of the *xid*
- bqual_length is the length in bytes of the *bqual* part of the *xid*
- data is the concatenation of the *gtrid* and *bqual* parts of the *xid*

13.3.7.2 XA Transaction States

An XA transaction progresses through the following states:

1. Use XA START to start an XA transaction and put it in the ACTIVE state.
2. For an ACTIVE XA transaction, issue the SQL statements that make up the transaction, and then issue an XA END statement. XA END puts the transaction in the IDLE state.
3. For an IDLE XA transaction, you can issue either an XA PREPARE statement or an XA COMMIT ... ONE PHASE statement:
   - XA PREPARE puts the transaction in the PREPARED state. An XA RECOVER statement at this point will include the transaction's *xid* value in its output, because XA RECOVER lists all XA transactions that are in the PREPARED state.
   - XA COMMIT ... ONE PHASE prepares and commits the transaction. The *xid* value will not be listed by XA RECOVER because the transaction terminates.
4. For a PREPARED XA transaction, you can issue an XA COMMIT statement to commit and terminate the transaction, or XA ROLLBACK to roll back and terminate the transaction.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if `XA START` has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with `START TRANSACTION`, no XA statements can be used until the transaction has been committed or rolled back.

If an XA transaction is in the `ACTIVE` state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You will receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

Statements to which the preceding remark applies are listed at Section 13.3.3, “Statements That Cause an Implicit Commit”.

13.4 Replication Statements

Replication can be controlled through the SQL interface using the statements described in this section. One group of statements controls master servers, the other controls slave servers.

13.4.1 SQL Statements for Controlling Master Servers

This section discusses statements for managing master replication servers. Section 13.4.2, “SQL Statements for Controlling Slave Servers”, discusses statements for managing slave servers.

In addition to the statements described here, the following `SHOW` statements are used with master servers in replication. For information about these statements, see Section 13.7.5, “SHOW Syntax”.

- `SHOW BINARY LOGS`
- `SHOW BINLOG EVENTS`
- `SHOW MASTER STATUS`
- `SHOW SLAVE HOSTS`

13.4.1.1 PURGE BINARY LOGS Syntax

```
PURGE { BINARY | MASTER } LOGS
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    { TO 'log_name' | BEFORE datetime_expr }
```

The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file (see Section 5.4.3, “The Binary Log”).

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

This statement has no effect if the server was not started with the `--log-bin` option to enable binary logging.

Examples:

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2008-04-02 22:46:26';
```

The `BEFORE` variant's *`datetime_expr`* argument should evaluate to a `DATETIME` value (a value in `'YYYY-MM-DD hh:mm:ss'` format).

This statement is safe to run while slaves are replicating. You need not stop them. If you have an active slave that currently is reading one of the log files you are trying to delete, this statement does nothing and fails with an error. However, if a slave is not connected and you happen to purge one of the log files it has yet to read, the slave will be unable to replicate after it reconnects.

To safely purge binary log files, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log file it is reading.
2. Obtain a listing of the binary log files on the master server with `SHOW BINARY LOGS`.
3. Determine the earliest log file among all the slaves. This is the target file. If all the slaves are up to date, this is the last log file on the list.
4. Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)
5. Purge all log files up to but not including the target file.

You can also set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days (see Section 5.1.4, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master.

Prior to MySQL 5.0.60, `PURGE BINARY LOGS TO` and `PURGE BINARY LOGS BEFORE` did not behave in the same way (and neither one behaved correctly) when binary log files listed in the `.index` file had been removed from the system by some other means (such as using `rm` on Linux). Beginning with MySQL 5.0.60, both variants of the statement fail with an error in such cases. (Bug #18199, Bug #18453) To handle such errors, edit the `.index` file (which is a simple text file) manually to ensure that it lists only the binary log files that are actually present, then run again the `PURGE BINARY LOGS` statement that failed.

13.4.1.2 RESET MASTER Syntax

```
RESET MASTER
```

Deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. This statement is intended to be used only when the master is started for the first time.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Important

The effects of `RESET MASTER` differ from those of `PURGE BINARY LOGS` in 2 key ways:

1. `RESET MASTER` removes *all* binary log files that are listed in the index file, leaving only a single, empty binary log file with a numeric suffix of `.000001`, whereas the numbering is not reset by `PURGE BINARY LOGS`.

2. `RESET MASTER` is *not* intended to be used while any replication slaves are running. The behavior of `RESET MASTER` when used while slaves are running is undefined (and thus unsupported), whereas `PURGE BINARY LOGS` may be safely used while replication slaves are running.

See also Section 13.4.1.1, "PURGE BINARY LOGS Syntax".

`RESET MASTER` can prove useful when you first set up the master and the slave, so that you can verify the setup as follows:

1. Start the master and slave, and start replication (see Section 16.1.1, "How to Set Up Replication").

2. Execute a few test queries on the master.

3. Check that the queries were replicated to the slave.

4. When replication is running correctly, issue `STOP SLAVE` followed by `RESET SLAVE` on the slave, then verify that any unwanted data no longer exists on the slave.

5. Issue `RESET MASTER` on the master to clean up the test queries.

After verifying the setup and getting rid of any unwanted and log files generated by testing, you can start the slave and begin replicating.

13.4.1.3 SET sql_log_bin Syntax

```
SET sql_log_bin = {0|1}
```

Disables or enables binary logging for the current session (`sql_log_bin` is a session variable) if the client has the `SUPER` privilege. The statement fails with an error if the client does not have that privilege.

13.4.2 SQL Statements for Controlling Slave Servers

This section discusses statements for managing slave replication servers. Section 13.4.1, "SQL Statements for Controlling Master Servers", discusses statements for managing master servers.

In addition to the statements described here, `SHOW SLAVE STATUS` is also used with replication slaves. For information about this statement, see Section 13.7.5.31, "SHOW SLAVE STATUS Syntax".

13.4.2.1 CHANGE MASTER TO Syntax

```
CHANGE MASTER TO option [, option] ...

option:
    MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = interval
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
```

`CHANGE MASTER TO` changes the parameters that the slave server uses for connecting to the master server, for reading the master binary log, and reading the slave relay log. It also updates the contents of the `master.info` and `relay-log.info` files. To use `CHANGE MASTER TO`, the slave replication threads must be stopped (use `STOP SLAVE` if necessary).

Options not specified retain their value, except as indicated in the following discussion. Thus, in most cases, there is no need to specify options that do not change. For example, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

`MASTER_HOST`, `MASTER_USER`, `MASTER_PASSWORD`, and `MASTER_PORT` provide information to the slave about how to connect to its master:

- `MASTER_HOST` and `MASTER_PORT` are the host name (or IP address) of the master host and its TCP/IP port.

  Note

  Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

  If you specify the `MASTER_HOST` or `MASTER_PORT` option, the slave assumes that the master server is different from before (even if the option value is the same as its current value.) In this case, the old values for the master binary log file name and position are considered no longer applicable, so if you do not specify `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the statement, `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4` are silently appended to it.

  Setting `MASTER_HOST=''`—that is, setting its value explicitly to an empty string—is *not* the same as not setting it at all. Setting this option to an empty string causes `START SLAVE` subsequently to fail. (Bug #28796)

- `MASTER_USER` and `MASTER_PASSWORD` are the user name and password of the account to use for connecting to the master.

  The password used for a MySQL Replication slave account in a `CHANGE MASTER TO` statement is limited to 32 characters in length; if the password is longer, the statement succeeds, but any excess characters are silently truncated. This is an issue specific to MySQL Replication, which is fixed in MySQL 5.7. (Bug #11752299, Bug #43439)

  The text of a running `CHANGE MASTER TO` statement, including values for `MASTER_USER` and `MASTER_PASSWORD`, can be seen in the output of a concurrent `SHOW PROCESSLIST` statement.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `MASTER_SSL_xxx` options provide information about using SSL for the connection. They correspond to the `--ssl-xxx` options described in Section 6.3.6.5, “Command Options for Secure Connections”, and Section 16.3.7, “Setting Up Replication to Use Secure Connections”. These options can be changed even on slaves that are compiled without SSL support. They are saved to the `master.info` file, but are ignored if the slave does not have SSL support enabled.

`MASTER_CONNECT_RETRY` specifies how many seconds to wait between connect retries. The default is 60. The *number* of reconnection attempts is limited by the `--master-retry-count` server option; for more information, see Section 16.1.2, “Replication and Binary Logging Options and Variables”.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. `RELAY_LOG_FILE` and `RELAY_LOG_POS` are the coordinates at which the slave SQL thread should begin reading from the relay log the next time the thread starts. If you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` is specified, the slave uses the last coordinates of the *slave SQL thread* before `CHANGE MASTER TO` was issued. This ensures that there is no discontinuity in replication, even if the slave SQL thread was late compared to the slave I/O thread, when you merely want to change, say, the password to use.

`CHANGE MASTER TO` *deletes all relay log files* and starts a new one, unless you specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. In that case, relay log files are kept; the `relay_log_purge` global variable is set silently to 0.

`CHANGE MASTER TO` is useful for setting up a slave when you have the snapshot of the master and have recorded the master binary log coordinates corresponding to the time of the snapshot. After loading the snapshot into the slave to synchronize it with the master, you can run `CHANGE MASTER TO MASTER_LOG_FILE='log_name', MASTER_LOG_POS=log_pos` on the slave to specify the coordinates at which the slave should begin reading the master binary log.

The following example changes the master server the slave uses and establishes the master binary log coordinates from which the slave begins reading. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay log files that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use `CHANGE MASTER TO` and start the SQL thread (`START SLAVE SQL_THREAD`):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

You can even use the second operation in a nonreplication setup with a standalone, nonslave server for recovery following a crash. Suppose that your server has crashed and you have restored it from a backup. You want to replay the server's own binary log files (not relay log files, but regular binary log files), named (for example) `myhost-bin.*`. First, make a backup copy of these binary log files in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary log. Use `SET GLOBAL relay_log_purge=0` for additional safety. Then start the server without the `--`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`log-bin` option, Instead, use the `--replicate-same-server-id`, `--relay-log=myhost-bin` (to make the server believe that these regular binary log files are relay log files) and `--skip-slave-start` options. After the server starts, issue these statements:

```
CHANGE MASTER TO
  RELAY_LOG_FILE='myhost-bin.153',
  RELAY_LOG_POS=410,
  MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

The server reads and executes its own binary log files, thus achieving crash recovery. Once the recovery is finished, run `STOP SLAVE`, shut down the server, delete the `master.info` and `relay-log.info` files, and restart the server with its original options.

Specifying the `MASTER_HOST` option (even with a dummy value) is required to make the server think it is a slave.

13.4.2.2 LOAD DATA FROM MASTER Syntax

```
LOAD DATA FROM MASTER
```

Note

This feature is deprecated and should be avoided. It is subject to removal in a future version of MySQL.

Since the current implementation of `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` is very limited, these statements are deprecated as of MySQL 4.1 and removed in MySQL 5.5.

The recommended alternative solution to using `LOAD DATA FROM MASTER` or `LOAD TABLE FROM MASTER` is using `mysqldump` or `mysqlhotcopy`. The latter requires Perl and two Perl modules (`DBI` and `DBD:mysql`) and works for `MyISAM` and `ARCHIVE` tables only. With `mysqldump`, you can create SQL dumps on the master and pipe (or copy) these to a `mysql` client on the slave. This has the advantage of working for all storage engines, but can be quite slow, since it works using `SELECT`.

This statement takes a snapshot of the master and copies it to the slave. It updates the values of `MASTER_LOG_FILE` and `MASTER_LOG_POS` so that the slave starts replicating from the correct position. Any table and database exclusion rules specified with the `--replicate-*-do-*` and `--replicate-*-ignore-*` options are honored. `--replicate-rewrite-db` is *not* taken into account because a user could use this option to set up a nonunique mapping such as `--replicate-rewrite-db="db1->db3"` and `--replicate-rewrite-db="db2->db3"`, which would confuse the slave when loading tables from the master.

Use of this statement is subject to the following conditions:

- It works only for `MyISAM` tables. Attempting to load a non-`MyISAM` table results in the following error:

  ```
  ERROR 1189 (08S01): Net error reading from master
  ```

- It acquires a global read lock on the master while taking the snapshot, which prevents updates on the master during the load operation.

If you are loading large tables, you might have to increase the values of `net_read_timeout` and `net_write_timeout` on both the master and slave servers. See Section 5.1.4, “Server System Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`LOAD DATA FROM MASTER` does *not* copy any tables from the `mysql` database. This makes it easy to have different users and privileges on the master and the slave.

To use `LOAD DATA FROM MASTER`, the replication account that is used to connect to the master must have the `RELOAD` and `SUPER` privileges on the master and the `SELECT` privilege for all master tables you want to load. All master tables for which the user does not have the `SELECT` privilege are ignored by `LOAD DATA FROM MASTER`. This is because the master hides them from the user: `LOAD DATA FROM MASTER` calls `SHOW DATABASES` to know the master databases to load, but `SHOW DATABASES` returns only databases for which the user has some privilege. See Section 13.7.5.11, "SHOW DATABASES Syntax". On the slave side, the user that issues `LOAD DATA FROM MASTER` must have privileges for dropping and creating the databases and tables that are copied.

13.4.2.3 LOAD TABLE tbl_name FROM MASTER Syntax

```
LOAD TABLE tbl_name FROM MASTER
```

Note

This feature is deprecated and should be avoided. It is subject to removal in a future version of MySQL.

Since the current implementation of `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` is very limited, these statements are deprecated as of MySQL 4.1 and removed in MySQL 5.5.

The recommended alternative solution to using `LOAD DATA FROM MASTER` or `LOAD TABLE FROM MASTER` is using `mysqldump` or `mysqlhotcopy`. The latter requires Perl and two Perl modules (`DBI` and `DBD:mysql`) and works for `MyISAM` and `ARCHIVE` tables only. With `mysqldump`, you can create SQL dumps on the master and pipe (or copy) these to a `mysql` client on the slave. This has the advantage of working for all storage engines, but can be quite slow, since it works using `SELECT`.

Transfers a copy of the table from the master to the slave. This statement is implemented mainly debugging `LOAD DATA FROM MASTER` operations. To use `LOAD TABLE`, the account used for connecting to the master server must have the `RELOAD` and `SUPER` privileges on the master and the `SELECT` privilege for the master table to load. On the slave side, the user that issues `LOAD TABLE FROM MASTER` must have privileges for dropping and creating the table.

The conditions for `LOAD DATA FROM MASTER` apply here as well. For example, `LOAD TABLE FROM MASTER` works only for `MyISAM` tables. The timeout notes for `LOAD DATA FROM MASTER` apply as well.

13.4.2.4 MASTER_POS_WAIT() Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos [, timeout])
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See Section 12.15, "Miscellaneous Functions", for a full description.

The following table shows the maximum permissible length for the string-valued options.

Option	Maximum Length
`MASTER_HOST`	60
`MASTER_USER`	16

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option	Maximum Length
MASTER_PASSWORD	32
MASTER_LOG_FILE	255
RELAY_LOG_FILE	255
MASTER_SSL_CA	255
MASTER_SSL_CAPATH	255
MASTER_SSL_CERT	255
MASTER_SSL_KEY	255
MASTER_SSL_CIPHER	511

13.4.2.5 RESET SLAVE Syntax

```
RESET SLAVE
```

`RESET SLAVE` makes the slave forget its replication position in the master's binary log. This statement is meant to be used for a clean start: It deletes the `master.info` and `relay-log.info` files, all the relay log files, and starts a new relay log file. To use `RESET SLAVE`, the slave replication threads must be stopped (use `STOP SLAVE` if necessary).

Note

All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a `STOP SLAVE` statement or if the slave is highly loaded.)

Connection information stored in the `master.info` file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. Options for which values are not specified are cleared. If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the slave.

13.4.2.6 SET GLOBAL sql_slave_skip_counter Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

This statement skips the next *N* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave threads are not running. Otherwise, it produces an error.

When using this statement, it is important to understand that the binary log is actually organized as a sequence of groups known as *event groups*. Each event group consists of a sequence of events.

- For transactional tables, an event group corresponds to a transaction.
- For nontransactional tables, an event group corresponds to a single SQL statement.

Note

A single transaction can contain changes to both transactional and nontransactional tables.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When you use `SET GLOBAL sql_slave_skip_counter` to skip events and the result is in the middle of a group, the slave continues to skip events until it reaches the end of the group. Execution then starts with the next event group.

13.4.2.7 START SLAVE Syntax

```
START SLAVE [thread_types]

START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos

START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos

thread_types:
    [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

`START SLAVE` with no *`thread_type`* options starts both of the slave threads. The I/O thread reads events from the master server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. `START SLAVE` requires the `SUPER` privilege.

If `START SLAVE` succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage to connect to the master or read its binary log, or some other problem). `START SLAVE` does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with `SHOW SLAVE STATUS`.

`START SLAVE` sends an acknowledgment to the user after both the I/O thread and the SQL thread have started. However, the I/O thread may not yet have connected. For this reason, a successful `START SLAVE` causes `SHOW SLAVE STATUS` to show `Slave_SQL_Running=Yes`, but this does not guarantee that `Slave_IO_Running=Yes` (because `Slave_IO_Running=Yes` only if the I/O thread is running *and connected*). For more information, see Section 13.7.5.31, “SHOW SLAVE STATUS Syntax”, and Section 16.1.3.1, “Checking Replication Status”.

You can add `IO_THREAD` and `SQL_THREAD` options to the statement to name which of the threads to start.

An `UNTIL` clause may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary log or in the slave relay log. When the SQL thread reaches that point, it stops. If the `SQL_THREAD` option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the `UNTIL` clause is ignored and a warning is issued.

For an `UNTIL` clause, you must specify both a log file name and position. Do not mix master and relay log options.

Any `UNTIL` condition is reset by a subsequent `STOP SLAVE` statement, a `START SLAVE` statement that includes no `UNTIL` clause, or a server restart.

The `UNTIL` clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate an event. For example, if an unwise `DROP TABLE` statement was executed on the master, you can use `UNTIL` to tell the slave to execute up to that point but no farther. To find what the event is, use `mysqlbinlog` with the master binary log or slave relay log, or by using a `SHOW BINLOG EVENTS` statement.

If you are using `UNTIL` to have the slave process replicated queries in sections, it is recommended that you start the slave with the `--skip-slave-start` option to prevent the SQL thread from running when

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The `SHOW SLAVE STATUS` statement includes output fields that display the current values of the `UNTIL` condition.

In old versions of MySQL (before 4.0.5), this statement was called `SLAVE START`. This usage is still accepted in MySQL 5.0 for backward compatibility, but is deprecated and is removed in MySQL 5.6.

13.4.2.8 STOP SLAVE Syntax

```
STOP SLAVE [thread_types]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Stops the slave threads. `STOP SLAVE` requires the `SUPER` privilege. Recommended best practice is to execute `STOP SLAVE` on the slave before stopping the slave server (see Section 5.1.10, “The Server Shutdown Process”, for more information).

Like `START SLAVE`, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped.

Note

The transactional behavior of `STOP SLAVE` changed in MySQL 5.0.82. Previously, it took effect immediately; beginning with MySQL 5.0.82, it waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. (Bug #319, Bug #38205)

In old versions of MySQL (before 4.0.5), this statement was called `SLAVE STOP`. This usage is still accepted in MySQL 5.0 for backward compatibility, but is deprecated and is removed in MySQL 5.6.

13.5 SQL Syntax for Prepared Statements

MySQL 5.0 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol, provided that you use an appropriate client programming interface. Candidate interfaces include the MySQL C API client library (for C programs), MySQL Connector/J (for Java programs), and MySQL Connector/Net. For example, the C API provides a set of function calls that make up its prepared statement API. See Section 20.6.8, “C API Prepared Statements”. Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the `mysqli` extension, available in PHP 5.0 and later.

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that enables you to send SQL statements to the server to be executed, such as the `mysql` client program.
- You can use it even if the client is using an old version of the client library. The only requirement is that you be able to connect to a server that is recent enough to support SQL syntax for prepared statements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SQL syntax for prepared statements is intended to be used for situations such as these:

- You want to test how prepared statements work in your application before coding it.
- An application has problems executing prepared statements and you want to determine interactively what the problem is.
- You want to create a test case that describes a problem you are having with prepared statements, so that you can file a bug report.
- You need to use prepared statements but do not have access to a programming API that supports them.

SQL syntax for prepared statements is based on three SQL statements:

- `PREPARE` prepares a statement for execution (see Section 13.5.1, “PREPARE Syntax”).
- `EXECUTE` executes a prepared statement (see Section 13.5.2, “EXECUTE Syntax”).
- `DEALLOCATE PREPARE` releases a prepared statement (see Section 13.5.3, “DEALLOCATE PREPARE Syntax”).

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|          5 |
+------------+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

Here is an additional example which demonstrates how to choose the table on which to perform a query at runtime, by storing the name of the table as a user variable:

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+----+
| a  |
+----+
|  4 |
|  8 |
| 11 |
| 32 |
| 80 |
+----+

mysql> DEALLOCATE PREPARE stmt3;
```

A prepared statement is specific to the session in which it was created. If you terminate a session without deallocating a previously prepared statement, the server deallocates it automatically.

A prepared statement is also global to the session. If you create a prepared statement within a stored routine, it is not deallocated when the stored routine ends.

To guard against too many prepared statements being created simultaneously, set the `max_prepared_stmt_count` system variable. To prevent the use of prepared statements, set the value to 0.

The following SQL statements can be used in prepared statements:

```
ALTER TABLE
CALL
COMMIT
{CREATE | DROP} INDEX
{CREATE | DROP} TABLE
DELETE
DO
INSERT
RENAME TABLE
REPLACE
SELECT
SET
SHOW (most variants)
TRUNCATE TABLE
UPDATE
```

As of MySQL 5.0.15, the following additional statements are supported:

```
{CREATE | DROP} VIEW
```

As of MySQL 5.0.23, the following additional statements are supported:

```
ANALYZE TABLE
OPTIMIZE TABLE
REPAIR TABLE
```

Other statements are not supported in MySQL 5.0.

Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. Exceptions are noted in Section C.1, “Restrictions on Stored Programs”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As of MySQL 5.0.7, placeholders can be used for the arguments of the `LIMIT` clause when using prepared statements. See Section 13.2.8, “SELECT Syntax”.

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholder support for `OUT` and `INOUT` parameters is not available in MySQL 5.0. See Section 13.2.1, “CALL Syntax”, for an example and a workaround. Placeholders can be used for `IN` parameters regardless of version.

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to `PREPARE` cannot itself be a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements cannot be used within stored routines (procedures or functions), or triggers. This restriction is lifted as of MySQL 5.0.13 for stored procedures, but not for stored functions or triggers. However, a cursor cannot be used for a dynamic statement that is prepared and executed with `PREPARE` and `EXECUTE`. The statement for a cursor is checked at cursor creation time, so the statement cannot be dynamic.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by “`;`” characters).

To write C programs that use the `CALL` SQL statement to execute stored procedures that contain prepared statements, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). For additional information, see Section 13.2.1, “CALL Syntax”.

13.5.1 PREPARE Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

The `PREPARE` statement prepares a SQL statement and assigns it a name, *`stmt_name`*, by which to refer to the statement later. The prepared statement is executed with `EXECUTE` and released with `DEALLOCATE PREPARE`. For examples, see Section 13.5, “SQL Syntax for Prepared Statements”.

Statement names are not case sensitive. *`preparable_stmt`* is either a string literal or a user variable that contains the text of the SQL statement. The text must represent a single statement, not multiple statements. Within the statement, `?` characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The `?` characters should not be enclosed within quotation marks, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the session within which it is created, which as several implications:

- A prepared statement created in one session is not available to other sessions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- When a session ends, whether normally or abnormally, its prepared statements no longer exist. If auto-reconnect is enabled, the client is not notified that the connection was lost. For this reason, clients may wish to disable auto-reconnect. See Section 20.6.15, “Controlling Automatic Reconnection Behavior”.
- A prepared statement created within a stored program continues to exist after the program finishes executing and can be executed outside the program later.
- A statement prepared in stored program context cannot refer to stored procedure or function parameters or local variables because they go out of scope when the program ends and would be unavailable were the statement to be executed later outside the program. As a workaround, refer instead to user-defined variables, which also have session scope; see Section 9.4, “User-Defined Variables”.

13.5.2 EXECUTE Syntax

```
EXECUTE stmt_name
    [USING @var_name [, @var_name] ...]
```

After preparing a statement with `PREPARE`, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

For examples, see Section 13.5, “SQL Syntax for Prepared Statements”.

13.5.3 DEALLOCATE PREPARE Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

To deallocate a prepared statement produced with `PREPARE`, use a `DEALLOCATE PREPARE` statement that refers to the prepared statement name. Attempting to execute a prepared statement after deallocating it results in an error.

For examples, see Section 13.5, “SQL Syntax for Prepared Statements”.

13.6 MySQL Compound-Statement Syntax

This section describes the syntax for the `BEGIN ... END` compound statement and other statements that can be used in the body of stored programs: Stored procedures and functions and triggers. These objects are defined in terms of SQL code that is stored on the server for later invocation (see Chapter 18, *Stored Programs and Views*).

A compound statement is a block that can contain other blocks; declarations for variables, condition handlers, and cursors; and flow control constructs such as loops and conditional tests.

13.6.1 BEGIN ... END Compound-Statement Syntax

```
[begin_label:] BEGIN
    [statement_list]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
END [end_label]
```

`BEGIN ... END` syntax is used for writing compound statements, which can appear within stored programs (stored procedures and functions, and triggers). A compound statement can contain multiple statements, enclosed by the `BEGIN` and `END` keywords. *`statement_list`* represents a list of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The *`statement_list`* itself is optional, so the empty compound statement (`BEGIN END`) is legal.

`BEGIN ... END` blocks can be nested.

Use of multiple statements requires that a client is able to send statement strings containing the `;` statement delimiter. In the `mysql` command-line client, this is handled with the `delimiter` command. Changing the `;` end-of-statement delimiter (for example, to `//`) permit `;` to be used in a program body. For an example, see Section 18.1, “Defining Stored Programs”.

A `BEGIN ... END` block can be labeled. See Section 13.6.2, “Statement Label Syntax”.

The optional `[NOT] ATOMIC` clause is not supported. This means that no transactional savepoint is set at the start of the instruction block and the `BEGIN` clause used in this context has no effect on the current transaction.

Note

Within all stored programs, the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

13.6.2 Statement Label Syntax

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]

[begin_label:] LOOP
    statement_list
END LOOP [end_label]

[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Labels are permitted for `BEGIN ... END` blocks and for the `LOOP`, `REPEAT`, and `WHILE` statements. Label use for those statements follows these rules:

- *`begin_label`* must be followed by a colon.
- *`begin_label`* can be given without *`end_label`*. If *`end_label`* is present, it must be the same as *`begin_label`*.
- *`end_label`* cannot be given without *`begin_label`*.
- Labels at the same nesting level must be distinct.
- Labels can be up to 16 characters long.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To refer to a label within the labeled construct, use an `ITERATE` or `LEAVE` statement. The following example uses those statements to continue iterating or terminate the loop:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
END;
```

The scope of a block label does not include the code for handlers declared within the block. For details, see Section 13.6.7.2, “DECLARE ... HANDLER Syntax”.

13.6.3 DECLARE Syntax

The `DECLARE` statement is used to define various items local to a program:

- Local variables. See Section 13.6.4, “Variables in Stored Programs”.
- Conditions and handlers. See Section 13.6.7, “Condition Handling”.
- Cursors. See Section 13.6.6, “Cursors”.

`DECLARE` is permitted only inside a `BEGIN ... END` compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursor declarations must appear before handler declarations. Variable and condition declarations must appear before cursor or handler declarations.

13.6.4 Variables in Stored Programs

System variables and user-defined variables can be used in stored programs, just as they can be used outside stored-program context. In addition, stored programs can use `DECLARE` to define local variables, and stored routines (procedures and functions) can be declared to take parameters that communicate values between the routine and its caller.

- To declare local variables, use the `DECLARE` statement, as described in Section 13.6.4.1, “Local Variable DECLARE Syntax”.
- Variables can be set directly with the `SET` statement. See Section 13.7.4, “SET Syntax”.
- Results from queries can be retrieved into local variables using `SELECT ... INTO` *`var_list`* or by opening a cursor and using `FETCH ... INTO` *`var_list`*. See Section 13.2.8.1, “SELECT ... INTO Syntax”, and Section 13.6.6, “Cursors”.

For information about the scope of local variables and how MySQL resolves ambiguous names, see Section 13.6.4.2, “Local Variable Scope and Resolution”.

13.6.4.1 Local Variable DECLARE Syntax

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

This statement declares local variables within stored programs. To provide a default value for a variable, include a `DEFAULT` clause. The value can be specified as an expression; it need not be a constant. If the `DEFAULT` clause is missing, the initial value is `NULL`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.

Variable declarations must appear before cursor or handler declarations.

Local variable names are not case sensitive. Permissible characters and quoting rules are the same as for other identifiers, as described in Section 9.2, “Schema Object Names”.

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

13.6.4.2 Local Variable Scope and Resolution

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Syntax”.

A local variable should not have the same name as a table column. If an SQL statement, such as a `SELECT ... INTO` statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable. Consider the following procedure definition:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname, id INTO newname, xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

MySQL interprets `xname` in the `SELECT` statement as a reference to the `xname` *variable* rather than the `xname` *column*. Consequently, when the procedure `sp1()`is called, the `newname` variable returns the value `'bob'` regardless of the value of the `table1.xname` column.

Similarly, the cursor definition in the following procedure contains a `SELECT` statement that refers to `xname`. MySQL interprets this as a reference to the variable of that name rather than a column reference.

```
CREATE PROCEDURE sp2 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;
  DECLARE done TINYINT DEFAULT 0;
  DECLARE cur1 CURSOR FOR SELECT xname, id FROM table1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  OPEN cur1;
  read_loop: LOOP
    FETCH FROM cur1 INTO newname, xid;
    IF done THEN LEAVE read_loop; END IF;
    SELECT newname;
  END LOOP;
  CLOSE cur1;
END;
```

See also Section C.1, “Restrictions on Stored Programs”.

13.6.5 Flow Control Statements

MySQL supports the `IF`, `CASE`, `ITERATE`, `LEAVE LOOP`, `WHILE`, and `REPEAT` constructs for flow control within stored programs. It also supports `RETURN` within stored functions.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an `IF` statement might contain a `WHILE` loop, which itself contains a `CASE` statement.

MySQL does not support `FOR` loops.

13.6.5.1 CASE Syntax

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

The `CASE` statement for stored programs implements a complex conditional construct.

Note

There is also a `CASE` *expression*, which differs from the `CASE` *statement* described here. See Section 12.4, “Control Flow Functions”. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

For the first syntax, *`case_value`* is an expression. This value is compared to the *`when_value`* expression in each `WHEN` clause until one of them is equal. When an equal *`when_value`* is found, the corresponding `THEN` clause *`statement_list`* executes. If no *`when_value`* is equal, the `ELSE` clause *`statement_list`* executes, if there is one.

This syntax cannot be used to test for equality with `NULL` because `NULL = NULL` is false. See Section 3.3.4.6, “Working with NULL Values”.

For the second syntax, each `WHEN` clause *`search_condition`* expression is evaluated until one is true, at which point its corresponding `THEN` clause *`statement_list`* executes. If no *`search_condition`* is equal, the `ELSE` clause *`statement_list`* executes, if there is one.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If no *`when_value`* or *`search_condition`* matches the value tested and the `CASE` statement contains no `ELSE` clause, a `Case not found for CASE statement` error results.

Each *`statement_list`* consists of one or more SQL statements; an empty *`statement_list`* is not permitted.

To handle situations where no value is matched by any `WHEN` clause, use an `ELSE` containing an empty `BEGIN ... END` block, as shown in this example. (The indentation used here in the `ELSE` clause is for purposes of clarity only, and is not otherwise significant.)

```
DELIMITER |

CREATE PROCEDURE p()
  BEGIN
    DECLARE v INT DEFAULT 1;

    CASE v
      WHEN 2 THEN SELECT v;
      WHEN 3 THEN SELECT 0;
      ELSE
        BEGIN
        END;
    END CASE;
  END;
  |
```

13.6.5.2 IF Syntax

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

The `IF` statement for stored programs implements a basic conditional construct.

Note

There is also an `IF()` *function*, which differs from the `IF` *statement* described here. See Section 12.4, “Control Flow Functions”. The `IF` statement can have `THEN`, `ELSE`, and `ELSEIF` clauses, and it is terminated with `END IF`.

If the *`search_condition`* evaluates to true, the corresponding `THEN` or `ELSEIF` clause *`statement_list`* executes. If no *`search_condition`* matches, the `ELSE` clause *`statement_list`* executes.

Each *`statement_list`* consists of one or more SQL statements; an empty *`statement_list`* is not permitted.

An `IF ... END IF` block, like all other flow-control blocks used within stored programs, must be terminated with a semicolon, as shown in this example:

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END //

DELIMITER ;
```

As with other flow-control constructs, `IF ... END IF` blocks may be nested within other flow-control constructs, including other `IF` statements. Each `IF` must be terminated by its own `END IF` followed by a semicolon. You can use indentation to make nested flow-control blocks more easily readable by humans (although this is not required by MySQL), as shown here:

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
  RETURNS VARCHAR(50)

  BEGIN
    DECLARE s VARCHAR(50);

    IF n = m THEN SET s = 'equals';
    ELSE
      IF n > m THEN SET s = 'greater';
      ELSE SET s = 'less';
      END IF;

      SET s = CONCAT('is ', s, ' than');
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m, '.');

    RETURN s;
  END //

DELIMITER ;
```

In this example, the inner `IF` is evaluated only if `n` is not equal to `m`.

13.6.5.3 ITERATE Syntax

```
ITERATE label
```

`ITERATE` can appear only within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means “start the loop again.”

For an example, see Section 13.6.5.5, “LOOP Syntax”.

13.6.5.4 LEAVE Syntax

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given label. If the label is for the outermost stored program block, `LEAVE` exits the program.

`LEAVE` can be used within `BEGIN ... END` or loop constructs (`LOOP`, `REPEAT`, `WHILE`).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For an example, see Section 13.6.5.5, “LOOP Syntax”.

13.6.5.5 LOOP Syntax

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

LOOP implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (;) statement delimiter. The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a LEAVE statement. Within a stored function, RETURN can also be used, which exits the function entirely.

Neglecting to include a loop-termination statement results in an infinite loop.

A LOOP statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

13.6.5.6 REPEAT Syntax

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the *search_condition* expression is true. Thus, a REPEAT always enters the loop at least once. *statement_list* consists of one or more statements, each terminated by a semicolon (;) statement delimiter.

A REPEAT statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT
    ->     SET @x = @x + 1;
    ->   UNTIL @x > p1 END REPEAT;
    -> END
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

13.6.5.7 RETURN Syntax

```
RETURN expr
```

The `RETURN` statement terminates execution of a stored function and returns the value *`expr`* to the function caller. There must be at least one `RETURN` statement in a stored function. There may be more than one if the function has multiple exit points.

This statement is not used in stored procedures or triggers. The `LEAVE` statement can be used to exit a stored program of those types.

13.6.5.8 WHILE Syntax

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement list within a `WHILE` statement is repeated as long as the *`search_condition`* expression is true. *`statement_list`* consists of one or more SQL statements, each terminated by a semicolon (`;`) statement delimiter.

A `WHILE` statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

13.6.6 Cursors

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Nonscrollable: Can be traversed only in one direction and cannot skip rows

Cursor declarations must appear before handler declarations and after variable and condition declarations.

Example:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END;
```

13.6.6.1 Cursor CLOSE Syntax

```
CLOSE cursor_name
```

This statement closes a previously opened cursor. For an example, see Section 13.6.6, “Cursors”.

An error occurs if the cursor is not open.

If not closed explicitly, a cursor is closed at the end of the `BEGIN ... END` block in which it was declared.

13.6.6.2 Cursor DECLARE Syntax

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor and associates it with a `SELECT` statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a `FETCH` statement. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

The `SELECT` statement cannot have an `INTO` clause.

Cursor declarations must appear before handler declarations and after variable and condition declarations.

A stored program may contain multiple cursor declarations, but each cursor declared in a given block must have a unique name. For an example, see Section 13.6.6, “Cursors”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For information available through SHOW statements, it is possible in many cases to obtain equivalent information by using a cursor with an INFORMATION_SCHEMA table.

13.6.6.3 Cursor FETCH Syntax

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row for the SELECT statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the SELECT statement must match the number of output variables specified in the FETCH statement.

If no more rows are available, a No Data condition occurs with SQLSTATE value '02000'. To detect this condition, you can set up a handler for it (or for a NOT FOUND condition). For an example, see Section 13.6.6, “Cursors”.

13.6.6.4 Cursor OPEN Syntax

```
OPEN cursor_name
```

This statement opens a previously declared cursor. For an example, see Section 13.6.6, “Cursors”.

13.6.7 Condition Handling

Conditions may arise during stored program execution that require special handling, such as exiting the current program block or continuing execution. Handlers can be defined for general conditions such as warnings or exceptions, or for specific conditions such as a particular error code. Specific conditions can be assigned names and referred to that way in handlers.

To name a condition, use the DECLARE ... CONDITION statement. To declare a handler, use the DECLARE ... HANDLER statement. See Section 13.6.7.1, “DECLARE ... CONDITION Syntax”, and Section 13.6.7.2, “DECLARE ... HANDLER Syntax”.

Other statements related to conditions are SIGNAL, RESIGNAL, and GET DIAGNOSTICS. The SIGNAL and RESIGNAL statements are not supported until MySQL 5.5. The GET DIAGNOSTICS statement is not supported until MySQL 5.6.

13.6.7.1 DECLARE ... CONDITION Syntax

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
```

The DECLARE ... CONDITION statement declares a named error condition, associating a name with a condition that needs specific handling. The name can be referred to in a subsequent DECLARE ... HANDLER statement (see Section 13.6.7.2, “DECLARE ... HANDLER Syntax”).

Condition declarations must appear before cursor or handler declarations.

The *condition_value* for DECLARE ... CONDITION indicates the specific condition or class of conditions to associate with the condition name. It can take the following forms:

- *mysql_error_code*: An integer literal indicating a MySQL error code.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Do not use MySQL error code 0 because that indicates success rather than an error condition. For a list of MySQL error codes, see Section B.3, “Server Error Codes and Messages”.

- SQLSTATE [VALUE] *`sqlstate_value`*: A 5-character string literal indicating an SQLSTATE value.

  Do not use SQLSTATE values that begin with `'00'` because those indicate success rather than an error condition. For a list of SQLSTATE values, see Section B.3, “Server Error Codes and Messages”.

Using names for conditions can help make stored program code clearer. For example, this handler applies to attempts to drop a nonexistent table, but that is apparent only if you know that 1051 is the MySQL error code for “unknown table”:

```
DECLARE CONTINUE HANDLER FOR 1051
  BEGIN
    -- body of handler
  END;
```

By declaring a name for the condition, the purpose of the handler is more readily seen:

```
DECLARE no_such_table CONDITION FOR 1051;
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
    -- body of handler
  END;
```

Here is a named condition for the same condition, but based on the corresponding SQLSTATE value rather than the MySQL error code:

```
DECLARE no_such_table CONDITION FOR SQLSTATE '42S02';
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
    -- body of handler
  END;
```

13.6.7.2 DECLARE ... HANDLER Syntax

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action:
    CONTINUE
  | EXIT
  | UNDO

condition_value:
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

The `DECLARE ... HANDLER` statement specifies a handler that deals with one or more conditions. If one of these conditions occurs, the specified *`statement`* executes. *`statement`* can be a simple statement such as `SET var_name = value`, or a compound statement written using `BEGIN` and `END` (see Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Handler declarations must appear after variable or condition declarations.

The *`handler_action`* value indicates what action the handler takes after execution of the handler statement:

- `CONTINUE`: Execution of the current program continues.
- `EXIT`: Execution terminates for the `BEGIN ... END` compound statement in which the handler is declared. This is true even if the condition occurs in an inner block.
- `UNDO`: Not supported.

The *`condition_value`* for `DECLARE ... HANDLER` indicates the specific condition or class of conditions that activates the handler. It can take the following forms:

- *`mysql_error_code`*: An integer literal indicating a MySQL error code, such as 1051 to specify "unknown table":

  ```
  DECLARE CONTINUE HANDLER FOR 1051
    BEGIN
      -- body of handler
    END;
  ```

  Do not use MySQL error code 0 because that indicates success rather than an error condition. For a list of MySQL error codes, see Section B.3, "Server Error Codes and Messages".

- SQLSTATE [VALUE] *`sqlstate_value`*: A 5-character string literal indicating an SQLSTATE value, such as `'42S01'` to specify "unknown table":

  ```
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
    BEGIN
      -- body of handler
    END;
  ```

  Do not use SQLSTATE values that begin with `'00'` because those indicate success rather than an error condition. For a list of SQLSTATE values, see Section B.3, "Server Error Codes and Messages".

- *`condition_name`*: A condition name previously specified with `DECLARE ... CONDITION`. A condition name can be associated with a MySQL error code or SQLSTATE value. See Section 13.6.7.1, "DECLARE ... CONDITION Syntax".

- `SQLWARNING`: Shorthand for the class of SQLSTATE values that begin with `'01'`.

  ```
  DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      -- body of handler
    END;
  ```

- `NOT FOUND`: Shorthand for the class of SQLSTATE values that begin with `'02'`. This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it or for a `NOT FOUND` condition.

  ```
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
      -- body of handler
    END;
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For another example, see Section 13.6.6, “Cursors”. The `NOT FOUND` condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.

- `SQLEXCEPTION`: Shorthand for the class of SQLSTATE values that do not begin with `'00'`, `'01'`, or `'02'`.

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  BEGIN
    -- body of handler
  END;
```

If a condition occurs for which no handler has been declared, the action taken depends on the condition class:

- For `SQLEXCEPTION` conditions, the stored program terminates at the statement that raised the condition, as if there were an `EXIT` handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For `SQLWARNING` or `NOT FOUND` conditions, the program continues executing, as if there were a `CONTINUE` handler.

The following example uses a handler for `SQLSTATE '23000'`, which occurs for a duplicate-key error:

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Notice that `@x` is `3` after the procedure executes, which shows that execution continued to the end of the procedure after the error occurred. If the `DECLARE ... HANDLER` statement had not been present, MySQL would have taken the default action (`EXIT`) after the second `INSERT` failed due to the `PRIMARY KEY` constraint, and `SELECT @x` would have returned `2`.

To ignore a condition, declare a `CONTINUE` handler for it and associate it with an empty block. For example:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

The scope of a block label does not include the code for handlers declared within the block. Therefore, the statement associated with a handler cannot use `ITERATE` or `LEAVE` to refer to labels for blocks that enclose the handler declaration. Consider the following example, where the `REPEAT` block has a label of `retry`:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
    REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING
          BEGIN
            ITERATE retry;    # illegal
          END;
        IF i < 0 THEN
          LEAVE retry;        # legal
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

The `retry` label is in scope for the `IF` statement within the block. It is not in scope for the `CONTINUE` handler, so the reference there is invalid and results in an error:

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

To avoid references to outer labels in handlers, use one of these strategies:

- To leave the block, use an `EXIT` handler. If no block cleanup is required, the `BEGIN ... END` handler body can be empty:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
  ```

  Otherwise, put the cleanup statements in the handler body:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING
    BEGIN
      block cleanup statements
    END;
  ```

- To continue execution, set a status variable in a `CONTINUE` handler that can be checked in the enclosing block to determine whether the handler was invoked. The following example uses the variable `done` for this purpose:

  ```
  CREATE PROCEDURE p ()
  BEGIN
    DECLARE i INT DEFAULT 3;
    DECLARE done INT DEFAULT FALSE;
    retry:
      REPEAT
        BEGIN
          DECLARE CONTINUE HANDLER FOR SQLWARNING
            BEGIN
              SET done = TRUE;
            END;
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        IF done OR i < 0 THEN
          LEAVE retry;
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

13.7 Database Administration Statements

13.7.1 Account Management Statements

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in Chapter 5, *MySQL Server Administration*, which you should consult for additional details.

Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. To ensure that you can take advantage of any new capabilities, update your grant tables to have the current structure whenever you update to a new version of MySQL. See Section 4.4.9, "`mysql_upgrade` — Check Tables for MySQL Upgrade".

13.7.1.1 CREATE USER Syntax

```
CREATE USER user_specification [, user_specification] ...

user_specification:
    user [ identified_option ]

identified_option: {
    IDENTIFIED BY 'auth_string'
  | IDENTIFIED BY PASSWORD 'hash_string'
}
```

The `CREATE USER` statement creates new MySQL accounts. An error occurs if you try to create an account that already exists. To use this statement, you must have the global `CREATE USER` privilege or the `INSERT` privilege for the `mysql` database. For each account, `CREATE USER` creates a new row in the `mysql.user` table with no privileges. Depending on the syntax used, `CREATE USER` may also assign the account a password.

Each *`user_specification`* clause consists of an account name and information about how authentication occurs for clients that use the account. This part of `CREATE USER` syntax is shared with `GRANT`, so the description here applies to `GRANT` as well.

Each account name uses the format described in Section 6.2.3, "Specifying Account Names". For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

`CREATE USER` examples:

- To enable the user to connect with no password, include no `IDENTIFIED BY` clause:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE USER 'jeffrey'@'localhost';
```

- To assign a password, use `IDENTIFIED BY` with the literal cleartext password value:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

- To avoid specifying the cleartext password if you know its hash value (the value that `PASSWORD()` would return for the password), specify the hash value preceded by the keyword `PASSWORD`:

```
CREATE USER 'jeffrey'@'localhost'
IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
```

For additional information about setting passwords, see Section 6.3.5, “Assigning Account Passwords”.

Important

`CREATE USER` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about password logging in the server logs, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Logging”.

The `CREATE USER` statement was added in MySQL 5.0.2.

13.7.1.2 DROP USER Syntax

```
DROP USER user [, user] ...
```

The `DROP USER` statement removes one or more MySQL accounts. An error occurs for accounts that do not exist. To use this statement, you must have the global `CREATE USER` privilege or the `DELETE` privilege for the `mysql` database.

Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. For example:

```
DROP USER 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

`DROP USER` as present in MySQL 5.0.0 removes only accounts that have no privileges. In MySQL 5.0.2, it was modified to remove account privileges as well. This means that the procedure for removing an account depends on your version of MySQL.

As of MySQL 5.0.2, you can remove an account and its privileges as follows:

```
DROP USER user;
```

The statement removes privilege rows for the account from all grant tables.

Before MySQL 5.0.2, `DROP USER` serves only to remove account rows from the `user` table for accounts that have no privileges. To remove a MySQL account completely (including all of its privileges), you should use the following procedure, performing these steps in the order shown:

1. Use `SHOW GRANTS` to determine what privileges the account has. See Section 13.7.5.17, “SHOW GRANTS Syntax”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. Use `REVOKE` to revoke the privileges displayed by `SHOW GRANTS`. This removes rows for the account from all the grant tables except the `user` table, and revokes any global privileges listed in the `user` table. See Section 13.7.1.3, “GRANT Syntax”.

3. Delete the account by using `DROP USER` to remove the `user` table row.

Important

`DROP USER` does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by design*.

`DROP USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the dropped user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 18.5, “Access Control for Stored Programs and Views”.)

13.7.1.3 GRANT Syntax

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [, user_specification] ...
    [REQUIRE {NONE | tsl_option [[AND] tsl_option] ...}]
    [WITH {GRANT OPTION | resource_option} ...]

object_type: {
    TABLE
  | FUNCTION
  | PROCEDURE
}

priv_level: {
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}

user_specification:
    user [ auth_option ]

auth_option: {
    IDENTIFIED BY 'auth_string'
  | IDENTIFIED BY PASSWORD 'hash_string'
}

tsl_option: {
    SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'
}

resource_option: {
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  | MAX_USER_CONNECTIONS count
}
```

The GRANT statement grants privileges to MySQL user accounts. To use GRANT, you must have the GRANT OPTION privilege, and you must have the privileges that you are granting.

GRANT also serves to specify other account characteristics such as use of secure connections and limits on access to server resources.

The REVOKE statement is related to GRANT and enables administrators to remove account privileges. See Section 13.7.1.5, “REVOKE Syntax”.

Normally, a database administrator first uses CREATE USER to create an account, then GRANT to define its privileges and characteristics. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
GRANT USAGE ON *.* TO 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

Note

Examples shown here include no IDENTIFIED clause. It is assumed that you establish passwords with CREATE USER at account-creation time to avoid creating insecure accounts.

If an account named in a GRANT statement does not already exist, GRANT may create it under the conditions described later in the discussion of the NO_AUTO_CREATE_USER SQL mode.

From the mysql program, GRANT responds with Query OK, 0 rows affected when executed successfully. To determine what privileges result from the operation, use SHOW GRANTS. See Section 13.7.5.17, “SHOW GRANTS Syntax”.

There are several aspects to the GRANT statement, described under the following topics in this section:

- Privileges Supported by MySQL
- Global Privileges
- Database Privileges
- Table Privileges
- Column Privileges
- Stored Routine Privileges
- Account Names and Passwords
- Implicit Account Creation
- Other Account Characteristics
- MySQL and Standard SQL Versions of GRANT

GRANT supports host names up to 60 characters long. Database, table, column, and routine names can be up to 64 characters. User names can be up to 16 characters.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Warning

The permissible length for user names cannot be changed by altering the `mysql.user` table. Attempting to do so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server. You should never alter the structure of tables in the `mysql` database in any manner whatsoever except by means of the procedure described in Section 4.4.9, "`mysql_upgrade` — Check Tables for MySQL Upgrade".

Important

`GRANT` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about password logging in the server logs, see Section 6.1.2.3, "Passwords and Logging". For similar information about client-side logging, see Section 4.5.1.3, "mysql Logging".

Privileges Supported by MySQL

The following table summarizes the permissible *`priv_type`* privilege types that can be specified for the `GRANT` and `REVOKE` statements, and the levels at which each privilege can be granted. For additional information about these privileges, see Section 6.2.1, "Privileges Provided by MySQL".

Table 13.2 Permissible Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
`ALL [PRIVILEGES]`	Grant all privileges at specified access level except `GRANT OPTION`
`ALTER`	Enable use of `ALTER TABLE`. Levels: Global, database, table.
`ALTER ROUTINE`	Enable stored routines to be altered or dropped. Levels: Global, database, procedure.
`CREATE`	Enable database and table creation. Levels: Global, database, table.
`CREATE ROUTINE`	Enable stored routine creation. Levels: Global, database.
`CREATE TEMPORARY TABLES`	Enable use of `CREATE TEMPORARY TABLE`. Levels: Global, database.
`CREATE USER`	Enable use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`. Level: Global.
`CREATE VIEW`	Enable views to be created or altered. Levels: Global, database, table.
`DELETE`	Enable use of `DELETE`. Level: Global, database, table.
`DROP`	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
`EXECUTE`	Enable the user to execute stored routines. Levels: Global, database, table.
`FILE`	Enable the user to cause the server to read or write files. Level: Global.
`GRANT OPTION`	Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, procedure.
`INDEX`	Enable indexes to be created or dropped. Levels: Global, database, table.
`INSERT`	Enable use of `INSERT`. Levels: Global, database, table, column.
`LOCK TABLES`	Enable use of `LOCK TABLES` on tables for which you have the `SELECT` privilege. Levels: Global, database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Privilege	Meaning and Grantable Levels
PROCESS	Enable the user to see all processes with SHOW PROCESSLIST. Level: Global.
REFERENCES	Not implemented
RELOAD	Enable use of FLUSH operations. Level: Global.
REPLICATION CLIENT	Enable the user to ask where master or slave servers are. Level: Global.
REPLICATION SLAVE	Enable replication slaves to read binary log events from the master. Level: Global.
SELECT	Enable use of SELECT. Levels: Global, database, table, column.
SHOW DATABASES	Enable SHOW DATABASES to show all databases. Level: Global.
SHOW VIEW	Enable use of SHOW CREATE VIEW. Levels: Global, database, table.
SHUTDOWN	Enable use of mysqladmin shutdown. Level: Global.
SUPER	Enable use of other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin debug command. Level: Global.
UPDATE	Enable use of UPDATE. Levels: Global, database, table, column.
USAGE	Synonym for “no privileges”

The EXECUTE privilege is not operational until MySQL 5.0.3. CREATE VIEW and SHOW VIEW were added in MySQL 5.0.1. CREATE USER, CREATE ROUTINE, and ALTER ROUTINE were added in MySQL 5.0.3.

In GRANT statements, the ALL [PRIVILEGES] privilege is named by itself and cannot be specified along with other privileges. It stands for all privileges available for the level at which privileges are to be granted except for the GRANT OPTION privilege.

USAGE can be specified to create a user that has no privileges, or to specify the REQUIRE or WITH clauses for an account without changing its existing privileges.

MySQL account information is stored in the tables of the mysql database. For additional details, consult Section 6.2, “The MySQL Access Privilege System”, which discusses the mysql database and the access control system extensively.

If the grant tables hold privilege rows that contain mixed-case database or table names and the lower_case_table_names system variable is set to a nonzero value, REVOKE cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (GRANT will not create such rows when lower_case_table_names is set, but such rows might have been created prior to setting that variable.)

Privileges can be granted at several levels, depending on the syntax used for the ON clause. For REVOKE, the same ON syntax specifies which privileges to remove.

For the global, database, table, and routine levels, GRANT ALL assigns only the privileges that exist at the level you are granting. For example, GRANT ALL ON *db_name*.* is a database-level statement, so it does not grant any global-only privileges such as FILE. Granting ALL does not assign the GRANT OPTION privilege.

The *object_type* clause was added in MySQL 5.0.6. If present, it should be specified as TABLE, FUNCTION, or PROCEDURE when the following object is a table, a stored function, or a stored procedure.

The privileges for a database, table, column, or routine are formed additively as the logical OR of the privileges at each of the privilege levels. For example, if a user has a global SELECT privilege, the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

privilege cannot be denied by an absence of the privilege at the database, table, or column level. Details of the privilege-checking procedure are presented in Section 6.2.5, “Access Control, Stage 2: Request Verification”.

If you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL enables you to grant privileges on databases or tables that do not exist. For tables, the privileges to be granted must include the `CREATE` privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases or tables that are to be created at a later time.

Important

MySQL does not automatically revoke any privileges when you drop a database or table. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use `ON *.*` syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

Before MySQL 5.0.23, privileges also are assigned at the global level if you use `ON *` syntax and you have *not* selected a default database. As of 5.0.23, `ON *` requires a default database and produces an error if there is none.

The `CREATE USER`, `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN`, and `SUPER` privileges are administrative and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

MySQL stores global privileges in the `mysql.user` table.

Database Privileges

Database privileges apply to all objects in a given database. To assign database-level privileges, use `ON *db_name*.*` syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use `ON *` syntax (rather than `ON *.*`) and you have selected a default database, privileges are assigned at the database level for the default database. An error occurs if there is no default database.

The `CREATE`, `DROP`, `GRANT OPTION`, and `LOCK TABLES` privileges can be specified at the database level. Table or routine privileges also can be specified at the database level, in which case they apply to all tables or routines in the database.

MySQL stores database privileges in the `mysql.db` table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use `ON db_name.tbl_name` syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify *`tbl_name`* rather than *`db_name.tbl_name`*, the statement applies to *`tbl_name`* in the default database. An error occurs if there is no default database.

The permissible *`priv_type`* values at the table level are `ALTER`, `CREATE VIEW`, `CREATE`, `DELETE`, `DROP`, `GRANT OPTION`, `INDEX`, `INSERT`, `SELECT`, `SHOW VIEW`, and `UPDATE`.

MySQL stores table privileges in the `mysql.tables_priv` table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values for a column (that is, when you use a *`column_list`* clause) are `INSERT`, `SELECT`, and `UPDATE`.

MySQL stores column privileges in the `mysql.columns_priv` table.

Stored Routine Privileges

The `ALTER ROUTINE`, `CREATE ROUTINE`, `EXECUTE`, and `GRANT OPTION` privileges apply to stored routines (procedures and functions). They can be granted at the global and database levels. Except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have this privilege to create a routine in the first place.

MySQL stores routine-level privileges in the `mysql.procs_priv` table.

Account Names and Passwords

The *`user_specification`* clause names a user and optionally provides authentication information such as a password.

The *`user`* value indicates the MySQL account to which the `GRANT` statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *`user`* value in the form *`user_name@host_name`*. If a *`user_name`* or *`host_name`* value is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a *`user_name`* string containing special characters (such as "`-`"), or a *`host_name`* string containing special characters or wildcard characters (such as "`%`"); for example, `'test-user'@'%.com'`. Quote the user name and host name separately.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can specify wildcards in the host name. For example, *user_name*@'%.example.com' applies to *user_name* for any host in the example.com domain, and *user_name*@'192.168.1.%' applies to *user_name* for any host in the 192.168.1 class C subnet.

The simple form *user_name* is a synonym for *user_name*@'%'.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the GRANT statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...;
```

In this case, any user who connects from the local host with the correct password for the anonymous user will be permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see Section 6.2.3, “Specifying Account Names”.

To specify quoted values, quote database, table, column, and routine names as identifiers. Quote user names and host names as identifiers or as strings. Quote passwords as strings. For string-quoting and identifier-quoting guidelines, see Section 9.1.1, “String Literals”, and Section 9.2, “Schema Object Names”.

The “_” and “%” wildcards are permitted when specifying database names in GRANT statements that grant privileges at the global or database levels. This means, for example, that if you want to use a “_” character as part of a database name, you should specify it as “_” in the GRANT statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, GRANT ... ON `foo\_bar`.* TO ....

Warning

If you permit anonymous users to connect to the MySQL server, you should also grant privileges to all local users as *user_name*@localhost. Otherwise, the anonymous user account for localhost in the mysql.user table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see Section 6.2.4, “Access Control, Stage 1: Connection Verification”.

To determine whether the preceding warning applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

To indicate how a user should authenticate when connecting to the server, the *user_specification* value may include an IDENTIFIED clause to specify a password. Syntax of the user specification is the same as for the CREATE USER statement. For details, see Section 13.7.1.1, “CREATE USER Syntax”.

When IDENTIFIED BY is present and you have the global grant privilege (GRANT OPTION), the password becomes the new password for the account, even if the account exists and already has a password. Without IDENTIFIED BY, the account password remains unchanged.

If the NO_AUTO_CREATE_USER SQL mode is not enabled and the account named in a GRANT statement does not exist in the mysql.user table, GRANT creates it. If you specify no IDENTIFIED BY clause or provide an empty password, the user has no password. *This is very insecure.*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If NO_AUTO_CREATE_USER is enabled and the account does not exist, GRANT fails and does not create the account unless the IDENTIFIED BY clause is given to provide a nonempty password.

Other Account Characteristics

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the user name and credentials. For background information on the use of SSL with MySQL, see Section 6.3.6, “Using Secure Connections”.

The optional REQUIRE clause specifies SSL-related options for a MySQL account, using one or more *tsl_option* values.

Implicit Account Creation

GRANT permits these *tsl_option* values:

- NONE

  Indicates that the account has no SSL or X509 requirements. Unencrypted connections are permitted if the user name and password are valid. However, encrypted connections can also be used, at the client's option, if the client has the proper certificate and key files. That is, the client need not specify any SSL command options, in which case the connection will be unencrypted. To use an encrypted connection, the client must specify either the --ssl-ca option, or all three of the --ssl-ca, --ssl-key, and --ssl-cert options.

  NONE is the default if no SSL-related REQUIRE options are specified.

- SSL

  Tells the server to permit only encrypted connections for the account.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    REQUIRE SSL;
  ```

  To connect, the client must specify the --ssl-ca option to authenticate the server certificate, and may additionally specify the --ssl-key and --ssl-cert options. If neither the --ssl-ca option nor --ssl-capath option is specified, the client does not authenticate the server certificate.

- X509

  Requires that the client must have a valid certificate but the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates. Use of X509 certificates always implies encryption, so the SSL option is unnecessary in this case.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    REQUIRE X509;
  ```

  The client must specify the --ssl-key and --ssl-cert options to connect. (It is recommended but not required that --ssl-ca also be specified so that the public certificate provided by the server can be verified.) This is true for ISSUER and SUBJECT as well because those REQUIRE options imply the requirements of X509.

- ISSUER '*issuer*'

  Places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA '*issuer*'. If the client presents a certificate that is valid but has a different issuer, the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

Because `ISSUER` implies the requirements of `X509`, the client must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL/CN=CA/emailAddress=ca@example.com';
```

Note

If MySQL is linked against a version of OpenSSL older than 0.9.6h, use `Email` rather than `emailAddress` in the *`'issuer'`* value.

- `SUBJECT '`*`subject`*`'`

  Places the restriction on connection attempts that the client must present a valid X509 certificate containing the subject *`subject`*. If the client presents a certificate that is valid but has a different subject, the server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  Because `SUBJECT` implies the requirements of `X509`, the client must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
      O=MySQL demo client certificate/
      CN=client/emailAddress=client@example.com';
  ```

  MySQL does a simple string comparison of the *`'subject'`* value to the value in the certificate, so lettercase and component ordering must be given exactly as present in the certificate.

  Note

  Regarding `emailAddress`, see the note in the description of `REQUIRE ISSUER`.

- `CIPHER '`*`cipher`*`'`

  Requests a specific cipher method for encrypting connections. This option is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
  ```

The `SUBJECT`, `ISSUER`, and `CIPHER` options can be combined in the `REQUIRE` clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL demo client certificate/
    CN=client/emailAddress=client@example.com'
  AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL/CN=CA/emailAddress=ca@example.com'
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The order of the options does not matter, but no option can be specified twice. The `AND` keyword is optional between `REQUIRE` options.

The optional `WITH` clause is used for these purposes:

- To enable a user to grant privileges to other users
- To specify resource limits for a user

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level.

To grant the `GRANT OPTION` privilege to an account without otherwise changing its privileges, do this:

```
GRANT USAGE ON *.* TO 'someuser'@'somehost' WITH GRANT OPTION;
```

Be careful to whom you give the `GRANT OPTION` privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` privilege on a database. If you then grant the `SELECT` privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` privilege, but also `INSERT`. If you then grant the `UPDATE` privilege to the user on the database, the user can grant `INSERT`, `SELECT`, and `UPDATE`.

For a nonadministrative user, you should not grant the `ALTER` privilege globally or for the `mysql` database. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see Section 6.2.1, “Privileges Provided by MySQL”.

It is possible to place limits on use of server resources by an account, as discussed in Section 6.3.4, “Setting Account Resource Limits”. To do so, use a `WITH` clause that specifies one or more *`resource_option`* values. Limits not specified retain their current values.

`GRANT` permits these *`resource_option`* values:

- `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, `MAX_CONNECTIONS_PER_HOUR` *`count`*

  These options restrict the number of queries, updates, and connections to the server permitted to this account during any given one-hour period. (Queries for which results are served from the query cache do not count against the `MAX_QUERIES_PER_HOUR` limit.) If *`count`* is `0` (the default), this means that there is no limitation for the account.

- `MAX_USER_CONNECTIONS` *`count`*

  Restricts the maximum number of simultaneous connections to the server by the account. A nonzero *`count`* specifies the limit for the account explicitly. If *`count`* is `0` (the default), the server determines the number of simultaneous connections for the account from the global value of the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`max_user_connections` system variable. If `max_user_connections` is also zero, there is no limit for the account.

If a given resource limit is specified multiple times, the last instance takes precedence.

To specify resource limits for an existing user without affecting existing privileges, use `GRANT USAGE` at the global level (`ON *.*`) and name the limits to be changed. For example:

```
GRANT USAGE ON *.* TO ...
  WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

MySQL and Standard SQL Versions of GRANT

The biggest differences between the MySQL and standard SQL versions of `GRANT` are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL `UNDER` privilege, and does not support the `TRIGGER` privilege until MySQL 5.1.6.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL 5.0.2 and up if you use `DROP USER`. Before 5.0.2, the granted privileges are not automatically revoked; you must revoke them yourself. See Section 13.7.1.2, “DROP USER Syntax”.
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped only with explicit `DROP USER` or `REVOKE` statements or by manipulating the MySQL grant tables directly.
- In MySQL, it is possible to have the `INSERT` privilege for only some of the columns in a table. In this case, you can still execute `INSERT` statements on the table, provided that you insert values only for those columns for which you have the `INSERT` privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the `INSERT` privilege on all columns.) For information about strict SQL mode and implicit default values, see Section 5.1.7, “Server SQL Modes”, and Section 11.6, “Data Type Default Values”.

13.7.1.4 RENAME USER Syntax

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

The `RENAME USER` statement renames existing MySQL accounts. An error occurs for old accounts that do not exist or new accounts that already exist. To use this statement, you must have the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` database.

Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. For example:

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`RENAME USER` causes the privileges held by the old user to be those held by the new user. However, `RENAME USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the old user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 18.5, “Access Control for Stored Programs and Views”.)

The privilege changes take effect as indicated in Section 6.2.6, “When Privilege Changes Take Effect”.

The `RENAME USER` statement was added in MySQL 5.0.2.

13.7.1.5 REVOKE Syntax

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...
```

The `REVOKE` statement enables system administrators to revoke privileges from MySQL accounts. Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. For example:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

For details on the levels at which privileges exist, the permissible *`priv_type`* and *`priv_level`* values, and the syntax for specifying users and passwords, see Section 13.7.1.3, “GRANT Syntax”

To use the first `REVOKE` syntax, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this `REVOKE` syntax, you must have the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` database.

`REVOKE` removes privileges, but does not drop `mysql.user` table entries. To remove a user account entirely, use `DROP USER` (see Section 13.7.1.2, “DROP USER Syntax”) or `DELETE`.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting the variable.)

When successfully executed from the `mysql` program, `REVOKE` responds with `Query OK, 0 rows affected`. To determine what privileges result from the operation, use `SHOW GRANTS`. See Section 13.7.5.17, “SHOW GRANTS Syntax”.

13.7.1.6 SET PASSWORD Syntax

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SET PASSWORD [FOR user] = password_option

password_option: {
    PASSWORD('auth_string')
  | OLD_PASSWORD('auth_string')
  | 'hash_string'
}
```

The `SET PASSWORD` statement assigns a password to a MySQL user account, specified as either a cleartext (unencrypted) or encrypted value:

- `'auth_string'` represents a cleartext password.
- `'hash_string'` represents an encrypted password.

`SET PASSWORD` can be used with or without an explicitly named user account:

- With a `FOR user` clause, the statement sets the password for the named account, which must exist:

  ```
  SET PASSWORD FOR 'jeffrey'@'localhost' = password_option;
  ```

  In this case, you must have the `UPDATE` privilege for the `mysql` database.

- With no `FOR user` clause, the statement sets the password for the current user:

  ```
  SET PASSWORD = password_option;
  ```

  Any client who connects to the server using a nonanonymous account can change the password for that account. To see which account the server authenticated you as, invoke the `CURRENT_USER()` function:

  ```
  SELECT CURRENT_USER();
  ```

In MySQL 5.0, enabling the `read_only` system variable does not prevent use of `SET PASSWORD`.

If a `FOR user` clause is given, the account name uses the format described in Section 6.2.3, “Specifying Account Names”. The `user` value should be given as `'user_name'@'host_name'`, where `'user_name'` and `'host_name'` are exactly as listed in the `User` and `Host` columns of the account's `mysql.user` table row. If you specify only a user name, a host name of `'%'` is used. For example, to set the password for an account with `User` and `Host` column values of `'bob'` and `'%.example.org'`, write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.example.org' = PASSWORD('auth_string');
```

The password can be specified in these ways:

- Using the `PASSWORD()` function

  The `'auth_string'` function argument is the cleartext (unencrypted) password. `PASSWORD()` hashes the password and returns the encrypted password string for storage in the `mysql.user` account row.

  The `PASSWORD()` function hashes the password using the hashing method determined by the value of the `old_passwords` system variable value. For descriptions of the permitted values, see Section 5.1.4, “Server System Variables”.

- Using the `OLD_PASSWORD()` function:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The *'auth_string'* function argument is the cleartext (unencrypted) password. OLD_PASSWORD() hashes the password using pre-4.1 hashing and returns the encrypted password string for storage in the mysql.user account row.

- Using an already encrypted password string

  The password is specified as a string literal. It must represent the already encrypted password value, in the hash format required by the authentication method used for the account.

For more information about setting passwords, see Section 6.3.5, “Assigning Account Passwords”.

Important

SET PASSWORD may be recorded in server logs or on the client side in a history file such as ~/.mysql_history, which means that cleartext passwords may be read by anyone having read access to that information. For information about password logging in the server logs, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Logging”.

Caution

If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not change your password without first reading Section 6.1.2.4, “Password Hashing in MySQL”. The default password hashing format changed in MySQL 4.1, and if you change your password, it might be stored using a hashing format that pre-4.1 clients cannot generate, thus preventing you from connecting to the server afterward.

If you are using MySQL Replication, be aware that, currently, a password used by a replication slave as part of a CHANGE MASTER TO statement is effectively limited to 32 characters in length; if the password is longer, any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

13.7.2 Table Maintenance Statements

13.7.2.1 ANALYZE TABLE Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

ANALYZE TABLE analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for MyISAM, BDB, and InnoDB. This statement works with MyISAM, BDB, InnoDB, and NDB tables. For MyISAM tables, this statement is equivalent to using myisamchk --analyze. This statement does not work with views.

For more information on how the analysis works within InnoDB, see Section 14.2.14, “Limits on InnoDB Tables”.

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

This statement requires SELECT and INSERT privileges for the table.

ANALYZE TABLE returns a result set with the following columns.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Column	Value
`Table`	The table name
`Op`	Always `analyze`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

You can check the stored key distribution with the `SHOW INDEX` statement. See Section 13.7.5.18, “SHOW INDEX Syntax”.

If the table has not changed since the last `ANALYZE TABLE` statement, the table is not analyzed again.

By default, the server writes `ANALYZE TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

13.7.2.2 BACKUP TABLE Syntax

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory'
```

Note

This statement is deprecated and is removed in MySQL 5.5. As an alternative, `mysqldump` or `mysqlhotcopy` can be used instead.

`BACKUP TABLE` copies to the backup directory the minimum number of table files needed to restore the table, after flushing any buffered changes to disk. The statement works only for `MyISAM` tables; it does not work for views. `BACKUP TABLE` works by copying the table's `.frm` definition and `.MYD` data files. The `.MYI` index file can be rebuilt from those two files. The directory should be specified as a full path name. To restore the table, use `RESTORE TABLE`.

During the backup, a read lock is held for each table, one at time, as they are being backed up. If you want to back up several tables as a snapshot (preventing any of them from being changed during the backup operation), issue a `LOCK TABLES` statement first, to obtain a read lock for all tables in the group.

`BACKUP TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `backup`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

13.7.2.3 CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {
    FOR UPGRADE
  | QUICK
  | FAST
  | MEDIUM
  | EXTENDED
  | CHANGED
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
}
```

`CHECK TABLE` checks a table or tables for errors. `CHECK TABLE` works for `MyISAM`, `InnoDB`, and (as of MySQL 5.0.16) `ARCHIVE` tables. For `MyISAM` tables, the key statistics are updated as well.

As of MySQL 5.0.2, `CHECK TABLE` can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

`CHECK TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `check`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

Note that the statement might produce many rows of information for each checked table. The last row has a `Msg_type` value of `status` and the `Msg_text` normally should be `OK`. If you don't get `OK`, or `Table is already up to date` you should or `Table is already up to date` for a `MyISAM` table, you should normally run a repair of the table. See Section 7.6, “MyISAM Table Maintenance and Crash Recovery”. `Table is already up to date` means that the storage engine for the table indicated that there was no need to check the table.

The `FOR UPGRADE` option checks whether the named tables are compatible with the current version of MySQL. This option was added in MySQL 5.0.19. With `FOR UPGRADE`, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table (which might take some time). If the full check succeeds, the server marks the table's `.frm` file with the current MySQL version number. Marking the `.frm` file ensures that further checks for the table with the same version of the server will be fast.

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

`FOR UPGRADE` discovers these incompatibilities:

- The indexing order for end-space in `TEXT` columns for `InnoDB` and `MyISAM` tables changed between MySQL 4.1 and 5.0.
- The storage method of the new `DECIMAL` data type changed between MySQL 5.0.3 and 5.0.5.
- As of MySQL 5.0.62, if your table was created by a different version of the MySQL server than the one you are currently running, `FOR UPGRADE` indicates that the table has an `.frm` file with an incompatible version. In this case, the result set returned by `CHECK TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Table upgrade required. Please do "REPAIR TABLE` *`tbl_name`* `" to fix it!`
- Changes are sometimes made to character sets or collations that require table indexes to be rebuilt. For details about these changes and when `FOR UPGRADE` detects them, see Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.

The following table shows the other check options that can be given. These options are passed to the storage engine, which may use them or not. `MyISAM` uses them; they are ignored for `InnoDB` tables and views.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Type	Meaning
QUICK	Do not scan the rows to check for incorrect links.
FAST	Check only tables that have not been closed properly.
CHANGED	Check only tables that have been changed since the last check or that have not been closed properly.
MEDIUM	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys.
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time.

If none of the options QUICK, MEDIUM, or EXTENDED are specified, the default check type for dynamic-format MyISAM tables is MEDIUM. This has the same result as running myisamchk --medium-check *tbl_name* on the table. The default check type also is MEDIUM for static-format MyISAM tables, unless CHANGED or FAST is specified. In that case, the default is QUICK. The row scan is skipped for CHANGED and FAST because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note

CHECK TABLE may change the table if the table is marked as “corrupted” or “not closed properly” but CHECK TABLE does not find any problems in the table. In this case, CHECK TABLE marks the table as okay.

If a table is corrupted, the problem is most likely in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the QUICK option. The latter should be used when you are in a hurry and can take the very small risk that QUICK does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

FAST and CHANGED are mostly intended to be used from a script (for example, to be executed from cron) if you want to check tables from time to time. In most cases, FAST is to be preferred over CHANGED. (The only case when it is not preferred is when you suspect that you have found a bug in the MyISAM code.)

EXTENDED is to be used only after you have run a normal check but still get strange errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of CHECK TABLE ... EXTENDED might influence the execution plan generated by the query optimizer.

Some problems reported by CHECK TABLE cannot be corrected automatically:

- Found row where the auto_increment column has the value 0.

  This means that you have a row in the table where the AUTO_INCREMENT index column contains the value 0. (It is possible to create a row where the AUTO_INCREMENT column is 0 by explicitly setting the column to 0 with an UPDATE statement.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an `ALTER TABLE` on the table. In this case, the `AUTO_INCREMENT` column changes value according to the rules of `AUTO_INCREMENT` columns, which could cause problems such as a duplicate-key error.

To get rid of the warning, execute an `UPDATE` statement to set the column to some value other than 0.

- If `CHECK TABLE` finds a problem for an `InnoDB` table, the server shuts down to prevent error propagation. Details of the error will be written to the error log.

13.7.2.4 CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

`CHECKSUM TABLE` reports a table checksum.

This statement is not supported for views. If you run `CHECKSUM TABLE` against a view, the `Checksum` column value is always `NULL`, and a warning is returned.

With `QUICK`, the live table checksum is reported if it is available, or `NULL` otherwise. This is very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option when you create the table; currently, this is supported only for `MyISAM` tables. See Section 13.1.10, “CREATE TABLE Syntax”.

With `EXTENDED`, the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither `QUICK` nor `EXTENDED` is specified, MySQL returns a live checksum if the table storage engine supports it and scans the table otherwise.

For a nonexistent table, `CHECKSUM TABLE` returns `NULL` and, as of MySQL 5.0.3, generates a warning.

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the storage format for `VARCHAR` changed between MySQL 4.1 and 5.0, so if a 4.1 table is upgraded to MySQL 5.0, the checksum value may change.

Important

If the checksums for two tables are different, then it is almost certain that the tables are different in some way. However, because the hashing function used by `CHECKSUM TABLE` is not guaranteed to be collision-free, there is a slight chance that two tables which are not identical can produce the same checksum.

13.7.2.5 OPTIMIZE TABLE Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

`OPTIMIZE TABLE` should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have `VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT` columns). Deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file. After extensive changes to a table, this statement may also improve performance of statements that use the table, sometimes significantly. This statement does not work with views.

This statement requires `SELECT` and `INSERT` privileges for the table.

`OPTIMIZE TABLE` works for `MyISAM`, `InnoDB`, and (as of MySQL 5.0.16) `ARCHIVE` tables.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For BDB tables, OPTIMIZE TABLE currently is mapped to ANALYZE TABLE. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.

For InnoDB tables, OPTIMIZE TABLE is mapped to ALTER TABLE, which rebuilds the table to update index statistics and free unused space in the clustered index.

By default, OPTIMIZE TABLE does *not* work for tables created using any other storage engine and returns a result indicating this lack of support. You can make OPTIMIZE TABLE work for other storage engines by starting mysqld with the --skip-new option. In this case, OPTIMIZE TABLE is just mapped to ALTER TABLE.

For MyISAM tables, OPTIMIZE TABLE works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

OPTIMIZE TABLE returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always optimize
Msg_type	status, error, info, note, or warning
Msg_text	An informational message

Note that MySQL locks the table during the time OPTIMIZE TABLE is running.

By default, the server writes OPTIMIZE TABLE statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

OPTIMIZE TABLE does not sort R-tree indexes, such as spatial indexes on POINT columns. (Bug #23578)

13.7.2.6 REPAIR TABLE Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE repairs a possibly corrupted table. By default, it has the same effect as myisamchk --recover *tbl_name*. REPAIR TABLE works for MyISAM and for ARCHIVE tables. See Section 14.1, “The MyISAM Storage Engine”, and Section 14.8, “The ARCHIVE Storage Engine”. This statement does not work with views.

This statement requires SELECT and INSERT privileges for the table.

Normally, you should never have to run this statement. However, if disaster strikes, REPAIR TABLE is very likely to get back all your data from a MyISAM table. If your tables become corrupted often, you should try to find the reason for it, to eliminate the need to use REPAIR TABLE. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”, and Section 14.1.4, “MyISAM Table Problems”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors. See Chapter 7, *Backup and Recovery*.

Warning

If the server crashes during a `REPAIR TABLE` operation, it is essential after restarting it that you immediately execute another `REPAIR TABLE` statement for the table before performing any other operations on it. In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario that underscores the value of making a backup first.

`REPAIR TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `repair`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

The `REPAIR TABLE` statement might produce many rows of information for each repaired table. The last row has a `Msg_type` value of `status` and `Msg_test` normally should be `OK`. If you do not get `OK` for a `MyISAM` table, you should try repairing it with `myisamchk --safe-recover`. (`REPAIR TABLE` does not implement all the options of `myisamchk`.) With `myisamchk --safe-recover`, you can also use options that `REPAIR TABLE` does not support, such as `--max-record-length`.

If you use the `QUICK` option, `REPAIR TABLE` tries to repair only the index file, and not the data file. This type of repair is like that done by `myisamchk --recover --quick`.

If you use the `EXTENDED` option, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by `myisamchk --safe-recover`.

The `USE_FRM` option is available for use if the `.MYI` index file is missing or if its header is corrupted. This option tells MySQL not to trust the information in the `.MYI` file header and to re-create it using information from the `.frm` file. This kind of repair cannot be done with `myisamchk`.

Note

Use the `USE_FRM` option *only* if you cannot use regular `REPAIR` modes! Telling the server to ignore the `.MYI` file makes important table metadata stored in the `.MYI` unavailable to the repair process, which can have deleterious consequences:

- The current `AUTO_INCREMENT` value is lost.
- The link to deleted records in the table is lost, which means that free space for deleted records will remain unoccupied thereafter.
- The `.MYI` header indicates whether the table is compressed. If the server ignores this information, it cannot tell that a table is compressed and repair can cause change or loss of table contents. This means that `USE_FRM` should not be used

This documentation is for an older version. If you're
This documentation is for an older version. If you're

with compressed tables. That should not be necessary, anyway: Compressed tables are read only, so they should not become corrupt.

Caution

As of MySQL 5.0.62, if you use `USE_FRM` for a table that was created by a different version of the MySQL server than the one you are currently running, `REPAIR TABLE` will not attempt to repair the table. In this case, the result set returned by `REPAIR TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Failed repairing incompatible .FRM file`.

Prior to MySQL 5.0.62, *do not use* `USE_FRM` if your table was created by a different version of the MySQL server. Doing so risks the loss of all rows in the table. It is particularly dangerous to use `USE_FRM` after the server returns this message:

```
Table upgrade required. Please do
"REPAIR TABLE `tbl_name`" to fix it!
```

If `USE_FRM` is *not* used, `REPAIR TABLE` checks the table to see whether an upgrade is required. If so, it performs the upgrade, following the same rules as `CHECK TABLE ... FOR UPGRADE`. See Section 13.7.2.3, “CHECK TABLE Syntax”, for more information. As of MySQL 5.0.62, `REPAIR TABLE` without `USE_FRM` upgrades the `.frm` file to the current version.

By default, the server writes `REPAIR TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

Important

In the event that a table on the master becomes corrupted and you run `REPAIR TABLE` on it, any resulting changes to the original table are *not* propagated to slaves.

You may be able to increase `REPAIR TABLE` performance by setting certain system variables. See Section 8.5.3, “Speed of REPAIR TABLE Statements”.

13.7.2.7 RESTORE TABLE Syntax

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

Note

This statement is deprecated and is removed in MySQL 5.5.

`RESTORE TABLE` restores the table or tables from a backup that was made with `BACKUP TABLE`. The directory should be specified as a full path name.

Existing tables are not overwritten; if you try to restore over an existing table, an error occurs. Just as for `BACKUP TABLE`, `RESTORE TABLE` currently works only for `MyISAM` tables. Restored tables are not replicated from master to slave. If there is an existing view of the same name, the view is overwritten (views cannot be backed up using `BACKUP TABLE`).

The backup for each table consists of its `.frm` format file and `.MYD` data file. The restore operation restores those files, and then uses them to rebuild the `.MYI` index file. Restoring takes longer than backing up due to the need to rebuild the indexes. The more indexes the table has, the longer it takes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`RESTORE TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `restore`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

13.7.3 User-Defined Function Statements

13.7.3.1 CREATE FUNCTION Syntax for User-defined Functions

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL|DECIMAL}
    SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`.

`function_name` is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. As of MySQL 5.0.3, `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

`shared_library_name` is the base name of the shared object file that contains the code that implements the function. As of MySQL 5.0.67, the file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` system variable. If the value of `plugin_dir` is empty, the behavior that is used before 5.0.67 applies: The file must be located in a directory that is searched by your system's dynamic linker. For more information, see Section 21.2.2.5, “UDF Compiling and Installing”.

To create a function, you must have the `INSERT` privilege for the `mysql` database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name. If you do not have this table, you should run the `mysql_upgrade` command to create it. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see Section 21.2.2, “Adding a New User-Defined Function”. For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()`. For `AGGREGATE` to work, your `mysql.func` table must contain a `type` column. If your `mysql.func` table does not have this column, you should run the `mysql_upgrade` program to create it (see Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”).

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

13.7.3.2 DROP FUNCTION Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *`function_name`*.

To drop a function, you must have the `DELETE` privilege for the `mysql` database. This is because `DROP FUNCTION` removes a row from the `mysql.func` system table that records the function's name, type, and shared library name.

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

`DROP FUNCTION` is also used to drop stored functions (see Section 13.1.16, “DROP PROCEDURE and DROP FUNCTION Syntax”).

13.7.4 SET Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
      user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | [@@global. | @@session. | @@]system_var_name = expr
```

The `SET` statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed `SET OPTION`, but this syntax is deprecated in favor of `SET` without `OPTION`.

This section describes use of `SET` for assigning values to variables. The `SET` statement can be used to assign values to these types of variables:

- System variables. See Section 5.1.4, “Server System Variables”. System variables also can be set at server startup, as described in Section 5.1.5, “Using System Variables”.

  User-defined variables. See Section 9.4, “User-Defined Variables”.

- Stored procedure and function parameters, and stored program local variables. See Section 13.6.4, “Variables in Stored Programs”.

Some variants of `SET` syntax are used in other contexts:

- `SET CHARACTER SET` and `SET NAMES` assign values to character set and collation variables associated with the connection to the server. `SET ONE_SHOT` is used for replication. These variants are described later in this section.

- `SET PASSWORD` assigns account passwords. See Section 13.7.1.6, “SET PASSWORD Syntax”.

- `SET TRANSACTION ISOLATION LEVEL` sets the isolation level for transaction processing. See Section 13.3.6, “SET TRANSACTION Syntax”.

The following discussion shows the different `SET` syntaxes that you can use to set variables. The examples use the `=` assignment operator, but you can also use the `:=` assignment operator for this purpose.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A user variable is written as `@var_name` and can be set as follows:

```
SET @var_name = expr;
```

Many system variables are dynamic and can be changed while the server runs by using the `SET` statement. For a list, see Section 5.1.5.2, “Dynamic System Variables”. To change a system variable with `SET`, refer to it as `var_name`, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by `GLOBAL` or `@@global.`. The `SUPER` privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by `SESSION`, `@@session.`, or `@@`. Setting a session variable normally requires no special privilege, although there are exceptions (such as `sql_log_bin`.) A client can change its own session variables, but not those of any other client.
- `LOCAL` and `@@local.` are synonyms for `SESSION` and `@@session.`.
- If no modifier is present, `SET` changes the session variable.

A `SET` statement can contain multiple variable assignments, separated by commas. For example, the statement can assign values to a user-defined variable and a system variable. If you set several system variables, the most recent `GLOBAL` or `SESSION` modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The `@@var_name` syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the `SET GLOBAL` statement).

To prevent incorrect usage, MySQL produces an error if you use `SET GLOBAL` with a variable that can only be used with `SET SESSION` or if you do not specify `GLOBAL` (or `@@global.`) when setting a global variable.

To set a `SESSION` variable to the `GLOBAL` value or a `GLOBAL` value to the compiled-in MySQL default value, use the `DEFAULT` keyword. For example, the following two statements are identical in setting the session value of `max_join_size` to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to `DEFAULT`. In such cases, use of `DEFAULT` results in an error.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

It is not permitted to assign the value `DEFAULT` to user-defined variables, and not supported for stored procedure or function parameters or stored program local variables. This results in a syntax error for user-defined variables, and the results are undefined for parameters or local variables.

You can refer to the values of specific global or session system variables in expressions by using one of the `@@`-modifiers. For example, you can retrieve values in a `SELECT` statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as *`@@var_name`* (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)

Note

Some variables displayed by `SHOW VARIABLES` may not be available using `SELECT @@var_name` syntax; an `Unknown system variable` occurs. As a workaround in such cases, you can use `SHOW VARIABLES LIKE 'var_name'`.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

To display system variables names and values, use the `SHOW VARIABLES` statement. (See Section 13.7.5.36, “SHOW VARIABLES Syntax”.)

The following list describes `SET` options that have nonstandard syntax (that is, options that are not set with *`name = value`* syntax).

- `CHARACTER SET {`*`charset_name`*` | DEFAULT}`

  This maps all strings from and to the client with the given mapping. You can add new mappings by editing `sql/convert.cc` in the MySQL source distribution. `SET CHARACTER SET` sets three session system variables: `character_set_client` and `character_set_results` are set to the given character set, and `character_set_connection` to the value of `character_set_database`. See Section 10.1.4, “Connection Character Sets and Collations”.

  The default mapping can be restored by using the value `DEFAULT`. The default depends on the server configuration.

  `ucs2` cannot be used as a client character set, which means that it does not work for `SET CHARACTER SET`.

- `NAMES {'`*`charset_name`*`' [COLLATE '`*`collation_name`*`'] | DEFAULT}`

  `SET NAMES` sets the three session system variables `character_set_client`, `character_set_connection`, and `character_set_results` to the given character set. Setting

This documentation is for an older version. If you're

`character_set_connection` to `charset_name` also sets `collation_connection` to the default collation for `charset_name`. The optional `COLLATE` clause may be used to specify a collation explicitly. See Section 10.1.4, “Connection Character Sets and Collations”.

The default mapping can be restored by using a value of `DEFAULT`. The default depends on the server configuration.

`ucs2` cannot be used as a client character set, which means that it does not work for `SET NAMES`.

- `ONE_SHOT`

This option is a modifier, not a variable. It is *only* for internal use for replication: `mysqlbinlog` uses `SET ONE_SHOT` to modify temporarily the values of character set, collation, and time zone variables to reflect at rollforward what they were originally. `ONE_SHOT` is for internal use only and is deprecated for MySQL 5.0 and up.

`ONE_SHOT` is intended for use only with the permitted set of variables. With other variables, an error occurs:

```
mysql> SET ONE_SHOT max_allowed_packet = 1;
ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for purposes
internal to the MySQL server
```

If `ONE_SHOT` is used with the permitted variables, it changes the variables as requested, but only for the next non-`SET` statement. After that, the server resets all character set, collation, and time zone-related system variables to their previous values. Example:

```
mysql> SET ONE_SHOT character_set_connection = latin5;

mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;

mysql> SHOW VARIABLES LIKE '%_connection';
+--------------------------+-------------------+
| Variable_name            | Value             |
+--------------------------+-------------------+
| character_set_connection | latin5            |
| collation_connection     | latin5_turkish_ci |
+--------------------------+-------------------+

mysql> SHOW VARIABLES LIKE '%_connection';
+--------------------------+-------------------+
| Variable_name            | Value             |
+--------------------------+-------------------+
| character_set_connection | latin1            |
| collation_connection     | latin1_swedish_ci |
+--------------------------+-------------------+
```

13.7.5 SHOW Syntax

`SHOW` has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE FUNCTION func_name
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {LOGS | STATUS }
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW [BDB] LOGS
SHOW MASTER STATUS
SHOW MUTEX STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where:
    LIKE 'pattern'
  | WHERE expr
```

If the syntax for a given `SHOW` statement includes a `LIKE 'pattern'` part, `'pattern'` is a string that can contain the SQL "`%`" and "`_`" wildcard characters. The pattern is useful for restricting statement output to matching values.

Several `SHOW` statements also accept a `WHERE` clause that provides more flexibility in specifying which rows to display. See Section 19.18, "Extensions to SHOW Statements".

Many MySQL APIs (such as PHP) enable you to treat the result returned from a `SHOW` statement as you would a result set from a `SELECT`; see Chapter 20, *Connectors and APIs*, or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the `INFORMATION_SCHEMA` database, which you cannot easily do with results from `SHOW` statements. See Chapter 19, *INFORMATION_SCHEMA Tables*.

13.7.5.1 SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in Section 13.4.1.1, "PURGE BINARY LOGS Syntax", that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| binlog.000016 |    733481 |
+---------------+-----------+
```

`SHOW MASTER LOGS` is equivalent to `SHOW BINARY LOGS`. The `File_size` column is displayed as of MySQL 5.0.7.

You must have the `SUPER` privilege to use this statement.

13.7.5.2 SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify `'log_name'`, the first binary log is displayed.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.8, “SELECT Syntax”.

Note

Issuing a `SHOW BINLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to `SHOW BINLOG EVENTS`, use the `mysqlbinlog` utility to save the binary log to a text file for later examination and analysis. See Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW BINLOG EVENTS`. To get complete coverage of events within a binary log, use `mysqlbinlog`.

13.7.5.3 SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW CHARACTER SET` statement shows all available character sets. The `LIKE` clause, if present, indicates which character set names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The `Maxlen` column shows the maximum number of bytes required to store one character.

13.7.5.4 SHOW COLLATION Syntax

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SHOW COLLATION
    [LIKE 'pattern' | WHERE expr]
```

This statement lists collations supported by the server. By default, the output from `SHOW COLLATION` includes all available collations. The `LIKE` clause, if present, indicates which collation names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

The `Collation` and `Charset` columns indicate the names of the collation and the character set with which it is associated. `Id` is the collation ID. `Default` indicates whether the collation is the default for its character set. `Compiled` indicates whether the character set is compiled into the server. `Sortlen` is related to the amount of memory required to sort strings expressed in the character set.

To see the default collation for each character set, use the following statement. `Default` is a reserved word, so to use it as an identifier, it must be quoted as such:

```
mysql> SHOW COLLATION WHERE `Default` = 'Yes';
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| big5_chinese_ci     | big5     |  1 | Yes     | Yes      |       1 |
| dec8_swedish_ci     | dec8     |  3 | Yes     | Yes      |       1 |
| cp850_general_ci    | cp850    |  4 | Yes     | Yes      |       1 |
| hp8_english_ci      | hp8      |  6 | Yes     | Yes      |       1 |
| koi8r_general_ci    | koi8r    |  7 | Yes     | Yes      |       1 |
| latin1_swedish_ci   | latin1   |  8 | Yes     | Yes      |       1 |
...
```

13.7.5.5 SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS {FROM | IN} tbl_name [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views as of MySQL 5.0.1. The `LIKE` clause, if present, indicates which column names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

```
mysql> SHOW COLUMNS FROM City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

If the data types differ from what you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in Section 13.1.10.4, “Silent Column Specification Changes”.

The `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

You can use *`db_name.tbl_name`* as an alternative to the *`tbl_name`* `FROM` *`db_name`* syntax. In other words, these two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

`SHOW COLUMNS` displays the following values for each table column:

`Field` indicates the column name.

`Type` indicates the column data type.

`Collation` indicates the collation for nonbinary string columns, or `NULL` for other columns. This value is displayed only if you use the `FULL` keyword.

The `Null` field contains `YES` if `NULL` values can be stored in the column. If not, the column contains `NO` as of MySQL 5.0.3, and `''` before that.

The `Key` field indicates whether the column is indexed:

- If `Key` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
- If `Key` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
- If `Key` is `UNI`, the column is the first column of a unique-valued index that cannot contain `NULL` values.
- If `Key` is `MUL`, multiple occurrences of a given value are permitted within the column. The column is the first column of a nonunique index or a unique-valued index that can contain `NULL` values.

If more than one of the `Key` values applies to a given column of a table, `Key` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

Before MySQL 5.0.11, if the column permits `NULL` values, the `Key` value can be `MUL` even when a single-column `UNIQUE` index is used. The rationale was that multiple rows in a `UNIQUE` index can hold a `NULL` value if the column is not declared `NOT NULL`. As of MySQL 5.0.11, the display is `UNI` rather than `MUL` regardless of whether the column permits `NULL`; you can see from the `Null` field whether or not the column can contain `NULL`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `Default` field indicates the default value that is assigned to the column. This is `NULL` if the column has an explicit default of `NULL`. As of MySQL 5.0.50, `Default` is also `NULL` if the column definition has no `DEFAULT` clause.

The `Extra` field contains any additional information that is available about a given column. The value is `auto_increment` for columns that have the `AUTO_INCREMENT` attribute and empty otherwise.

`Privileges` indicates the privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

`Comment` indicates any comment the column has. This value is displayed only if you use the `FULL` keyword.

`SHOW FIELDS` is a synonym for `SHOW COLUMNS`. You can also list a table's columns with the `mysqlshow` *`db_name tbl_name`* command.

The `DESCRIBE` statement provides information similar to `SHOW COLUMNS`. See Section 13.8.1, “DESCRIBE Syntax”.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See Section 13.7.5, “SHOW Syntax”.

13.7.5.6 SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

Shows the `CREATE DATABASE` statement that creates the named database. If the `SHOW` statement includes an `IF NOT EXISTS` clause, the output too includes such a clause. `SHOW CREATE SCHEMA` is a synonym for `SHOW CREATE DATABASE` as of MySQL 5.0.2.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

`SHOW CREATE DATABASE` quotes table and column names according to the value of the `sql_quote_show_create` option. See Section 5.1.4, “Server System Variables”.

13.7.5.7 SHOW CREATE FUNCTION Syntax

```
SHOW CREATE FUNCTION func_name
```

This statement is similar to `SHOW CREATE PROCEDURE` but for stored functions. See Section 13.7.5.8, “SHOW CREATE PROCEDURE Syntax”.

13.7.5.8 SHOW CREATE PROCEDURE Syntax

```
SHOW CREATE PROCEDURE proc_name
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This statement is a MySQL extension. It returns the exact string that can be used to re-create the named stored procedure. A similar statement, `SHOW CREATE FUNCTION`, displays information about stored functions (see Section 13.7.5.7, “SHOW CREATE FUNCTION Syntax”).

To use either statement, you must be the user named in the routine `DEFINER` clause or have `SELECT` access to the `mysql.proc` table. If you do not have privileges for the routine itself, the value displayed for the `Create Procedure` or `Create Function` field will be `NULL`.

```
mysql> SHOW CREATE PROCEDURE test.simpleproc\G
*************************** 1. row ***************************
       Procedure: simpleproc
        sql_mode:
Create Procedure: CREATE PROCEDURE `simpleproc`(OUT param1 INT)
                  BEGIN
                  SELECT COUNT(*) INTO param1 FROM t;
                  END

mysql> SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
       Function: hello
       sql_mode:
Create Function: CREATE FUNCTION `hello`(s CHAR(20))
                 RETURNS CHAR(50)
                 RETURN CONCAT('Hello, ',s,'!')
```

13.7.5.9 SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the `CREATE TABLE` statement that creates the named table. To use this statement, you must have some privilege for the table. As of MySQL 5.0.1, this statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM
```

`SHOW CREATE TABLE` quotes table and column names according to the value of the `sql_quote_show_create` option. See Section 5.1.4, “Server System Variables”.

13.7.5.10 SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows the `CREATE VIEW` statement that creates the named view.

```
mysql> SHOW CREATE VIEW v;
+------+----------------------------------------------------+
| View | Create View                                        |
+------+----------------------------------------------------+
| v    | CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b` |
+------+----------------------------------------------------+
```

This statement was added in MySQL 5.0.1.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Prior to MySQL 5.0.11, the output columns from this statement were shown as `Table` and `Create Table`.

Use of `SHOW CREATE VIEW` requires the `SHOW VIEW` privilege and the `SELECT` privilege for the view in question.

You can also obtain information about view objects from `INFORMATION_SCHEMA`, which contains a `VIEWS` table. See Section 19.17, "The INFORMATION_SCHEMA VIEWS Table".

MySQL lets you use different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE VIEW test.v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE VIEW "v" AS select concat('a','b') AS "col1"
...
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

13.7.5.11 SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]
```

`SHOW DATABASES` lists the databases on the MySQL server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES` as of MySQL 5.0.2. The `LIKE` clause, if present, indicates which database names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, "Extensions to SHOW Statements".

You see only those databases for which you have some kind of privilege, unless you have the global `SHOW DATABASES` privilege. You can also get this list using the `mysqlshow` command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the `SHOW DATABASES` privilege.

MySQL implements databases as directories in the data directory, so this statement simply lists directories in that location. However, the output may include names of directories that do not correspond to actual databases.

13.7.5.12 SHOW ENGINE Syntax

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

`SHOW ENGINE` displays log or status information about a storage engine. The statement has these variants:

```
SHOW ENGINE BDB LOGS
SHOW ENGINE INNODB STATUS
SHOW ENGINE NDB STATUS
SHOW ENGINE NDBCLUSTER STATUS
```

With `LOGS`, this statement requires the `FILE` privilege. With `STATUS`, it requires the `PROCESS` privilege.

`SHOW ENGINE BDB LOGS` displays status information about existing `BDB` log files. It returns the following fields:

- `File`

  The full path to the log file.

- `Type`

  The log file type (`BDB` for Berkeley DB log files).

- `Status`

  The status of the log file (`FREE` if the file can be removed, or `IN USE` if the file is needed by the transaction subsystem)

`SHOW ENGINE INNODB STATUS` displays extensive information from the standard `InnoDB` Monitor about the state of the `InnoDB` storage engine. For information about the standard monitor and other `InnoDB` Monitors that provide information about `InnoDB` processing, see Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”.

Older (and now deprecated) synonyms for `SHOW ENGINE BDB LOGS` and `SHOW ENGINE INNODB STATUS` are `SHOW [BDB] LOGS` and `SHOW INNODB STATUS`, respectively.

If the server has the `NDBCLUSTER` storage engine enabled, `SHOW ENGINE NDB STATUS` can be used to display cluster status information. Sample output from this statement is shown here:

```
mysql> SHOW ENGINE NDB STATUS;
+----------------------+---------+------+--------+
| free_list            | created | free | sizeof |
+----------------------+---------+------+--------+
| NdbTransaction       |       5 |    0 |    208 |
| NdbOperation         |       4 |    4 |    660 |
| NdbIndexScanOperation |       1 |    1 |    736 |
| NdbIndexOperation    |       0 |    0 |   1060 |
| NdbRecAttr           |     645 |  645 |     72 |
| NdbApiSignal         |      16 |   16 |    136 |
| NdbLabel             |       0 |    0 |    196 |
| NdbBranch            |       0 |    0 |     24 |
| NdbSubroutine        |       0 |    0 |     68 |
| NdbCall              |       0 |    0 |     16 |
| NdbBlob              |       2 |    2 |    204 |
| NdbReceiver          |       2 |    0 |     68 |
+----------------------+---------+------+--------+
12 rows in set (0.00 sec)
```

The most useful of the rows from the output of this statement are described in the following list:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `NdbTransaction`: The number and size of `NdbTransaction` objects that have been created. An `NdbTransaction` is created each time a table schema operation (such as `CREATE TABLE` or `ALTER TABLE`) is performed on an `NDB` table.
- `NdbOperation`: The number and size of `NdbOperation` objects that have been created.
- `NdbIndexScanOperation`: The number and size of `NdbIndexScanOperation` objects that have been created.
- `NdbIndexOperation`: The number and size of `NdbIndexOperation` objects that have been created.
- `NdbRecAttr`: The number and size of `NdbRecAttr` objects that have been created. In general, one of these is created each time a data manipulation statement is performed by an SQL node.
- `NdbBlob`: The number and size of `NdbBlob` objects that have been created. An `NdbBlob` is created for each new operation involving a `BLOB` column in an `NDB` table.
- `NdbReceiver`: The number and size of any `NdbReceiver` object that have been created. The number in the `created` column is the same as the number of data nodes in the cluster to which the MySQL server has connected.

Note

`SHOW ENGINE NDB STATUS` returns an empty result if no operations involving `NDB` tables have been performed by the MySQL client accessing the SQL node on which this statement is run.

`SHOW ENGINE NDBCLUSTER STATUS` is a synonym for `SHOW ENGINE NDB STATUS`.

13.7.5.13 SHOW ENGINES Syntax

```
SHOW [STORAGE] ENGINES
```

`SHOW ENGINES` displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. `SHOW TABLE TYPES` is a synonym, but is deprecated and is removed in MySQL 5.5.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 3. row ***************************
 Engine: HEAP
Support: YES
Comment: Alias for MEMORY
*************************** 4. row ***************************
 Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
*************************** 5. row ***************************
 Engine: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
*************************** 6. row ***************************
 Engine: ISAM
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Support: NO
Comment: Obsolete storage engine, now replaced by MyISAM
*************************** 7. row ***************************
 Engine: MRG_ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MERGE
*************************** 8. row ***************************
 Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
*************************** 9. row ***************************
 Engine: INNOBASE
Support: YES
Comment: Alias for INNODB
*************************** 10. row ***************************
 Engine: BDB
Support: YES
Comment: Supports transactions and page-level locking
*************************** 11. row ***************************
 Engine: BERKELEYDB
Support: YES
Comment: Alias for BDB
*************************** 12. row ***************************
 Engine: NDBCLUSTER
Support: NO
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
 Engine: NDB
Support: NO
Comment: Alias for NDBCLUSTER
*************************** 14. row ***************************
 Engine: EXAMPLE
Support: NO
Comment: Example storage engine
*************************** 15. row ***************************
 Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
*************************** 16. row ***************************
 Engine: CSV
Support: NO
Comment: CSV storage engine
*************************** 17. row ***************************
 Engine: FEDERATED
Support: YES
Comment: Federated MySQL storage engine
*************************** 18. row ***************************
 Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappears)
```

The output from `SHOW ENGINES` may vary according to the MySQL version used and other factors. The values shown in the `Support` column indicate the server's level of support for the storage engine, as shown in the following table.

Value	Meaning
`YES`	The engine is supported and is active
`DEFAULT`	Like `YES`, plus this is the default engine
`NO`	The engine is not supported
`DISABLED`	The engine is supported but has been disabled

A value of `NO` means that the server was compiled without support for the engine, so it cannot be activated at runtime.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A value of DISABLED occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log file should contain a reason indicating why the option is disabled. See Section 5.4.1, “The Error Log”.

You might also see DISABLED for a storage engine if the server was compiled to support it, but was started with a --skip-*engine_name* option. For example, --skip-innodb disables the InnoDB engine. For the NDBCLUSTER storage engine, DISABLED means the server was compiled with support for MySQL Cluster, but was not started with the --ndbcluster option.

All MySQL servers support MyISAM tables, because MyISAM is the default storage engine. It is not possible to disable MyISAM.

13.7.5.14 SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

SHOW ERRORS is a diagnostic statement that is similar to SHOW WARNINGS, except that it displays information only for errors, rather than for errors, warnings, and notes.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.8, “SELECT Syntax”.

The SHOW COUNT(*) ERRORS statement displays the number of errors. You can also retrieve this number from the error_count variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

SHOW ERRORS and error_count apply only to errors, not warnings or notes. In other respects, they are similar to SHOW WARNINGS and warning_count. In particular, SHOW ERRORS cannot display information for more than max_error_count messages, and error_count can exceed the value of max_error_count if the number of errors exceeds max_error_count.

For more information, see Section 13.7.5.37, “SHOW WARNINGS Syntax”.

13.7.5.15 SHOW FUNCTION CODE Syntax

```
SHOW FUNCTION CODE func_name
```

This statement is similar to SHOW PROCEDURE CODE but for stored functions. See Section 13.7.5.25, “SHOW PROCEDURE CODE Syntax”. SHOW FUNCTION CODE was added in MySQL 5.0.17.

13.7.5.16 SHOW FUNCTION STATUS Syntax

```
SHOW FUNCTION STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is similar to SHOW PROCEDURE STATUS but for stored functions. See Section 13.7.5.26, “SHOW PROCEDURE STATUS Syntax”.

13.7.5.17 SHOW GRANTS Syntax

```
SHOW GRANTS [FOR user]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This statement displays the GRANT statement or statements that must be issued to duplicate the privileges that are granted to a MySQL user account. SHOW GRANTS requires the SELECT privilege for the mysql database, except to see the privileges for the current user.

To name the account, use the same format as for the GRANT statement; for example, 'jeffrey'@'localhost'. If you specify only the user name part of the account name, a host name part of '%' is used. For additional information about specifying account names, see Section 13.7.1.3, "GRANT Syntax".

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

To display the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

As of MySQL 5.0.24, if SHOW GRANTS FOR CURRENT_USER (or any of the equivalent syntaxes) is used in DEFINER context, such as within a stored procedure that is defined with SQL SECURITY DEFINER), the grants displayed are those of the definer and not the invoker.

SHOW GRANTS displays only the privileges granted explicitly to the named account. Other privileges that might be available to the account are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but SHOW GRANTS will not display them.

13.7.5.18 SHOW INDEX Syntax

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [WHERE expr]
```

SHOW INDEX returns table index information. The format resembles that of the SQLStatistics call in ODBC.

SHOW INDEX returns the following fields:

- Table

  The name of the table.

- Non_unique

  0 if the index cannot contain duplicates, 1 if it can.

- Key_name

  The name of the index. If the index is the primary key, the name is always PRIMARY.

- Seq_in_index

  The column sequence number in the index, starting with 1.

This documentation is for an older version. If you're

- `Column_name`

  The column name.

- `Collation`

  How the column is sorted in the index. In MySQL, this can have values "`A`" (Ascending) or `NULL` (Not sorted).

- `Cardinality`

  An estimate of the number of unique values in the index. This is updated by running `ANALYZE TABLE` or `myisamchk -a`. `Cardinality` is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `Sub_part`

  The index prefix. That is, the number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

  Note

  Prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements is interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  For additional information about index prefixes, see Section 13.1.8, "CREATE INDEX Syntax".

- `Packed`

  Indicates how the key is packed. `NULL` if it is not.

- `Null`

  Contains `YES` if the column may contain `NULL` values and `''` if not.

- `Index_type`

  The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `Comment`

  Information about the index not described in its own column, such as `disabled` if the index is disabled.

You can use *`db_name.tbl_name`* as an alternative to the *`tbl_name FROM db_name`* syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, "Extensions to SHOW Statements".

You can also list a table's indexes with the `mysqlshow -k db_name tbl_name` command.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

13.7.5.19 SHOW INNODB STATUS Syntax

```
SHOW INNODB STATUS
```

In MySQL 5.0, this is a deprecated synonym for `SHOW ENGINE INNODB STATUS`. See Section 13.7.5.12, “SHOW ENGINE Syntax”. `SHOW INNODB STATUS` is removed in MySQL 5.5.

13.7.5.20 SHOW LOGS Syntax

```
SHOW [BDB] LOGS
```

In MySQL 5.0, this is a deprecated synonym for `SHOW ENGINE BDB LOGS`. See Section 13.7.5.12, “SHOW ENGINE Syntax”.

13.7.5.21 SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

This statement provides status information about the binary log files of the master. It requires either the `SUPER` or `REPLICATION CLIENT` privilege.

Example:

```
mysql> SHOW MASTER STATUS;
+--------------+----------+--------------+------------------+
| File         | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+--------------+----------+--------------+------------------+
```

13.7.5.22 SHOW MUTEX STATUS Syntax

```
SHOW MUTEX STATUS
```

`SHOW MUTEX STATUS` displays `InnoDB` mutex statistics. From MySQL 5.0.3 to 5.0.32, the statement displays the following output fields:

- `Mutex`

  The mutex name. The name indicates the mutex purpose. For example, the `log_sys` mutex is used by the `InnoDB` logging subsystem and indicates how intensive logging activity is. The `buf_pool` mutex protects the `InnoDB` buffer pool.

- `Module`

  The source file where the mutex is implemented.

- `Count` indicates how many times the mutex was requested.

- `Spin_waits` indicates how many times the spinlock had to run.

- `Spin_rounds` indicates the number of spinlock rounds. (`spin_rounds` divided by `spin_waits` provides the average round count.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `OS_waits` indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
- `OS_yields` indicates the number of times that a thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that permitting other threads to run will free the mutex so that it can be locked).
- `OS_waits_time` indicates the amount of time (in ms) spent in operating system waits, if the `timed_mutexes` system variable is 1 (`ON`). If `timed_mutexes` is 0 (`OFF`), timing is disabled, so `OS_waits_time` is 0. `timed_mutexes` is off by default.

From MySQL 5.0.33 on, the statement uses the same output format as that just described, but only if `UNIV_DEBUG` was defined at MySQL compilation time (for example, in `include/univ.i` in the `InnoDB` part of the MySQL source tree). If `UNIV_DEBUG` was not defined, the statement displays the following fields. In the latter case (without `UNIV_DEBUG`), the information on which the statement output is based is insufficient to distinguish regular mutexes and mutexes that protect rw-locks (which permit multiple readers or a single writer). Consequently, the output may appear to contain multiple rows for the same mutex.

- `File`

  The source file where the mutex is implemented.

- `Line`

  The line number in the source file where the mutex is created. This may change depending on your version of MySQL.

- `OS_waits`

  Same as `OS_waits_time`.

Information from this statement can be used to diagnose system problems. For example, large values of `spin_waits` and `spin_rounds` may indicate scalability problems.

`SHOW MUTEX STATUS` was added in MySQL 5.0.3. In MySQL 5.1, `SHOW MUTEX STATUS` is deprecated and `SHOW ENGINE INNODB MUTEX` should be used instead. The latter statement displays similar information but in a somewhat different output format. `SHOW MUTEX STATUS` is removed in MySQL 5.5.

13.7.5.23 SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW OPEN TABLES` lists the non-`TEMPORARY` tables that are currently open in the table cache. See Section 8.4.3.1, “How MySQL Opens and Closes Tables”. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

The `FROM` and `LIKE` clauses may be used as of MySQL 5.0.12. The `LIKE` clause, if present, indicates which table names to match. The `FROM` clause, if present, restricts the tables shown to those present in the *`db_name`* database.

`SHOW OPEN TABLES` output has the following columns:

- `Database`

  The database containing the table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Table`

  The table name.

- `In_use`

  The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using `LOCK TABLE t1 WRITE`, `In_use` will be 1. If another client issues `LOCK TABLE t1 WRITE` while the table remains locked, the client will block waiting for the lock, but the lock request causes `In_use` to be 2. If the count is zero, the table is open but not currently being used. `In_use` is also increased by the `HANDLER ... OPEN` statement and decreased by `HANDLER ... CLOSE`.

- `Name_locked`

  Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

13.7.5.24 SHOW PRIVILEGES Syntax

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` shows the list of system privileges that the MySQL server supports. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
*************************** 2. row ***************************
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
*************************** 3. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
  Context: Databases
  Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
...
```

Privileges belonging to a specific user are displayed by the `SHOW GRANTS` statement. See Section 13.7.5.17, “SHOW GRANTS Syntax”, for more information.

13.7.5.25 SHOW PROCEDURE CODE Syntax

```
SHOW PROCEDURE CODE proc_name
```

This statement is a MySQL extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, `SHOW FUNCTION CODE`, displays information about stored functions (see Section 13.7.5.15, “SHOW FUNCTION CODE Syntax”).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To use either statement, you must be the owner of the routine or have `SELECT` access to the `mysql.proc` table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is `Pos`, which is an ordinal number beginning with 0. The second column is `Instruction`, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
    -> BEGIN
    ->   DECLARE fanta INT DEFAULT 55;
    ->   DROP TABLE t2;
    ->   LOOP
    ->     INSERT INTO t3 VALUES (fanta);
    ->     END LOOP;
    ->   END//
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW PROCEDURE CODE p1//
+-----+----------------------------------------+
| Pos | Instruction                            |
+-----+----------------------------------------+
|   0 | set fanta@0 55                         |
|   1 | stmt 9 "DROP TABLE t2"                 |
|   2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
|   3 | jump 2                                 |
+-----+----------------------------------------+
4 rows in set (0.00 sec)
```

In this example, the nonexecutable `BEGIN` and `END` statements have disappeared, and for the `DECLARE` *`variable_name`* statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word `stmt` followed by a type (9 means `DROP`, 5 means `INSERT`, and so on). The final row contains an instruction `jump 2`, meaning `GOTO instruction #2`.

`SHOW PROCEDURE CODE` was added in MySQL 5.0.17.

13.7.5.26 SHOW PROCEDURE STATUS Syntax

```
SHOW PROCEDURE STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is a MySQL extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW FUNCTION STATUS`, displays information about stored functions (see Section 13.7.5.16, “SHOW FUNCTION STATUS Syntax”).

The `LIKE` clause, if present, indicates which procedure or function names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

```
mysql> SHOW PROCEDURE STATUS LIKE 'sp1'\G
*************************** 1. row ***************************
           Db: test
         Name: sp1
         Type: PROCEDURE
      Definer: testuser@localhost
     Modified: 2004-08-03 15:29:37
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
      Created: 2004-08-03 15:29:37
Security_type: DEFINER
      Comment:
```

You can also get information about stored routines from the `ROUTINES` table in `INFORMATION_SCHEMA`. See Section 19.8, “The INFORMATION_SCHEMA ROUTINES Table”.

13.7.5.27 SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

`SHOW PROCESSLIST` shows you which threads are running. You can also get this information using the `mysqladmin processlist` command. If you have the `PROCESS` privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the `FULL` keyword, only the first 100 characters of each statement are shown in the `Info` field.

This statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the `SUPER` privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the `KILL` statement. See Section 13.7.6.3, “KILL Syntax”.

Here is an example of `SHOW PROCESSLIST` output:

```
mysql> SHOW FULL PROCESSLIST\G
*************************** 1. row ***************************
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
*************************** 2. row ***************************
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
       I/O thread to update it
Info: NULL
*************************** 3. row ***************************
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 4. row ***************************
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 5. row ***************************
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

The columns produced by `SHOW PROCESSLIST` have the following meanings:

- `Id`

  The connection identifier. This is the same type of value returned by the `CONNECTION_ID()` function.

- `User`

  The MySQL user who issued the statement. If this is `system user`, it refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. For `system user`, there is no host specified in the `Host` column.

- `Host`

  The host name of the client issuing the statement (except for `system user` where there is no host). `SHOW PROCESSLIST` reports the host name for TCP/IP connections in *`host_name:client_port`* format to make it easier to determine which client is doing what.

- `db`

  The default database, if one is selected, otherwise `NULL`.

- `Command`

  The type of command the thread is executing. For descriptions for thread commands, see Section 8.14, “Examining Thread Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.6, “Server Status Variables”

- `Time`

  The time in seconds that the thread has been in its current state. For a slave SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. See Section 16.2.1, “Replication Implementation Details”.

- `State`

  An action, event, or state that indicates what the thread is doing. Descriptions for `State` values can be found at Section 8.14, “Examining Thread Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

  For the `SHOW PROCESSLIST` statement, the value of `State` is `NULL`.

- `Info`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The statement the thread is executing, or `NULL` if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `Info` value shows the `SELECT` statement.

13.7.5.28 SHOW PROFILE Syntax

```
SHOW PROFILE [type [, type] ... ]
    [FOR QUERY n]
    [LIMIT row_count [OFFSET offset]]

type:
    ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
```

The `SHOW PROFILE` and `SHOW PROFILES` statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Profiling is controlled by the `profiling` session variable, which has a default value of 0 (`OFF`). Profiling is enabled by setting `profiling` to 1 or `ON`:

```
mysql> SET profiling = 1;
```

`SHOW PROFILES` displays a list of the most recent statements sent to the server. The size of the list is controlled by the `profiling_history_size` session variable, which has a default value of 15. The maximum value is 100. Setting the value to 0 has the practical effect of disabling profiling.

All statements are profiled except `SHOW PROFILE` and `SHOW PROFILES`, so you will find neither of those statements in the profile list. Malformed statements are profiled. For example, `SHOW PROFILING` is an illegal statement, and a syntax error occurs if you try to execute it, but it will show up in the profiling list.

`SHOW PROFILE` displays detailed information about a single statement. Without the `FOR QUERY n` clause, the output pertains to the most recently executed statement. If `FOR QUERY n` is included, `SHOW PROFILE` displays information for statement *`n`*. The values of *`n`* correspond to the `Query_ID` values displayed by `SHOW PROFILES`.

The `LIMIT row_count` clause may be given to limit the output to *`row_count`* rows. If `LIMIT` is given, `OFFSET offset` may be added to begin the output *`offset`* rows into the full set of rows.

By default, `SHOW PROFILE` displays `Status` and `Duration` columns. The `Status` values are like the `State` values displayed by `SHOW PROCESSLIST`, although there might be some minor differences in interpretion for the two statements for some status values (see Section 8.14, “Examining Thread Information”).

Optional *`type`* values may be specified to display specific additional types of information:

- `ALL` displays all information
- `BLOCK IO` displays counts for block input and output operations

This documentation is for an older version. If you're

- `CONTEXT SWITCHES` displays counts for voluntary and involuntary context switches
- `CPU` displays user and system CPU usage times
- `IPC` displays counts for messages sent and received
- `MEMORY` is not currently implemented
- `PAGE FAULTS` displays counts for major and minor page faults
- `SOURCE` displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- `SWAPS` displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

```
mysql> SELECT @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)

mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE T1 (id INT);
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROFILES;
+----------+----------+--------------------------+
| Query_ID | Duration | Query                    |
+----------+----------+--------------------------+
|        0 | 0.000088 | SET PROFILING = 1        |
|        1 | 0.000136 | DROP TABLE IF EXISTS t1  |
|        2 | 0.011947 | CREATE TABLE t1 (id INT) |
+----------+----------+--------------------------+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| checking permissions | 0.000040 |
| creating table       | 0.000056 |
| After create         | 0.011363 |
| query end            | 0.000375 |
| freeing items        | 0.000089 |
| logging slow query   | 0.000019 |
| cleaning up          | 0.000005 |
+----------------------+----------+
7 rows in set (0.00 sec)

mysql> SHOW PROFILE FOR QUERY 1;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| query end          | 0.000107 |
| freeing items      | 0.000008 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| logging slow query | 0.000015 |
| cleaning up        | 0.000006 |
+--------------------+----------+
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| checking permissions | 0.000040 | 0.000038 |   0.000002 |
| creating table       | 0.000056 | 0.000028 |   0.000028 |
| After create         | 0.011363 | 0.000217 |   0.001571 |
| query end            | 0.000375 | 0.000013 |   0.000028 |
| freeing items        | 0.000089 | 0.000010 |   0.000014 |
| logging slow query   | 0.000019 | 0.000009 |   0.000010 |
| cleaning up          | 0.000005 | 0.000003 |   0.000002 |
+----------------------+----------+----------+------------+
7 rows in set (0.00 sec)
```

Note

Profiling is only partially functional on some architectures. For values that depend on the `getrusage()` system call, `NULL` is returned on systems such as Windows that do not support the call. In addition, profiling is per process and not per thread. This means that activity on threads within the server other than your own may affect the timing information that you see.

`SHOW PROFILES` and `SHOW PROFILE` were added in MySQL 5.0.37.

You can also get profiling information from the `PROFILING` table in `INFORMATION_SCHEMA`. See Section 19.7, “The INFORMATION_SCHEMA PROFILING Table”. For example, the following queries produce the same result:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

Important

Please note that the `SHOW PROFILE` and `SHOW PROFILES` functionality is part of the MySQL 5.0 Community Server only.

13.7.5.29 SHOW PROFILES Syntax

```
SHOW PROFILES
```

The `SHOW PROFILES` statement, together with `SHOW PROFILE`, displays profiling information that indicates resource usage for statements executed during the course of the current session. For more information, see Section 13.7.5.28, “SHOW PROFILE Syntax”.

13.7.5.30 SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master. Only slaves started with the `--report-host=`*`host_name`* option are visible in this list.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`SHOW SLAVE HOSTS` should be executed on a server that acts as a replication master. The statement displays information about servers that are or have been connected as replication slaves, with each row of the result corresponding to one slave server, as shown here:

```
mysql> SHOW SLAVE HOSTS;
+------------+-----------+------+-----------+
| Server_id  | Host      | Port | Master_id |
+------------+-----------+------+-----------+
|  192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena    | 3306 | 192168011 |
+------------+-----------+------+-----------+
```

- `Server_id`: The unique server ID of the slave server, as configured in the slave server's option file, or on the command line with `--server-id=value`.
- `Host`: The host name of the slave server as specified on the slave with the `--report-host` option. This can differ from the machine name as configured in the operating system.
- `User`: The slave server user name as, specified on the slave with the `--report-user` option. Statement output includes this column only if the master server is started with the `--show-slave-auth-info` option.
- `Password`: The slave server password as, specified on the slave with the `--report-password` option. Statement output includes this column only if the master server is started with the `--show-slave-auth-info` option.
- `Port`: The port on the master to which the slave server is listening, as specified on the slave with the `--report-port` option.
- `Master_id`: The unique server ID of the master server that the slave server is replicating from. This is the server ID of the server on which `SHOW SLAVE HOSTS` is executed, so this same value is listed for each row in the result.

Some MySQL versions report another variable, `Rpl_recovery_rank`. This variable was never used, and was eventually removed. (Bug #13963)

13.7.5.31 SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS
```

This statement provides status information on essential parameters of the slave threads. It requires either the `SUPER` or `REPLICATION CLIENT` privilege.

If you issue this statement using the `mysql` client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: localhost
                Master_User: root
                Master_Port: 3306
              Connect_Retry: 3
            Master_Log_File: gbichot-bin.005
        Read_Master_Log_Pos: 79
             Relay_Log_File: gbichot-relay-bin.005
              Relay_Log_Pos: 548
```


This documentation is for an older version. If you're

```
      Relay_Master_Log_File: gbichot-bin.005
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB:
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 79
            Relay_Log_Space: 552
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 8
```

The following list describes the fields returned by `SHOW SLAVE STATUS`. For additional information about interpreting their meanings, see Section 16.1.3.1, “Checking Replication Status”.

- `Slave_IO_State`

  A copy of the `State` field of the `SHOW PROCESSLIST` output for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. For a listing of possible states, see Section 8.14.6, “Replication Slave I/O Thread States”. For versions of MySQL prior to 5.0.12, it is necessary to check this field for connection problems. In those versions, the thread could be running while unsuccessfully trying to connect to the master; only this field makes you aware of the problem. The state of the SQL thread is not copied because it is simpler. If it is running, there is no problem; if it is not, you can find the error in the `Last_Error` field (described later).

- `Master_Host`

  The master host that the slave is connected to.

- `Master_User`

  The user name of the account used to connect to the master.

- `Master_Port`

  The port used to connect to the master.

- `Connect_Retry`

  The number of seconds between connect retries (default 60). This can be set with the `CHANGE MASTER TO` statement or `--master-connect-retry` option.

- `Master_Log_File`

  The name of the master binary log file from which the I/O thread is currently reading.

- `Read_Master_Log_Pos`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The position in the current master binary log file up to which the I/O thread has read.

- `Relay_Log_File`

  The name of the relay log file from which the SQL thread is currently reading and executing.

- `Relay_Log_Pos`

  The position in the current relay log file up to which the SQL thread has read and executed.

- `Relay_Master_Log_File`

  The name of the master binary log file containing the most recent event executed by the SQL thread.

- `Slave_IO_Running`

  Whether the I/O thread is started and has connected successfully to the master. Internally, the state of this thread is represented by one of the following three values:

  - **MYSQL_SLAVE_NOT_RUN.** The slave I/O thread is not running. For this state, `Slave_IO_Running` is `No`.
  - **MYSQL_SLAVE_RUN_NOT_CONNECT.** The slave I/O thread is running, but is not connected to a replication master. For this state, `Slave_IO_Running` depends on the server version as shown in the following table.

MySQL Version	`Slave_IO_Running`
4.1 (4.1.13 and earlier); 5.0 (5.0.11 and earlier)	`Yes`
4.1 (4.1.14 and later); 5.0 (5.0.12 and later)	`No`
5.1	`No`
5.5	`Connecting`

  - **MYSQL_SLAVE_RUN_CONNECT.** The slave I/O thread is running, and is connected to a replication master. For this state, `Slave_IO_Running` is `Yes`.

- `Slave_SQL_Running`

  Whether the SQL thread is started.

- `Replicate_Do_DB`, `Replicate_Ignore_DB`

  The lists of databases that were specified with the `--replicate-do-db` and `--replicate-ignore-db` options, if any.

- `Replicate_Do_Table`, `Replicate_Ignore_Table`, `Replicate_Wild_Do_Table`, `Replicate_Wild_Ignore_Table`

  The lists of tables that were specified with the `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, and `--replicate-wild-ignore-table` options, if any.

- `Last_Errno`, `Last_Error`

  The error number and error message returned by the most recently executed statement. An error number of 0 and message of the empty string mean “no error.” If the `Last_Error` value is not empty, it also appears as a message in the slave's error log. For example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Last_Errno: 1051
Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

The message indicates that the table `z` existed on the master and was dropped there, but it did not exist on the slave, so `DROP TABLE` failed on the slave. (This might occur, for example, if you forget to copy the table to the slave when setting up replication.)

Note

When the slave SQL thread receives an error, it reports the error first, then stops the SQL thread. This means that there is a small window of time during which `SHOW SLAVE STATUS` shows a nonzero value for `Last_Errno` even though `Slave_SQL_Running` still displays `Yes`.

- `Skip_Counter`

  The current value of the `sql_slave_skip_counter` system variable. See Section 13.4.2.6, “SET GLOBAL sql_slave_skip_counter Syntax”.

- `Exec_Master_Log_Pos`

  The position in the current master binary log file to which the SQL thread has read and executed, marking the start of the next transaction or event to be processed. You can use this value with the `CHANGE MASTER TO` statement's `MASTER_LOG_POS` option when starting a new slave from an existing slave, so that the new slave reads from this point. The coordinates given by (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log correspond to the coordinates given by (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

- `Relay_Log_Space`

  The total combined size of all existing relay log files.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

  The values specified in the `UNTIL` clause of the `START SLAVE` statement.

  `Until_Condition` has these values:

  - `None` if no `UNTIL` clause was specified
  - `Master` if the slave is reading until a given position in the master's binary log
  - `Relay` if the slave is reading until a given position in its relay log

  `Until_Log_File` and `Until_Log_Pos` indicate the log file name and position that define the coordinates at which the SQL thread stops executing.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_Key`

  These fields show the SSL parameters used by the slave to connect to the master, if any.

  `Master_SSL_Allowed` has these values:

  - `Yes` if an SSL connection to the master is permitted
  - `No` if an SSL connection to the master is not permitted

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `Ignored` if an SSL connection is permitted but the slave server does not have SSL support enabled

The values of the other SSL-related fields correspond to the values of the `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, and `MASTER_SSL_KEY` options to the `CHANGE MASTER TO` statement. See Section 13.4.2.1, "CHANGE MASTER TO Syntax".

- `Seconds_Behind_Master`

  This field is an indication of how "late" the slave is:

  - When the slave is actively processing updates, this field shows the difference between the current timestamp on the slave and the original timestamp logged on the master for the event currently being processed on the slave.
  - When no event is currently being processed on the slave, this value is 0.

  In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread. If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

  This time difference computation works even if the master and slave do not have identical clock times, provided that the difference, computed when the slave I/O thread starts, remains constant from then on. Any changes—including NTP updates—can lead to clock skews that can make calculation of `Seconds_Behind_Master` less reliable.

  This field is `NULL` (undefined or unknown) if the slave SQL thread is not running, or if the slave I/O thread is not running or is not connected to the master. For example, if the slave I/O thread is running but is not connected to the master and is sleeping for the number of seconds given by the `CHANGE MASTER TO` statement or `--master-connect-retry` option (default 60) before reconnecting, the value is `NULL`. This is because the slave cannot know what the master is doing, and so cannot say reliably how late it is.

  The value of `Seconds_Behind_Master` is based on the timestamps stored in events, which are preserved through replication. This means that if a master M1 is itself a slave of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the problem for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, the `Seconds_Behind_Master` value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

13.7.5.32 SHOW STATUS Syntax

```
SHOW [GLOBAL | SESSION] STATUS
    [LIKE 'pattern' | WHERE expr]
```

`SHOW STATUS` provides server status information (see Section 5.1.6, "Server Status Variables"). This statement does not require any privilege. It requires only the ability to connect to the server.

Status variable information is also available from the `mysqladmin extended-status` command (see Section 4.5.2, "`mysqladmin` — Client for Administering a MySQL Server").

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For SHOW STATUS, a LIKE clause, if present, indicates which variable names to match. A WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

As of MySQL 5.0.2, SHOW STATUS accepts an optional GLOBAL or SESSION variable scope modifier:

- With a GLOBAL modifier, the statement displays the global status values. A global status variable may represent status for some aspect of the server itself (for example, Aborted_connects), or the aggregated status over all connections to MySQL (for example, Bytes_received and Bytes_sent). If a variable has no global value, the session value is displayed.
- With a SESSION modifier, the statement displays the system varaible values that are in effect for the current connection. If a variable has no session value, the global value is displayed. LOCAL is a synonym for SESSION.
- If no modifier is present, the default is SESSION.

The scope for each status variable is listed at Section 5.1.6, “Server Status Variables”.

Note

Before MySQL 5.0.2, SHOW STATUS returned global status values. Because the default as of 5.0.2 is to return session values, this is incompatible with previous versions. To issue a SHOW STATUS statement that will retrieve global status values for all versions of MySQL, write it like this:

```
SHOW /*!50002 GLOBAL */ STATUS;
```

Partial output is shown here. The list of names and values may be different for your server. The meaning of each variable is given in Section 5.1.6, “Server Status Variables”.

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
| Created_tmp_disk_tables  | 0          |
| Created_tmp_tables       | 8340       |
| Created_tmp_files        | 60         |
...
| Open_tables              | 1          |
| Open_files               | 2          |
| Open_streams             | 0          |
| Opened_tables            | 44600      |
| Questions                | 2026873    |
...
| Table_locks_immediate    | 1920382    |
| Table_locks_waited       | 0          |
| Threads_cached           | 0          |
| Threads_created          | 30022      |
| Threads_connected        | 1          |
| Threads_running          | 1          |
| Uptime                   | 80380      |
+--------------------------+------------+
```

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern:

This documentation is for an older version. If you're

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

13.7.5.33 SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW TABLE STATUS` works likes `SHOW TABLES`, but provides a lot of information about each non-`TEMPORARY` table. You can also get this list using the `mysqlshow --status db_name` command. The `LIKE` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

As of MySQL 5.0.1, this statement also displays information about views.

`SHOW TABLE STATUS` output has the following columns:

- `Name`

  The name of the table.

- `Engine`

  The storage engine for the table. See Chapter 14, *Storage Engines*.

- `Version`

  The version number of the table's `.frm` file.

- `Row_format`

  The row-storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). For `MyISAM` tables, (`Dynamic` corresponds to what `myisamchk -dvv` reports as `Packed`. Starting with MySQL/InnoDB 5.0.3, the format of `InnoDB` tables is reported as `Redundant` or `Compact`. Prior to 5.0.3, `InnoDB` tables are always in the `Redundant` format.

- `Rows`

  The number of rows. Some storage engines, such as `MyISAM`, store the exact count. For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

  The `Rows` value is `NULL` for tables in the `INFORMATION_SCHEMA` database.

- `Avg_row_length`

  The average row length.

- `Data_length`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The length of the data file.

- `Max_data_length`

  The maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

- `Index_length`

  The length of the index file.

- `Data_free`

  The number of allocated but unused bytes.

- `Auto_increment`

  The next `AUTO_INCREMENT` value.

- `Create_time`

  When the table was created.

- `Update_time`

  When the data file was last updated. For some storage engines, this value is `NULL`. For example, `InnoDB` stores multiple tables in its tablespace and the data file timestamp does not apply. For `MyISAM`, the data file timestamp is used; however, on Windows the timestamp is not updated by updates so the value is inaccurate.

- `Check_time`

  When the table was last checked. Not all storage engines update this time, in which case the value is always `NULL`.

- `Collation`

  The table's character set and collation.

- `Checksum`

  The live checksum value (if any).

- `Create_options`

  Extra options used with `CREATE TABLE`. The original options supplied when `CREATE TABLE` is called are retained and the options reported here may differ from the active table settings and options.

- `Comment`

  The comment used when creating the table (or information as to why MySQL could not access the table information).

In the table comment, `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of completely free 1MB extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For MEMORY tables, the Data_length, Max_data_length, and Index_length values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

Beginning with MySQL 5.0.3, for NDBCLUSTER tables, the output of this statement shows appropriate values for the Avg_row_length and Data_length columns, with the exception that BLOB columns are not taken into account. In addition, the number of replicas is now shown in the Comment column (as number_of_replicas).

For views, all the fields displayed by SHOW TABLE STATUS are NULL except that Name indicates the view name and Comment says view.

13.7.5.34 SHOW TABLES Syntax

```
SHOW [FULL] TABLES [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TABLES lists the non-TEMPORARY tables in a given database. You can also get this list using the mysqlshow *db_name* command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

Matching performed by the LIKE clause is dependent on the setting of the lower_case_table_names system variable.

Before MySQL 5.0.1, the output from SHOW TABLES contains a single column of table names. Beginning with MySQL 5.0.1, this statement also lists any views in the database. As of MySQL 5.0.2, the FULL modifier is supported such that SHOW FULL TABLES displays a second output column. Values for the second column are BASE TABLE for a table and VIEW for a view.

If you have no privileges for a base table or view, it does not show up in the output from SHOW TABLES or mysqlshow db_name.

13.7.5.35 SHOW TRIGGERS Syntax

```
SHOW TRIGGERS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TRIGGERS lists the triggers currently defined for tables in a database (the default database unless a FROM clause is given). This statement returns results only if you have the SUPER privilege. It was implemented in MySQL 5.0.10. The LIKE clause, if present, indicates which table names to match (not trigger names) and causes the statement to display triggers for those tables. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

For the trigger ins_sum as defined in Section 18.3, “Using Triggers”, the output of this statement is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%'\G
*************************** 1. row ***************************
  Trigger: ins_sum
    Event: INSERT
    Table: account
Statement: SET @sum = @sum + NEW.amount
   Timing: BEFORE
  Created: NULL
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
 sql_mode:
  Definer: me@localhost
```

`SHOW TRIGGERS` output has the following columns:

- `Trigger`: The trigger name.
- `Event`: The type of operation that causes trigger activation. The value is `'INSERT'`, `'UPDATE'`, or `'DELETE'`.
- `Table`: The table for which the trigger is defined.
- `Statement`: The trigger body; that is, the statement executed when the trigger activates.
- `Timing`: Whether the trigger activates before or after the triggering event. The value is `'BEFORE'` or `'AFTER'`.
- `Created`: The value of this column is always `NULL`.
- `sql_mode`: The SQL mode in effect when the trigger executes. This column was added in MySQL 5.0.11.
- `Definer`: The account that created the trigger. This column was added in MySQL 5.0.17.

You must have the `SUPER` privilege to execute `SHOW TRIGGERS`.

You can also obtain information about trigger objects from `INFORMATION_SCHEMA`, which contains a `TRIGGERS` table. See Section 19.15, “The INFORMATION_SCHEMA TRIGGERS Table”.

13.7.5.36 SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
    [LIKE 'pattern' | WHERE expr]
```

`SHOW VARIABLES` shows the values of MySQL system variables (see Section 5.1.4, “Server System Variables”). This statement does not require any privilege. It requires only the ability to connect to the server.

System variable information is also available from the `mysqladmin variables` command (see Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”).

For `SHOW VARIABLES`, a `LIKE` clause, if present, indicates which variable names to match. A `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.18, “Extensions to SHOW Statements”.

`SHOW VARIABLES` accepts an optional `GLOBAL` or `SESSION` variable scope modifier:

- With a `GLOBAL` modifier, the statement displays global system variable values. These are the values used to initialize the corresponding session variables for new connections to MySQL. If a variable has no global value, the session value is displayed.
- With a `SESSION` modifier, the statement displays the system varaible values that are in effect for the current connection. If a variable has no session value, the global value is displayed. `LOCAL` is a synonym for `SESSION`.
- If no modifier is present, the default is `SESSION`.

The scope for each system variable is listed at Section 5.1.4, “Server System Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Most system variables can be set at server startup (read-only variables such as `version_comment` are exceptions). Many can be changed at runtime with the `SET` statement. See Section 5.1.5, “Using System Variables”, and Section 13.7.4, “SET Syntax”.

Partial output is shown here. The list of names and values may differ for your server. Section 5.1.4, “Server System Variables”, describes the meaning of each variable, and Section 8.12.2, “Tuning Server Parameters”, provides information about tuning them.

```
mysql> SHOW VARIABLES;
+---------------------------------+------------------------------------+
| Variable_name                   | Value                              |
+---------------------------------+------------------------------------+
| auto_increment_increment        | 1                                  |
| auto_increment_offset           | 1                                  |
| automatic_sp_privileges         | ON                                 |
| back_log                        | 50                                 |
| basedir                         | /                                  |
| bdb_cache_size                  | 8388600                            |
| bdb_home                        | /var/lib/mysql/                    |
| bdb_log_buffer_size             | 32768                              |
...
| max_connections                 | 100                                |
| max_connect_errors              | 10                                 |
| max_delayed_threads             | 20                                 |
| max_error_count                 | 64                                 |
| max_heap_table_size             | 16777216                           |
| max_join_size                   | 4294967295                         |
| max_relay_log_size              | 0                                  |
| max_sort_length                 | 1024                               |
...
| time_zone                       | SYSTEM                             |
| timed_mutexes                   | OFF                                |
| tmp_table_size                  | 33554432                           |
| tmpdir                          |                                    |
| transaction_alloc_block_size    | 8192                               |
| transaction_prealloc_size       | 4096                               |
| tx_isolation                    | REPEATABLE-READ                    |
| updatable_views_with_limit      | YES                                |
| version                         | 5.0.19-Max                         |
| version_comment                 | MySQL Community Edition - Max (GPL) |
| version_compile_machine         | i686                               |
| version_compile_os              | pc-linux-gnu                       |
| wait_timeout                    | 28800                              |
+---------------------------------+------------------------------------+
```

With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the “`%`” wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because “`_`” is a wildcard that matches any single character, you should escape it as “`\_`” to match it literally. In practice, this is rarely necessary.

13.7.5.37 SHOW WARNINGS Syntax

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS` is a diagnostic statement that displays information about the conditions (errors, warnings, and notes) resulting from executing a statement in the current session. Warnings are generated for DML statements such as `INSERT`, `UPDATE`, and `LOAD DATA INFILE` as well as DDL statements such as `CREATE TABLE` and `ALTER TABLE`.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.8, “SELECT Syntax”.

`SHOW WARNINGS` is also used following `EXPLAIN EXTENDED`, to display the extra information generated by `EXPLAIN` when the `EXTENDED` keyword is used. See Section 8.8.3, “EXPLAIN EXTENDED Output Format”.

`SHOW WARNINGS` displays information about the conditions resulting from the most recent statement in the current session that generated messages. It shows nothing if the most recent statement used a table and generated no messages. (That is, statements that use a table but generate no messages clear the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

The `SHOW COUNT(*) WARNINGS` diagnostic statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the `warning_count` system variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

A related diagnostic statement, `SHOW ERRORS`, shows only error conditions (it excludes warnings and notes), and `SHOW COUNT(*) ERRORS` statement displays the total number of errors. See Section 13.7.5.14, “SHOW ERRORS Syntax”.

Here is a simple example that shows data-conversion warnings for `INSERT`:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'), (NULL,'test'), (300,'xyz');
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1048
Message: Column 'a' cannot be null
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value adjusted for column 'a' at row 3
3 rows in set (0.00 sec)
```

The `max_error_count` system variable controls the maximum number of error, warning, and note messages for which the server stores information, and thus the number of messages that `SHOW`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`WARNINGS` displays. To change the number of messages the server can store, change the value of `max_error_count`. The default is 64.

`max_error_count` controls only how many messages are stored, not how many are counted. The value of `warning_count` is not limited by `max_error_count`, even if the number of messages generated exceeds `max_error_count`. The following example demonstrates this. The `ALTER TABLE` statement produces three warning messages (strict SQL mode is disabled for the example to prevent an error from occuring after a single conversion issue). Only one message is stored and displayed because `max_error_count` has been set to 1, but all three are counted (as shown by the value of `warning_count`):

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> SET max_error_count=1, sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)
```

To disable message storage, set `max_error_count` to 0. In this case, `warning_count` still indicates how many warnings occurred, but messages are not stored and cannot be displayed.

The `sql_notes` system variable controls whether note messages increment `warning_count` and whether the server stores them. By default, `sql_notes` is 1, but if set to 0, notes do not increment `warning_count` and the server does not store them:

```
mysql> SET sql_notes = 1;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS;
+-------+------+-------------------------------+
| Level | Code | Message                       |
+-------+------+-------------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-------------------------------+
1 row in set (0.00 sec)

mysql> SET sql_notes = 0;
mysql> DROP TABLE IF EXISTS test.no_such_table;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW WARNINGS;
Empty set (0.00 sec)
```

The MySQL server sends to each client a count indicating the total number of errors, warnings, and notes resulting from the most recent statement executed by that client. From the C API, this value can be obtained by calling `mysql_warning_count()`. See Section 20.6.7.72, “mysql_warning_count()”.

In the `mysql` client, you can enable and disable automatic warnings display using the `warnings` and `nowarning` commands, respectively, or their shortcuts, `\W` and `\w` (see Section 4.5.1.2, “mysql Commands”). For example:

```
mysql> \W
Show warnings enabled.
mysql> SELECT 1/0;
+------+
| 1/0  |
+------+
| NULL |
+------+
1 row in set, 1 warning (0.03 sec)

Warning (Code 1365): Division by 0
mysql> \w
Show warnings disabled.
```

13.7.6 Other Administrative Statements

13.7.6.1 CACHE INDEX Syntax

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It is used only for `MyISAM` tables. After the indexes have been assigned, they can be preloaded into the cache if desired with `LOAD INDEX INTO CACHE`.

The following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The syntax of `CACHE INDEX` enables you to specify that only particular indexes from a table should be assigned to the cache. The current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See Section 5.1.5.1, “Structured System Variables”.

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

13.7.6.2 FLUSH Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL]
    flush_option [, flush_option] ...
```

The `FLUSH` statement has several variant forms that clear or reload various internal caches, flush tables, or acquire locks. To execute `FLUSH`, you must have the `RELOAD` privilege.

By default, the server writes `FLUSH` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

Note

`FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not written to the binary log in any case because they would cause problems if replicated to a slave.

Sending a `SIGHUP` signal to the server causes several flush operations to occur that are similar to various forms of the `FLUSH` statement. See Section 5.1.9, “Server Response to Signals”.

The `FLUSH` statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

The `RESET` statement is similar to `FLUSH`. See Section 13.7.6.5, “RESET Syntax”, for information about using the `RESET` statement with replication.

`flush_option` can be any of the following items:

- `DES_KEY_FILE`

  Reloads the DES keys from the file that was specified with the `--des-key-file` option at server startup time.

- `HOSTS`

  Empties the host cache. You should flush the host cache if some of your hosts change IP address or if the error message `Host 'host_name' is blocked` occurs. (See Section B.5.2.6, “Host 'host_name' is blocked”.) When more than `max_connect_errors` errors occur successively for a given host while

This documentation is for an older version. If you're
This documentation is for an older version. If you're

connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host cache enables further connection attempts from the host. The default value of `max_connect_errors` is 10. To avoid this error message, start the server with `max_connect_errors` set to a large value.

- `LOGS`

  Closes and reopens all log files. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file.

  If you execute `FLUSH LOGS` and `mysqld` is writing the error log to a file (for example, if it was started with the `--log-error` option), log file renaming occurs as described in Section 5.4.1, “The Error Log”.

- `MASTER`

  Deletes all binary logs, resets the binary log index file and creates a new binary log. `FLUSH MASTER` is deprecated in favor of `RESET MASTER`. `FLUSH MASTER` is still accepted in MySQL 5.0 for backward compatibility, but is removed in MySQL 5.6. See Section 13.4.1.2, “RESET MASTER Syntax”.

- `PRIVILEGES`

  Reloads the privileges from the grant tables in the `mysql` database.

  The server caches information in memory as a result of `GRANT` and `CREATE USER` statements. This memory is not released by the corresponding `REVOKE` and `DROP USER` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

- `QUERY CACHE`

  Defragment the query cache to better utilize its memory. `FLUSH QUERY CACHE` does not remove any queries from the cache, unlike `FLUSH TABLES` or `RESET QUERY CACHE`.

- `SLAVE`

  Resets all replication slave parameters, including relay log files and replication position in the master's binary logs. `FLUSH SLAVE` is deprecated in favor of `RESET SLAVE`. `FLUSH SLAVE` is still accepted in MySQL 5.0 for backward compatibility, but is removed in MySQL 5.6. See Section 13.4.2.5, “RESET SLAVE Syntax”.

- `STATUS`

  This option adds the current thread's session status variable values to the global values and resets the session values to zero. Some global variables may be reset to zero as well. It also resets the counters for key caches (default and named) to zero and sets `Max_used_connections` to the current number of open connections. This is something you should use only when debugging a query. See Section 1.7, “How to Report Bugs or Problems”.

- `TABLES`

  `FLUSH TABLES` flushes tables, and, depending on the variant used, acquires locks. The permitted syntax is discussed later in this section.

- `USER_RESOURCES`

  Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. `FLUSH USER_RESOURCES` does not

This documentation is for an older version. If you're
This documentation is for an older version. If you're

apply to the limit on maximum simultaneous connections. See Section 6.3.4, “Setting Account Resource Limits”.

The `mysqladmin` utility provides a command-line interface to some flush operations, using commands such as `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, and `flush-tables`. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

Note

It is not possible to issue `FLUSH` statements within stored functions or triggers. However, you may use `FLUSH` in stored procedures, so long as these are not called from stored functions or triggers. See Section C.1, “Restrictions on Stored Programs”.

FLUSH TABLES Syntax

`FLUSH TABLES` has several forms, described following. `FLUSH TABLE` is a synonym for `FLUSH TABLES`, except that `TABLE` does not work with the `WITH READ LOCK` variant.

- `FLUSH TABLES`

  Closes all open tables, forces all tables in use to be closed, and flushes the query cache. `FLUSH TABLES` also removes all query results from the query cache, like the `RESET QUERY CACHE` statement.

- `FLUSH TABLES` *`tbl_name`* `[,` *`tbl_name`*`] ...`

  With a list of one or more comma-separated table names, this statement is like `FLUSH TABLES` with no names except that the server flushes only the named tables. No error occurs if a named table does not exist.

- `FLUSH TABLES WITH READ LOCK`

  Closes all open tables and locks all tables for all databases with a global read lock. This is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time. Use `UNLOCK TABLES` to release the lock.

  `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits:

  - `UNLOCK TABLES` implicitly commits any active transaction only if any tables currently have been locked with `LOCK TABLES`. The commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table locks.
  - Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

13.7.6.3 KILL Syntax

```
KILL [CONNECTION | QUERY] processlist_id
```

Each connection to `mysqld` runs in a separate thread. You can kill a thread with the `KILL` *`processlist_id`* statement.

Thread processlist identifiers can be determined from the `Id` column of `SHOW PROCESSLIST` output. The value for the current thread is returned by the `CONNECTION_ID()` function.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In MySQL 5.0.0, `KILL` permits an optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given *`processlist_id`*, after terminating any statement the connection is executing.
- `KILL QUERY` terminates the statement the connection is currently executing, but leaves the connection itself intact.

If you have the `PROCESS` privilege, you can see all threads. If you have the `SUPER` privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.

Note

You cannot use `KILL` with the Embedded MySQL Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

- During `SELECT` operations, for `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- During `ALTER TABLE` operations, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.
- During `UPDATE` or `DELETE` operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. If you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` aborts and returns `NULL`.
- An `INSERT DELAYED` thread quickly flushes (inserts) all rows it has in memory and then terminates.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.

Warning

Killing a `REPAIR TABLE` or `OPTIMIZE TABLE` operation on a `MyISAM` table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

13.7.6.4 LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `LOAD INDEX INTO CACHE` statement preloads a table index into the key cache to which it has been assigned by an explicit `CACHE INDEX` statement, or into the default key cache otherwise. `LOAD INDEX INTO CACHE` is used only for `MyISAM` tables.

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

The following statement preloads nodes (index blocks) of indexes for the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

This statement preloads all index blocks from `t1`. It preloads only blocks for the nonleaf nodes from `t2`.

The syntax of `LOAD INDEX INTO CACHE` enables you to specify that only particular indexes from a table should be preloaded. The current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

`LOAD INDEX INTO CACHE ... IGNORE LEAVES` fails unless all indexes in a table have the same block size. (Prior to MySQL 5.0.87, it fails even without `IGNORE LEAVES`.) You can determine index block sizes for a table by using `myisamchk -dv` and checking the `Blocksize` column.

13.7.6.5 RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD` privilege to execute `RESET`.

`RESET` acts as a stronger version of the `FLUSH` statement. See Section 13.7.6.2, “FLUSH Syntax”.

The `RESET` statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`reset_option` can be any of the following:

- `MASTER`

  Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

- `QUERY CACHE`

  Removes all query results from the query cache.

- `SLAVE`

  Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one.

13.8 MySQL Utility Statements

13.8.1 DESCRIBE Syntax

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The DESCRIBE and EXPLAIN statements are synonyms, used either to obtain information about table structure or query execution plans. For more information, see Section 13.8.2, “EXPLAIN Syntax”.

13.8.2 EXPLAIN Syntax

```
{EXPLAIN | DESCRIBE | DESC}
    tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
    [EXTENDED] SELECT select_options
```

The DESCRIBE and EXPLAIN statements are synonyms. In practice, the DESCRIBE keyword is more often used to obtain information about table structure, whereas EXPLAIN is used to obtain a query execution plan (that is, an explanation of how MySQL would execute a query). The following discussion uses the DESCRIBE and EXPLAIN keywords in accordance with those uses, but the MySQL parser treats them as completely synonymous.

Obtaining Table Structure Information

DESCRIBE provides information about the columns in a table:

```
mysql> DESCRIBE City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

DESCRIBE is a shortcut for SHOW COLUMNS. As of MySQL 5.0.1, these statements also display information for views. The description for SHOW COLUMNS provides more information about the output columns. See Section 13.7.5.5, “SHOW COLUMNS Syntax”.

By default, DESCRIBE displays information about all columns in the table. *col_name*, if given, is the name of a column in the table. In this case, the statement displays information only for the named column. *wild*, if given, is a pattern string. It can contain the SQL “%” and “_” wildcard characters. In this case, the statement displays output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

The DESCRIBE statement is provided for compatibility with Oracle.

The SHOW CREATE TABLE, SHOW TABLE STATUS, and SHOW INDEX statements also provide information about tables. See Section 13.7.5, “SHOW Syntax”.

Obtaining Execution Plan Information

The EXPLAIN statement provides information about how MySQL executes statements:

- When you precede a SELECT statement with the keyword EXPLAIN, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using EXPLAIN to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `EXPLAIN EXTENDED` can be used to obtain additional execution plan information. See Section 8.8.3, “EXPLAIN EXTENDED Output Format”.

With the help of `EXPLAIN`, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 13.2.8, “SELECT Syntax”.)

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.

13.8.3 HELP Syntax

```
HELP 'search_string'
```

The `HELP` statement returns online information from the MySQL Reference manual. Its proper operation requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.8, “Server-Side Help”).

The `HELP` statement searches the help tables for the given search string and displays the result of the search. The search string is not case sensitive.

The search string can contain the wildcard characters “`%`” and “`_`”. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, `HELP 'rep%'` returns a list of topics that begin with `rep`.

The HELP statement understands several types of search strings:

- At the most general level, use `contents` to retrieve a list of the top-level help categories:

```
HELP 'contents'
```

- For a list of topics in a given help category, such as `Data Types`, use the category name:

```
HELP 'data types'
```

- For help on a specific help topic, such as the `ASCII()` function or the `CREATE TABLE` statement, use the associated keyword or keywords:

```
HELP 'ascii'
HELP 'create table'
```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response `HELP` returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the `mysql` client, but note that `mysql` itself reformats `HELP` result sets in a different way.

- Empty result set

This documentation is for an older version. If you're
This documentation is for an older version. If you're

No match could be found for the search string.

- Result set containing a single row with three columns

  This means that the search string yielded a hit for the help topic. The result has three columns:

  - `name`: The topic name.
  - `description`: Descriptive help text for the topic.
  - `example`: Usage example or examples. This column might be blank.

  Example: `HELP 'replace'`

  Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

  This means that the search string matched many help topics. The result set indicates the help topic names:

  - `name`: The help topic name.
  - `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

  Example: `HELP 'status'`

  Yields:

```
+----------------------+----------------+
| name                 | is_it_category |
+----------------------+----------------+
| SHOW                 | N              |
| SHOW ENGINE          | N              |
| SHOW INNODB STATUS   | N              |
| SHOW MASTER STATUS   | N              |
| SHOW PROCEDURE STATUS | N              |
| SHOW SLAVE STATUS    | N              |
| SHOW STATUS          | N              |
| SHOW TABLE STATUS    | N              |
+----------------------+----------------+
```

- Result set containing multiple rows with three columns

  This means the search string matches a category. The result set contains category entries:

  - `source_category_name`: The help category name.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `name`: The category or topic name
- `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

Example: `HELP 'functions'`

Yields:

```
+----------------------+-------------------------+----------------+
| source_category_name | name                    | is_it_category |
+----------------------+-------------------------+----------------+
| Functions            | CREATE FUNCTION         | N              |
| Functions            | DROP FUNCTION           | N              |
| Functions            | Bit Functions           | Y              |
| Functions            | Comparison operators    | Y              |
| Functions            | Control flow functions  | Y              |
| Functions            | Date and Time Functions | Y              |
| Functions            | Encryption Functions    | Y              |
| Functions            | Information Functions   | Y              |
| Functions            | Logical operators       | Y              |
| Functions            | Miscellaneous Functions | Y              |
| Functions            | Numeric Functions       | Y              |
| Functions            | String Functions        | Y              |
+----------------------+-------------------------+----------------+
```

If you intend to use the `HELP` statement while other tables are locked with `LOCK TABLES`, you must also lock the required `mysql.help_xxx` tables.

13.8.4 USE Syntax

```
USE db_name
```

The `USE db_name` statement tells MySQL to use the *`db_name`* database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another `USE` statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

Making a particular database the default by means of the `USE` statement does not preclude you from accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 14 Storage Engines

Table of Contents

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle nontransaction-safe tables:

- `MyISAM` manages nontransactional tables. It provides high-speed storage and retrieval, as well as fulltext searching capabilities. `MyISAM` is supported in all MySQL configurations, and is the default storage engine unless you have configured MySQL to use a different one by default.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- The `MEMORY` storage engine provides in-memory tables. The `MERGE` storage engine enables a collection of identical `MyISAM` tables to be handled as a single table. Like `MyISAM`, the `MEMORY` and `MERGE` storage engines handle nontransactional tables, and both are also included in MySQL by default.

  Note

  The `MEMORY` storage engine formerly was known as the `HEAP` engine.

- The `InnoDB` and `BDB` storage engines provide transaction-safe tables. To maintain data integrity, `InnoDB` also supports `FOREIGN KEY` referential-integrity constraints.

- The `EXAMPLE` storage engine is a “stub” engine that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

- `NDBCLUSTER` (also known as `NDB`) is the storage engine used by MySQL Cluster to implement tables that are partitioned over many computers. It is available in MySQL 5.0 binary distributions. This storage engine is currently supported on a number of Unix platforms. Experimental support for Windows is available beginning in MySQL Cluster NDB 7.0; however, we do not intend to backport this functionality to MySQL 5.0.

  MySQL Cluster is covered in a separate chapter of this Manual. See Chapter 17, *MySQL Cluster*, for more information.

  Note

  MySQL Cluster users wishing to upgrade from MySQL 5.0 should instead migrate to MySQL Cluster NDB 6.3, 7.0, or 7.1; these are based on MySQL 5.1 but contain the latest improvements and fixes for `NDBCLUSTER`. The `NDBCLUSTER` storage engine is not supported in standard MySQL 5.1 releases.

- The `ARCHIVE` storage engine is used for storing large amounts of data without indexes with a very small footprint.

- The `CSV` storage engine stores data in text files using comma-separated values format.

- The `BLACKHOLE` storage engine accepts but does not store data and retrievals always return an empty set.

- The `FEDERATED` storage engine was added in MySQL 5.0.3. This engine stores data in a remote database. It works with MySQL only, using the MySQL C Client API.

To determine which storage engines your server supports by using the `SHOW ENGINES` statement. The value in the `Support` column indicates whether an engine can be used. A value of `YES`, `NO`, or `DEFAULT` indicates that an engine is available, not available, or available and currently set as the default storage engine.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 3. row ***************************
```


This documentation is for an older version. If you're

```
 Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
*************************** 4. row ***************************
 Engine: BerkeleyDB
Support: NO
Comment: Supports transactions and page-level locking
*************************** 5. row ***************************
 Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappears)
...
```

This chapter describes each of the MySQL storage engines except for `NDBCLUSTER`, which is covered in Chapter 17, *MySQL Cluster*.

For information about storage engine support offered in commercial MySQL Server binaries, see *MySQL Enterprise Server 5.1*, on the MySQL Web site. The storage engines available might depend on which edition of Enterprise Server you are using.

For answers to some commonly asked questions about MySQL storage engines, see Section A.2, “MySQL 5.0 FAQ: Storage Engines”.

When you create a new table, you can specify which storage engine to use by adding an `ENGINE` or `TYPE` table option to the `CREATE TABLE` statement:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
CREATE TABLE t (i INT) TYPE = MEMORY;
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

If you omit the `ENGINE` or `TYPE` option, the default storage engine is used. Normally, this is `MyISAM`, but you can change it by using the `--default-storage-engine` or `--default-table-type` server startup option, or by setting the `default-storage-engine` or `default-table-type` option in the `my.cnf` configuration file.

You can set the default storage engine to be used during the current session by setting the `storage_engine` or `table_type` variable:

```
SET storage_engine=MYISAM;
SET table_type=BDB;
```

When MySQL is installed on Windows using the MySQL Configuration Wizard, the `InnoDB` or `MyISAM` storage engine can be selected as the default. See Section 2.10.3.5, “The Database Usage Dialog”.

To convert a table from one storage engine to another, use an `ALTER TABLE` statement that indicates the new engine:

```
ALTER TABLE t ENGINE = MYISAM;
ALTER TABLE t TYPE = BDB;
```

See Section 13.1.10, “CREATE TABLE Syntax”, and Section 13.1.4, “ALTER TABLE Syntax”.

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine. This behavior is convenient when you want to copy tables between MySQL servers that support different storage engines. (For example, in a replication setup, perhaps your master server supports transactional storage engines for increased safety, but the slave servers use only nontransactional storage engines for greater speed.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This automatic substitution of the default storage engine for unavailable engines can be confusing for new MySQL users. A warning is generated whenever a storage engine is automatically changed.

For new tables, MySQL always creates an `.frm` file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the `.frm` file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage.

A database may contain tables of different types. That is, tables need not all be created with the same storage engine.

Transaction-safe tables (TSTs) have several advantages over nontransaction-safe tables (NTSTs):

- They are safer. Even if MySQL crashes or you get hardware problems, you can get your data back, either by automatic recovery or from a backup plus the transaction log.
- You can combine many statements and accept them all at the same time with the `COMMIT` statement (if autocommit is disabled).
- You can execute `ROLLBACK` to ignore your changes (if autocommit is disabled).
- If an update fails, all of your changes are reverted. (With nontransaction-safe tables, all changes that have taken place are permanent.)
- Transaction-safe storage engines can provide better concurrency for tables that get many updates concurrently with reads.

You can combine transaction-safe and nontransaction-safe tables in the same statements to get the best of both worlds. However, although MySQL supports several transaction-safe storage engines, for best results, you should not mix different storage engines within a transaction with autocommit disabled. For example, if you do this, changes to nontransaction-safe tables still are committed immediately and cannot be rolled back. For information about this and other problems that can occur in transactions that use mixed storage engines, see Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

Nontransaction-safe tables have several advantages of their own, all of which occur because there is no transaction overhead:

- Much faster
- Lower disk space requirements
- Less memory required to perform updates

14.1 The MyISAM Storage Engine

`MyISAM` is the default storage engine. It is based on the older (and no longer available) `ISAM` storage engine but has many useful extensions.

Each `MyISAM` table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format. The data file has an `.MYD` (`MYData`) extension. The index file has an `.MYI` (`MYIndex`) extension.

To specify explicitly that you want a `MyISAM` table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Normally, it is unnecessary to use `ENGINE` to specify the `MyISAM` storage engine. `MyISAM` is the default engine unless the default has been changed. To ensure that `MyISAM` is used in situations where the default might have been changed, include the `ENGINE` option explicitly.

You can check or repair `MyISAM` tables with the `mysqlcheck` client or `myisamchk` utility. You can also compress `MyISAM` tables with `myisampack` to take up much less space. See Section 4.5.3, "`mysqlcheck` — A Table Maintenance Program", Section 4.6.3, "`myisamchk` — MyISAM Table-Maintenance Utility", and Section 4.6.5, "`myisampack` — Generate Compressed, Read-Only MyISAM Tables".

`MyISAM` tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

  There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to permit better index compression.

- Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.

- There is a limit of 2^{32} (~4.295E+09) rows in a `MyISAM` table. If you build MySQL with the `--with-big-tables` option, the row limitation is increased to $(2^{32})^2$ (1.844E+19) rows. See Section 2.17.3, "MySQL Source-Configuration Options". Binary distributions for Unix and Linux are built with this option.

- The maximum number of indexes per `MyISAM` table is 64. This can be changed by recompiling. Beginning with MySQL 5.0.18, you can configure the build by invoking `configure` with the `--with-max-indexes=`*`N`* option, where *`N`* is the maximum number of indexes to permit per `MyISAM` table. *`N`* must be less than or equal to 128. Before MySQL 5.0.18, you must change the source.

  The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.

- When rows are inserted in sorted order (as when you are using an `AUTO_INCREMENT` column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.

- Internal handling of one `AUTO_INCREMENT` column per table is supported. `MyISAM` automatically updates this column for `INSERT` and `UPDATE` operations. This makes `AUTO_INCREMENT` columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted. (When an `AUTO_INCREMENT` column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The `AUTO_INCREMENT` value can be reset with `ALTER TABLE` or `myisamchk`.

- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.

- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. A free block

This documentation is for an older version. If you're
This documentation is for an older version. If you're

can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See Section 8.11.3, “Concurrent Inserts”.

- You can put the data file and index file in different directories on different physical devices to get more speed with the `DATA DIRECTORY` and `INDEX DIRECTORY` table options to `CREATE TABLE`. See Section 13.1.10, “CREATE TABLE Syntax”.
- `BLOB` and `TEXT` columns can be indexed.
- `NULL` values are permitted in indexed columns. This takes 0 to 1 bytes per key.
- Each character column can have a different character set. See Section 10.1, “Character Set Support”.
- There is a flag in the `MyISAM` index file that indicates whether the table was closed correctly. If `mysqld` is started with the `--myisam-recover` option, `MyISAM` tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- `myisamchk` marks tables as checked if you run it with the `--update-state` option. `myisamchk --fast` checks only those tables that don't have this mark.
- `myisamchk --analyze` stores statistics for portions of keys, as well as for entire keys.
- `myisampack` can pack `BLOB` and `VARCHAR` columns.

`MyISAM` also supports the following features:

- Support for a true `VARCHAR` type; a `VARCHAR` column starts with a length stored in one or two bytes.
- Tables with `VARCHAR` columns may have fixed or dynamic row length.
- The sum of the lengths of the `VARCHAR` and `CHAR` columns in a table may be up to 64KB.
- Arbitrary length `UNIQUE` constraints.

Additional Resources

- A forum dedicated to the `MyISAM` storage engine is available at http://forums.mysql.com/list.php?21.

14.1.1 MyISAM Startup Options

The following options to `mysqld` can be used to change the behavior of `MyISAM` tables. For additional information, see Section 5.1.3, “Server Command Options”.

Table 14.1 MyISAM Option/Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
concurrent_insert	Yes	Yes	Yes		Global	Yes
delay-key-write	Yes	Yes			Global	Yes
- *Variable*: delay_key_write			Yes		Global	Yes
have_rtree_keys			Yes		Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes
log-isam	Yes	Yes				

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Yes
myisam_max_extra_sort_file_size	Yes	Yes	Yes		Global	No
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Yes
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam-recover	Yes	Yes				
- *Variable*: myisam_recover_options						
myisam_recover_options			Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Yes
myisam_stats_method	Yes	Yes	Yes		Both	Yes
skip-concurrent-insert	Yes	Yes				
- *Variable*: concurrent_insert						
tmp_table_size	Yes	Yes	Yes		Both	Yes

- `--myisam-recover=mode`

  Set the mode for automatic recovery of crashed `MyISAM` tables.

- `--delay-key-write=ALL`

  Don't flush key buffers between writes for any `MyISAM` table.

Note

If you do this, you should not access `MyISAM` tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` does not eliminate this risk.

The following system variables affect the behavior of `MyISAM` tables. For additional information, see Section 5.1.4, “Server System Variables”.

- `bulk_insert_buffer_size`

  The size of the tree cache used in bulk insert optimization.

Note

This is a limit *per thread*!

- `myisam_max_sort_file_size`

  The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger

This documentation is for an older version. If you're
This documentation is for an older version. If you're

than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_sort_buffer_size`

  Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `--myisam-recover` option. In this case, when the server opens a `MyISAM` table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `--myisam-recover` option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

If the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form *`tbl_name-datetime`*`.BAK`. You should have a `cron` script that automatically moves these files from the database directories to backup media.

14.1.2 Space Needed for Keys

`MyISAM` tables use B-tree indexes. You can roughly calculate the size for the index file as `(key_length+4)/0.67`, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In `MyISAM` tables, you can also prefix compress numbers by specifying the `PACK_KEYS=1` table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

14.1.3 MyISAM Table Storage Formats

`MyISAM` supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be

This documentation is for an older version. If you're
This documentation is for an older version. If you're

created only with the `myisampack` utility (see Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”).

When you use `CREATE TABLE` or `ALTER TABLE` for a table that has no `BLOB` or `TEXT` columns, you can force the table format to `FIXED` or `DYNAMIC` with the `ROW_FORMAT` table option.

See Section 13.1.10, “CREATE TABLE Syntax”, for information about `ROW_FORMAT`.

You can decompress (unpack) compressed `MyISAM` tables using `myisamchk --unpack`; see Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”, for more information.

14.1.3.1 Static (Fixed-Length) Table Characteristics

Static format is the default for `MyISAM` tables. It is used when the table contains no variable-length columns (`VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT`). Each row is stored using a fixed number of bytes.

Of the three `MyISAM` storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format `MyISAM` file. In this case, `myisamchk` can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. `MyISAM` table indexes can always be reconstructed based on the data rows.

Note

Fixed-length row format is only available for tables without `BLOB` or `TEXT` columns. Creating a table with these columns with an explicit `ROW_FORMAT` clause will not raise an error or warning; the format specification will be ignored.

Static-format tables have these characteristics:

- `CHAR` and `VARCHAR` columns are space-padded to the specified column width, although the column type is not altered. This is also true for `NUMERIC` and `DECIMAL` columns created before MySQL 5.0.3. `BINARY` and `VARBINARY` columns are space-padded to the column width before MySQL 5.0.15. As of 5.0.15, `BINARY` and `VARBINARY` columns are padded with `0x00` bytes.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use `OPTIMIZE TABLE` or `myisamchk -r`.
- Usually require more disk space than dynamic-format tables.

14.1.3.2 Dynamic Table Characteristics

Dynamic storage format is used if a `MyISAM` table contains any variable-length columns (`VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT`), or if the table was created with the `ROW_FORMAT=DYNAMIC` table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in noncontiguous pieces) when it is made longer as a result of an update.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can use `OPTIMIZE TABLE` or `myisamchk -r` to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). This does not include columns that contain `NULL` values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Nonempty strings are saved as a length byte plus the string contents.
- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run `OPTIMIZE TABLE` or `myisamchk -r` from time to time to improve performance. Use `myisamchk -ei` to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8
```

  There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using `myisamchk -ed`. All links may be removed with `OPTIMIZE TABLE` or `myisamchk -r`.

14.1.3.3 Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the `myisampack` tool. Compressed tables can be uncompressed with `myisamchk`.

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
  - Suffix space compression.
  - Prefix space compression.
  - Numbers with a value of zero are stored using one bit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.
- If a column has only a small set of possible values, the data type is converted to `ENUM`.
- A column may use any combination of the preceding compression types.

- Can be used for fixed-length or dynamic-length rows.

Note

While a compressed table is read only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use `DROP` to drop the table, and `TRUNCATE TABLE` to empty the table.

14.1.4 MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

14.1.4.1 Corrupted MyISAM Tables

Even though the `MyISAM` table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The `mysqld` process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as `myisamchk`) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or `MyISAM` code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a `MyISAM` table using the `CHECK TABLE` statement, and repair a corrupted `MyISAM` table with `REPAIR TABLE`. When `mysqld` is not running, you can also check or repair a table with the `myisamchk` command. See Section 13.7.2.3, “CHECK TABLE Syntax”, Section 13.7.2.6, “REPAIR TABLE Syntax”, and Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent `restarted mysqld` message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption

This documentation is for an older version. If you're
This documentation is for an older version. If you're

may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”, and Section 21.3, “Debugging and Porting MySQL”.

14.1.4.2 Problems from Tables Not Being Closed Properly

Each `MyISAM` index file (`.MYI` file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from `CHECK TABLE` or `myisamchk`, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a `FLUSH TABLES` operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A `MyISAM` table is copied without first issuing `LOCK TABLES` and `FLUSH TABLES`.
- MySQL has crashed between an update and the final close. (The table may still be okay because MySQL always issues writes for everything between each statement.)
- A table was modified by `myisamchk --recover` or `myisamchk --update-state` at the same time that it was in use by `mysqld`.
- Multiple `mysqld` servers are using the table and one server performed a `REPAIR TABLE` or `CHECK TABLE` on the table while it was in use by another server. In this setup, it is safe to use `CHECK TABLE`, although you might get the warning from other servers. However, `REPAIR TABLE` should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

  In general, it is a bad idea to share a data directory among multiple servers. See Section 5.5, “Running Multiple MySQL Instances on One Machine”, for additional discussion.

14.2 The InnoDB Storage Engine

Key Advantages of InnoDB

`InnoDB` is a high-reliability and high-performance storage engine for MySQL. Key advantages of `InnoDB` include:

- Its design follows the ACID model, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data.
- Row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent reads increase multi-user concurrency and performance.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `InnoDB` tables arrange your data on disk to optimize common queries based on primary keys. Each `InnoDB` table has a primary key index called the clustered index that organizes the data to minimize I/O for primary key lookups.
- To maintain data integrity, `InnoDB` also supports `FOREIGN KEY` referential-integrity constraints.
- You can freely mix `InnoDB` tables with tables from other MySQL storage engines, even within the same statement. For example, you can use a join operation to combine data from `InnoDB` and `MEMORY` tables in a single query.
- `InnoDB` has been designed for CPU efficiency and maximum performance when processing large data volumes.

To determine whether your server supports `InnoDB` use the `SHOW ENGINES` statement. See Section 13.7.5.13, “SHOW ENGINES Syntax”.

InnoDB Storage Engine Features

The `InnoDB` storage engine maintains its own buffer pool for caching data and indexes in main memory. `InnoDB` stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, `MyISAM` tables where each table is stored using separate files. `InnoDB` tables can be very large even on operating systems where file size is limited to 2GB.

The Windows Essentials installer makes `InnoDB` the MySQL default storage engine on Windows, if the server being installed supports `InnoDB`.

MySQL Enterprise Backup and InnoDB

The MySQL Enterprise Backup product lets you back up a running MySQL database, including `InnoDB` and `MyISAM` tables, with minimal disruption to operations while producing a consistent snapshot of the database. When MySQL Enterprise Backup is copying `InnoDB` tables, reads and writes to both `InnoDB` and `MyISAM` tables can continue. During the copying of `MyISAM` tables, reads (but not writes) to those tables are permitted. In addition, MySQL Enterprise Backup supports creating compressed backup files, and performing backups of subsets of `InnoDB` tables. In conjunction with MySQL’s binary log, users can perform point-in-time recovery. MySQL Enterprise Backup is commercially licensed. For a more complete description of MySQL Enterprise Backup, see Section 22.2, “MySQL Enterprise Backup Overview”.

Additional Resources

A forum dedicated to the `InnoDB` storage engine is available at http://forums.mysql.com/list.php?22.

`InnoDB` is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see http://www.mysql.com/company/legal/licensing/.

14.2.1 Configuring InnoDB

If you do not want to use `InnoDB` tables, start the server with the `--skip-innodb` option to disable the `InnoDB` storage engine. In this case, the server will not start if the default storage engine is set to `InnoDB`. Use `--default-storage-engine` to set the default to some other engine if necessary.

Caution

`InnoDB` is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write

This documentation is for an older version. If you're

operations to improve performance. On some operating systems, the very `fsync()` system call that should wait until all unwritten data for a file has been flushed might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, you should perform some "pull-the-plug" tests before using anything in production. On OS X 10.3 and up, `InnoDB` uses a special `fcntl()` file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATA/SATA disk drives, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

With regard to `InnoDB` recovery capabilities that protect user data, `InnoDB` uses a file flush technique involving a structure called the doublewrite buffer, which is enabled by default (`innodb_doublewrite=ON`). The doublewrite buffer adds safety to recovery following a crash or power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations. It is recommended that the `innodb_doublewrite` option remains enabled if you are concerned with data integrity or possible failures. For additional information about the doublewrite buffer, see Section 14.2.11, "InnoDB Disk I/O and File Space Management".

Overview of InnoDB Tablespace and Log Files

Two important disk-based resources managed by the `InnoDB` storage engine are its tablespace data files and its log files. If you specify no `InnoDB` configuration options, MySQL creates an auto-extending data file, slightly larger than 10MB, named `ibdata1` and two log files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. Their size is given by the size of the `innodb_log_file_size` system variable. To get good performance, explicitly provide `InnoDB` parameters as discussed in the following examples. Naturally, edit the settings to suit your hardware and requirements.

Caution

It is not a good idea to configure `InnoDB` to use data files or log files on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

The examples shown here are representative. See Section 14.2.2, "InnoDB Startup Options and System Variables" for additional information about `InnoDB`-related configuration parameters.

To set up the `InnoDB` tablespace files, use the `innodb_data_file_path` option in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. The value of `innodb_data_file_path` should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (`;`) characters:

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

For example, the following setting explicitly creates a tablespace having the same characteristics as the default:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This setting configures a single 10MB data file named `ibdata1` that is auto-extending. No location for the file is given, so by default, `InnoDB` creates it in the MySQL data directory.

Sizes are specified using `K`, `M`, or `G` suffix letters to indicate units of KB, MB, or GB.

A tablespace containing a fixed-size 50MB data file named `ibdata1` and a 50MB auto-extending file named `ibdata2` in the data directory can be configured like this:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the file name, its size, and several optional attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The `autoextend` and `max` attributes can be used only for the last data file in the `innodb_data_file_path` line.

If you specify the `autoextend` option for the last data file, `InnoDB` extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time by default. To modify the increment, change the `innodb_autoextend_increment` system variable.

If the disk becomes full, you might want to add another data file on another disk. For tablespace reconfiguration instructions, see Section 14.2.5, “Resizing the InnoDB System Tablespace”.

`InnoDB` is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the `max` attribute following the `autoextend` attribute. The following configuration permits `ibdata1` to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

`InnoDB` creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the `innodb_data_home_dir` option. For example, to use two files named `ibdata1` and `ibdata2` but create them in the `/ibdata` directory, configure `InnoDB` like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Note

`InnoDB` does not create directories, so make sure that the `/ibdata` directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS `mkdir` command to create any necessary directories.

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any directory where it needs to create data files or log files.

`InnoDB` forms the directory path for each data file by textually concatenating the value of `innodb_data_home_dir` to the data file name, adding a path name separator (slash or backslash) between values if necessary. If the `innodb_data_home_dir` option is not specified in `my.cnf` at all,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the default value is the “dot” directory `./`, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify `innodb_data_home_dir` as an empty string, you can specify absolute paths for the data files listed in the `innodb_data_file_path` value. The following example is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

Specifying InnoDB Configuration Options

Sample `my.cnf` file for small systems. Suppose that you have a computer with 512MB RAM and one hard disk. The following example shows possible configuration parameters in `my.cnf` or `my.ini` for `InnoDB`, including the `autoextend` attribute. The example suits most users, both on Unix and Windows, who do not want to distribute `InnoDB` data files and log files onto several disks. It creates an auto-extending data file `ibdata1` and two `InnoDB` log files `ib_logfile0` and `ib_logfile1` in the MySQL data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=256M
innodb_additional_mem_pool_size=20M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=64M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Note that data files must be less than 2GB in some file systems. The combined size of the log files must be less than 4GB. The combined size of data files must be at least slightly larger than 10MB.

Setting Up the InnoDB System Tablespace

When you create an `InnoDB` system tablespace for the first time, it is best that you start the MySQL server from the command prompt. `InnoDB` then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if `mysqld` is located in `C:\Program Files\MySQL\MySQL Server 5.0\bin`, you can start it like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --console
```

If you do not send server output to the screen, check the server's error log to see what `InnoDB` prints during the startup process.

For an example of what the information displayed by `InnoDB` should look like, see Section 14.2.1.1, “Initializing InnoDB”.

Editing the MySQL Configuration File

You can place `InnoDB` options in the `[mysqld]` group of any option file that your server reads when it starts. The locations for option files are described in Section 4.2.6, “Using Option Files”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you installed MySQL on Windows using the installation and configuration wizards, the option file will be the `my.ini` file located in your MySQL installation directory. See Section 2.10.3.1, “Starting the MySQL Server Instance Configuration Wizard”.

If your PC uses a boot loader where the `C:` drive is not the boot drive, your only option is to use the `my.ini` file in your Windows directory (typically `C:\WINDOWS`). You can use the `SET` command at the command prompt in a console window to print the value of `WINDIR`:

```
C:\> SET WINDIR
windir=C:\WINDOWS
```

To make sure that `mysqld` reads options only from a specific file, use the `--defaults-file` option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my_cnf
```

Sample `my.cnf` file for large systems. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks at directory paths `/`, `/dr2` and `/dr3`. The following example shows possible configuration parameters in `my.cnf` for `InnoDB`.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:2000M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next line if you want to use it
#innodb_thread_concurrency=5
```

In some cases, database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. `InnoDB` fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as `InnoDB` data files, which may speed up I/O. See Section 14.2.1.3, “Using Raw Devices for the System Tablespace”.

Determining the Maximum Memory Allocation for InnoDB

Warning

On 32-bit GNU/Linux x86, be careful not to set memory usage too high. `glibc` may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL binaries provided by Oracle Corporation.) and in the worst case also uses `sort_buffer_size + read_buffer_size` additional memory.

By compiling MySQL yourself, you can use up to 64GB of physical memory in 32-bit Windows. See the description for `innodb_buffer_pool_awe_mem_mb` in Section 14.2.2, “InnoDB Startup Options and System Variables”.

Tuning other `mysqld` server parameters. The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=value
```

On Linux, if the kernel is enabled for large page support, `InnoDB` can use large pages to allocate memory for its buffer pool and additional memory pool. See Section 8.12.5.2, “Enabling Large Page Support”.

14.2.1.1 Initializing InnoDB

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary `InnoDB` configuration parameters. Before starting MySQL, you should verify that the directories you have specified for `InnoDB` data files and log files exist and that the MySQL server has access rights to those directories. `InnoDB` does not create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server `mysqld` from the command prompt when you first start the server with `InnoDB` enabled, not from `mysqld_safe` or as a Windows service. When you run from a command prompt you see what `mysqld` prints and what is happening. On Unix, just invoke `mysqld`. On Windows, start `mysqld` with the `--console` option to direct the output to the console window.

When you start the MySQL server after initially configuring `InnoDB` in your option file, `InnoDB` creates your data files and log files, and prints something like this:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

At this point `InnoDB` has initialized its tablespace and log files. You can connect to the MySQL server with the usual MySQL client programs like `mysql`. When you shut down the MySQL server with `mysqladmin shutdown`, the output is like this:

```
010321 18:33:34  mysqld: Normal shutdown
010321 18:33:34  mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

You can look at the data file and log directories and you see the files created there. When MySQL is started again, the data files and log files have been created already, so the output is much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

If you add the `innodb_file_per_table` option to `my.cnf`, `InnoDB` stores each table in its own `.ibd` file in the same MySQL database directory where the `.frm` file is created. See Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”.

14.2.1.2 Dealing with InnoDB Initialization Problems

If `InnoDB` prints an operating system error during a file operation, usually the problem has one of the following causes:

- You did not create the `InnoDB` data file directory or the `InnoDB` log directory.
- `mysqld` does not have access rights to create files in those directories.
- `mysqld` cannot read the proper `my.cnf` or `my.ini` option file, and consequently does not see the options that you specified.
- The disk is full or a disk quota is exceeded.
- You have created a subdirectory whose name is equal to a data file that you specified, so the name cannot be used as a file name.
- There is a syntax error in the `innodb_data_home_dir` or `innodb_data_file_path` value.

If something goes wrong when `InnoDB` attempts to initialize its tablespace or its log files, delete all files created by `InnoDB`. This means all `ibdata` files and all `ib_logfile` files. In case you have already created some `InnoDB` tables, delete the corresponding `.frm` files for these tables (and any `.ibd` files if you are using multiple tablespaces) from the MySQL database directories as well. Then you can try the `InnoDB` database creation again. It is best to start the MySQL server from a command prompt so that you see what is happening.

14.2.1.3 Using Raw Devices for the System Tablespace

You can use raw disk partitions as data files in the `InnoDB` system tablespace. By using a raw disk, you can perform nonbuffered I/O on Windows and on some Unix systems without file system overhead. This may improve performance, but you are advised to perform tests with and without raw partitions to verify whether this is actually so on your system.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When you use a raw disk partition, be sure that it has permissions that enable read and write access by the account used for running the MySQL server. For example, if you run the server as the `mysql` user, the partition must permit read and write access to `mysql`. If you run the server with the `--memlock` option, the server must be run as `root`, so the partition must permit access to `root`.

The procedures described below involve option file modification. For additional information, see Section 4.2.6, “Using Option Files”.

Allocating a Raw Disk Partition on Linux and Unix Systems

1. When you create a new data file, specify the keyword `newraw` immediately after the data file size for the `innodb_data_file_path` option. The partition must be at least as large as the size that you specify. Note that 1MB in `InnoDB` is 1024 × 1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
   ```

2. Restart the server. `InnoDB` notices the `newraw` keyword and initializes the new partition. However, do not create or change any `InnoDB` tables yet. Otherwise, when you next restart the server, `InnoDB` reinitializes the partition and your changes are lost. (As a safety measure `InnoDB` prevents users from modifying data when any partition with `newraw` is specified.)

3. After `InnoDB` has initialized the new partition, stop the server, change `newraw` in the data file specification to `raw`:

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=/dev/hdd1:3Graw;/dev/hdd2:2Graw
   ```

4. Restart the server. `InnoDB` now permits changes to be made.

Allocating a Raw Disk Partition on Windows

On Windows systems, the same steps and accompanying guidelines described for Linux and Unix systems apply except that the `innodb_data_file_path` setting differs slightly on Windows.

1. When you create a new data file, specify the keyword `newraw` immediately after the data file size for the `innodb_data_file_path` option:

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=//./D::10Gnewraw
   ```

   The `//./` corresponds to the Windows syntax of `\\.\` for accessing physical drives. In the example above, `D:` is the drive letter of the partition.

2. Restart the server. `InnoDB` notices the `newraw` keyword and initializes the new partition.

3. After `InnoDB` has initialized the new partition, stop the server, change `newraw` in the data file specification to `raw`:

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=//./D::10Graw
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4. Restart the server. `InnoDB` now permits changes to be made.

14.2.1.4 InnoDB File-Per-Table Tablespaces

You can store each `InnoDB` table and its indexes in its own data file. This feature is called “file-per-table tablespaces” because in effect each table has its own tablespace.

Advantages of File-Per-Table Tablespaces

- You can reclaim disk space when truncating or dropping a table stored in a file-per-table tablepace. Truncating or dropping tables stored in the system tablespace creates free space internally in the system tablespace data files (ibdata files) which can only be used for new `InnoDB` data.
- The `TRUNCATE TABLE` operation is faster when run on individual `.ibd` files.
- You can store specific tables on separate storage devices, for I/O optimization, space management, or backup purposes.
- You can run `OPTIMIZE TABLE` to compact or recreate a file-per-table tablespace. When you run an `OPTIMIZE TABLE`, `InnoDB` creates a new `.ibd` file with a temporary name, using only the space required to store actual data. When the optimization is complete, `InnoDB` removes the old `.ibd` file and replaces it with the new one. If the previous `.ibd` file grew significantly but the actual data only accounted for a portion of its size, running `OPTIMIZE TABLE` can reclaim the unused space.
- You can move individual `InnoDB` tables rather than entire databases.
- You can enable more efficient storage for tables with large `BLOB` or `TEXT` columns using the dynamic row format.
- File-per-table tablespaces may improve chances for a successful recovery and save time when a corruption occurs, when a server cannot be restarted, or when backup and binary logs are unavailable.
- You can back up or restore a single table quickly, without interrupting the use of other `InnoDB` tables.
- You can excluded tables stored in file-per-table tablespaces from a backup. This is beneficial if you have tables that require backup less frequently or on a different schedule.
- File-per-table tablespaces are convenient for per-table status reporting when copying or backing up tables.
- You can monitor table size at a file system level, without accessing MySQL.
- Common Linux file systems do not permit concurrent writes to a single file when `innodb_flush_method` is set to `O_DIRECT`. As a result, there are possible performance improvements when using `innodb_file_per_table` in conjunction with `innodb_flush_method`.
- The system tablespace stores the data dictionary and undo logs, and has a 64TB size limit. By comparison, each file-per-table tablespace has a 64TB size limit, which provides you with room for growth. See Section C.7.3, “Limits on Table Size” for related information.

Potential Disadvantages of File-Per-Table Tablespaces

- With file-per-table tablespaces, each table may have unused space, which can only be utilized by rows of the same table. This could lead to wasted space if not properly managed.
- `fsync` operations must run on each open table rather than on a one file. Because there is a separate `fsync` operation for each file, write operations on multiple tables cannot be combined into a single I/O operation. This may require `InnoDB` to perform a higher total number of `fsync` operations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `mysqld` must keep one open file handle per table, which may impact performance if you have numerous tables in file-per-table tablespaces.
- More file descriptors are used.
- If backward compatibility with MySQL 5.1 is a concern, be aware that enabling `innodb_file_per_table` means that `ALTER TABLE` will move `InnoDB` tables from the system tablespace to individual `.ibd` files.
- If many tables are growing there is potential for more fragmentation which can impede `DROP TABLE` and table scan performance. However, when fragmentation is managed, having files in their own tablespace can improve performance.
- The buffer pool is scanned when dropping a file-per-table tablespace, which can take several seconds for buffer pools that are tens of gigabytes in size. The scan is performed with a broad internal lock, which may delay other operations. Tables in the system tablespace are not affected.
- The `innodb_autoextend_increment` variable, which defines increment size (in MB) for extending the size of an auto-extending shared tablespace file when it becomes full, does not apply to file-per-table tablespace files, which are auto-extending regardless of the `innodb_autoextend_increment` setting. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

Enabling and Disabling File-Per-Table Tablespaces

To enable file-per-table tablespaces, start the server with the `--innodb_file_per_table` option. For example, add a line to the `[mysqld]` section of `my.cnf`:

```
[mysqld]
innodb_file_per_table
```

With `innodb_file_per_table` enabled, `InnoDB` stores each newly created table in its own *`tbl_name`*`.ibd` file in the database directory where the table belongs. This is similar to what the `MyISAM` storage engine does, but `MyISAM` divides the table into a *`tbl_name`*`.MYD` data file and an *`tbl_name`*`.MYI` index file. For `InnoDB`, the data and the indexes are stored together in the `.ibd` file. The *`tbl_name`*`.frm` file is still created as usual.

You cannot freely move `.ibd` files between database directories as you can with `MyISAM` table files. This is because the table definition that is stored in the `InnoDB` shared tablespace includes the database name, and because `InnoDB` must preserve the consistency of transaction IDs and log sequence numbers.

If you remove the `innodb_file_per_table` line from `my.cnf` and restart the server, newly created `InnoDB` tables are created inside the shared tablespace files again.

The `--innodb_file_per_table` option affects only table creation, not access to existing tables. If you start the server with this option, new tables are created using `.ibd` files, but you can still access tables that exist in the shared tablespace. If you start the server without this option, new tables are created in the shared tablespace, but you can still access tables created in file-per-table tablespaces.

Note

`InnoDB` requires the shared tablespace to store its internal data dictionary and undo logs. The `.ibd` files alone are not sufficient for `InnoDB` to operate.

To move an `.ibd` file and the associated table from one database to another, use a `RENAME TABLE` statement:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an `.ibd` file, you can restore it to the MySQL installation from which it originated as follows:

1. Issue this `ALTER TABLE` statement to delete the current `.ibd` file:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

2. Copy the backup `.ibd` file to the proper database directory.

3. Issue this `ALTER TABLE` statement to tell `InnoDB` to use the new `.ibd` file for the table:

   ```
   ALTER TABLE tbl_name IMPORT TABLESPACE;
   ```

In this context, a “clean” `.ibd` file backup is one for which the following requirements are satisfied:

- There are no uncommitted modifications by transactions in the `.ibd` file.
- There are no unmerged insert buffer entries in the `.ibd` file.
- Purge has removed all delete-marked index records from the `.ibd` file.
- `mysqld` has flushed all modified pages of the `.ibd` file from the buffer pool to the file.

You can make a clean backup `.ibd` file using the following method:

1. Stop all activity from the `mysqld` server and commit all transactions.
2. Wait until `SHOW ENGINE INNODB STATUS` shows that there are no active transactions in the database, and the main thread status of `InnoDB` is `Waiting for server activity`. Then you can make a copy of the `.ibd` file.

Another method for making a clean copy of an `.ibd` file is to use the commercial `InnoDB Hot Backup` tool:

1. Use `InnoDB Hot Backup` to back up the `InnoDB` installation.
2. Start a second `mysqld` server on the backup and let it clean up the `.ibd` files in the backup.

14.2.2 InnoDB Startup Options and System Variables

This section describes the `InnoDB`-related command options and system variables.

- System variables that are true or false can be enabled at server startup by naming them, or disabled by using a `--skip-` prefix. For example, to enable or disable `InnoDB` checksums, you can use `--innodb_checksums` or `--skip-innodb_checksums` on the command line, or `innodb_checksums` or `skip-innodb_checksums` in an option file.
- System variables that take a numeric value can be specified as `--var_name=value` on the command line or as `var_name=value` in option files.
- Many system variables can be changed at runtime (see Section 5.1.5.2, “Dynamic System Variables”).
- For information about `GLOBAL` and `SESSION` variable scope modifiers, refer to the `SET` statement documentation.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- For more information on specifying options and system variables, see Section 4.2.3, “Specifying Program Options”.

Table 14.2 `InnoDB` Option/Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Com_show_innodb_status				Yes	Both	No
foreign_key_checks			Yes		Session	Yes
have_innodb			Yes		Global	No
innodb	Yes	Yes				
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	No
innodb_additional_mem_pool_size	Yes	Yes	Yes		Global	No
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_awe_mem_mb	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_ahead_seq				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_checksums	Yes	Yes	Yes		Global	No
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes
innodb_file_io_threads	Yes	Yes	Yes		Global	No
innodb_file_per_table	Yes	Yes	Yes		Global	No
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_lock_wait_timeout	Yes	Yes	Yes		Global	No
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_arch_dir	Yes	Yes	Yes		Global	No
innodb_log_archive	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
Innodb_log_waits				Yes	Global	No
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_mirrored_log_groups	Yes	Yes	Yes		Global	No
innodb_open_files	Yes	Yes	Yes		Global	No
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
Innodb_page_size				Yes	Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb-safe-binlog	Yes	Yes				
innodb-status-file	Yes	Yes				
innodb_support_xa	Yes	Yes	Yes		Both	Yes
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_use_legacy_cardinality_algorithm	Yes	Yes	Yes		Global	Yes
timed_mutexes	Yes	Yes	Yes		Global	Yes
unique_checks			Yes		Session	Yes

InnoDB Command Options

- `--innodb`

  Enables the `InnoDB` storage engine, if the server was compiled with `InnoDB` support.

  To disable `InnoDB`, use `--skip-innodb`. In this case, the server will not start if the default storage engine is set to `InnoDB`. Use `--default-storage-engine` to set the default to some other engine if necessary.

- `--innodb-status-file`

  Controls whether `InnoDB` creates a file named `innodb_status.<pid>` in the MySQL data directory. If enabled, `InnoDB` periodically writes the output of `SHOW ENGINE INNODB STATUS` to this file.

  By default, the file is not created. To create it, start `mysqld` with the `--innodb-status-file=1` option. The file is deleted during normal shutdown.

- `--skip-innodb`

  Disable the `InnoDB` storage engine. See the description of `--innodb`.

InnoDB System Variables

- `innodb_adaptive_hash_index`

Introduced	5.0.52	
Command-Line Format	`--innodb_adaptive_hash_index=#`	
System Variable	**Name**	`innodb_adaptive_hash_index`
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	No
Permitted Values	**Type**	`boolean`
	Default	`ON`

Whether `InnoDB` adaptive hash indexes are enabled or disabled (see Section 14.2.10.4, "Adaptive Hash Indexes"). This variable is enabled by default. Use `--skip-innodb_adaptive_hash_index` at server startup to disable it. This variable was added in MySQL 5.0.52.

- `innodb_additional_mem_pool_size`

Command-Line Format	`--innodb_additional_mem_pool_size=#`	
System Variable	**Name**	`innodb_additional_mem_pool_size`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`1048576`
	Min Value	`524288`
	Max Value	`4294967295`

The size in bytes of a memory pool `InnoDB` uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If `InnoDB` runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 1MB.

- `innodb_autoextend_increment`

Command-Line Format	`--innodb_autoextend_increment=#`	
System Variable	**Name**	`innodb_autoextend_increment`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`8`
	Min Value	`1`
	Max Value	`1000`

The increment size (in MB) for extending the size of an auto-extending shared tablespace file when it becomes full. The default value is 8. This variable does not affect the file-per-table tablespace files that are created if you use `innodb_file_per_table=1`. Those files are auto-extending regardless of the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

value of `innodb_autoextend_increment`. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

- `innodb_buffer_pool_awe_mem_mb`

Command-Line Format	`--innodb_buffer_pool_awe_mem_mb=#`	
System Variable	**Name**	`innodb_buffer_pool_awe_mem_mb`
	Variable Scope	Global
	Dynamic Variable	No
Platform Specific	Windows	
Permitted Values (Windows)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`63000`

The size of the buffer pool (in MB), if it is placed in the AWE memory. If it is greater than 0, `innodb_buffer_pool_size` is the window in the 32-bit address space of `mysqld` where `InnoDB` maps that AWE memory. A good value for `innodb_buffer_pool_size` is 500MB. The maximum possible value is 63000.

To take advantage of AWE memory, you will need to recompile MySQL yourself. The current project settings needed for doing this can be found in the `innobase/os/os0proc.c` source file.

This variable is relevant only in 32-bit Windows. If your 32-bit Windows operating system supports more than 4GB memory, using so-called “Address Windowing Extensions,” you can allocate the `InnoDB` buffer pool into the AWE physical memory using this variable.

- `innodb_buffer_pool_size`

Command-Line Format	`--innodb_buffer_pool_size=#`	
System Variable	**Name**	`innodb_buffer_pool_size`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`8388608`
	Min Value	`1048576`

The size in bytes of the memory buffer `InnoDB` uses to cache data and indexes of its tables. The default value is 8MB. The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. However, do not set it too large because competition for physical memory might cause paging in the operating system. Also, the time to initialize the buffer pool is roughly proportional to its size. On large

This documentation is for an older version. If you're
This documentation is for an older version. If you're

installations, this initialization time may be significant. For example, on a modern Linux x86_64 server, initialization of a 10GB buffer pool takes approximately 6 seconds. See Section 8.10.2, “The InnoDB Buffer Pool”

- `innodb_checksums`

Introduced	5.0.3	
Command-Line Format	`--innodb_checksums`	
System Variable	**Name**	`innodb_checksums`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`boolean`
	Default	`ON`

`InnoDB` can use checksum validation on all pages read from the disk to ensure extra fault tolerance against broken hardware or data files. This validation is enabled by default. However, under some rare circumstances (such as when running benchmarks) this extra safety feature is unneeded and can be disabled with `--skip-innodb_checksums`. This variable was added in MySQL 5.0.3.

- `innodb_commit_concurrency`

Introduced	5.0.12	
Command-Line Format	`--innodb_commit_concurrency=#`	
System Variable	**Name**	`innodb_commit_concurrency`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`1000`

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously. This variable was added in MySQL 5.0.12.

- `innodb_concurrency_tickets`

Introduced	5.0.3	
Command-Line Format	`--innodb_concurrency_tickets=#`	
System Variable	**Name**	`innodb_concurrency_tickets`
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`500`
	Min Value	`1`
	Max Value	`4294967295`

The number of threads that can enter `InnoDB` concurrently is determined by the `innodb_thread_concurrency` variable. A thread is placed in a queue when it tries to enter `InnoDB` if the number of threads has already reached the concurrency limit. When a thread is permitted to enter `InnoDB`, it is given a number of "free tickets" equal to the value of `innodb_concurrency_tickets`, and the thread can enter and leave `InnoDB` freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter `InnoDB`. The default value is 500. This variable was added in MySQL 5.0.3.

With a small `innodb_concurrency_tickets` value, small transactions that only need to process a few rows compete fairly with larger transactions that process many rows. The disadvantage of a small `innodb_concurrency_tickets` value is that large transactions must loop through the queue many times before they can complete, which extends the length of time required to complete their task.

With a large `innodb_concurrency_tickets` value, large transactions spend less time waiting for a position at the end of the queue (controlled by `innodb_thread_concurrency`) and more time retrieving rows. Large transactions also require fewer trips through the queue to complete their task. The disadvantage of a large `innodb_concurrency_tickets` value is that too many large transactions running at the same time can starve smaller transactions by making them wait a longer time before executing.

With a non-zero `innodb_thread_concurrency` value, you may need to adjust the `innodb_concurrency_tickets` value up or down to find the optimal balance between larger and smaller transactions. The `SHOW ENGINE INNODB STATUS` report shows the number of tickets remaining for an executing transaction in its current pass through the queue.

- `innodb_data_file_path`

Command-Line Format	`--innodb_data_file_path=name`	
System Variable	**Name**	`innodb_data_file_path`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`string`
	Default	`ibdata1:10M:autoextend`

The paths to individual data files and their sizes. The full directory path to each data file is formed by concatenating `innodb_data_home_dir` to each path specified here. The file sizes are specified in KB, MB, or GB (1024MB) by appending `K`, `M`, or `G` to the size value. The sum of the sizes of the files must be at least 10MB. If you do not specify `innodb_data_file_path`, the default behavior is to create a single 10MB auto-extending data file named `ibdata1`. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on those operating systems

This documentation is for an older version. If you're
This documentation is for an older version. If you're

that support big files. You can also use raw disk partitions as data files. For detailed information on configuring `InnoDB` tablespace files, see Section 14.2.1, “Configuring InnoDB”.

- `innodb_data_home_dir`

Command-Line Format	`--innodb_data_home_dir=dir_name`	
System Variable	**Name**	`innodb_data_home_dir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`

The common part of the directory path for all `InnoDB` data files in the shared tablespace. This setting does not affect the location of per-file tablespaces when `innodb_file_per_table` is enabled. The default value is the MySQL `data` directory. If you specify the value as an empty string, you can use absolute file paths in `innodb_data_file_path`.

- `innodb_doublewrite`

Introduced	5.0.3	
Command-Line Format	`--innodb-doublewrite`	
System Variable	**Name**	`innodb_doublewrite`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`boolean`
	Default	`ON`

If this variable is enabled (the default), `InnoDB` stores all data twice, first to the doublewrite buffer, and then to the actual data files. This variable can be turned off with `--skip-innodb_doublewrite` for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures. This variable was added in MySQL 5.0.3.

- `innodb_fast_shutdown`

Command-Line Format	`--innodb_fast_shutdown[=#]`	
System Variable	**Name**	`innodb_fast_shutdown`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`1`
	Valid Values	`0`
		`1`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

		2

The InnoDB shutdown mode. By default, the value is 1, which causes a “fast” shutdown (the normal type of shutdown). If the value is 0, InnoDB does a full purge and an insert buffer merge before a shutdown. These operations can take minutes, or even hours in extreme cases. If the value is 1, InnoDB skips these operations at shutdown. If the value is 2, InnoDB will just flush its logs and then shut down cold, as if MySQL had crashed; no committed transaction will be lost, but crash recovery will be done at the next startup. The value of 2 can be used as of MySQL 5.0.5, except that it cannot be used on NetWare.

- innodb_file_io_threads

Command-Line Format	--innodb_file_io_threads=#	
System Variable	**Name**	innodb_file_io_threads
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	integer
	Default	4
	Min Value	4
	Max Value	64

The number of file I/O threads in InnoDB. Normally, this should be left at the default value of 4, but disk I/O on Windows may benefit from a larger number. On Unix, increasing the number has no effect; InnoDB always uses the default value.

- innodb_file_per_table

Command-Line Format	--innodb_file_per_table	
System Variable	**Name**	innodb_file_per_table
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	boolean
	Default	OFF

If innodb_file_per_table is disabled (the default), InnoDB creates tables in the shared tablespace. If innodb_file_per_table is enabled, InnoDB creates each new table using its own .ibd file for storing data and indexes, rather than in the shared tablespace. See Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces” for more information including advantages and disadvantages of using file-per-table tablespaces.

- innodb_flush_log_at_trx_commit

Command-Line Format	--innodb_flush_log_at_trx_commit[=#]	
System Variable	**Name**	innodb_flush_log_at_trx_commit

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`enumeration`
	Default	`1`
	Valid Values	`0`
		`1`
		`2`

If the value of `innodb_flush_log_at_trx_commit` is 0, the log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit. When the value is 1 (the default), the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file. When the value is 2, the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also when the value is 2. Note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.

The default value of 1 is required for full ACID compliance. You can achieve better performance by setting the value different from 1, but then you can lose up to one second worth of transactions in a crash. With a value of 0, any `mysqld` process crash can erase the last second of transactions. With a value of 2, only an operating system crash or a power outage can erase the last second of transactions. `InnoDB`'s crash recovery works regardless of the value.

For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should use `innodb_flush_log_at_trx_commit=1`, `sync_binlog=1`, and, before MySQL 5.0.3, `innodb-safe-binlog` in your master server `my.cnf` file. (`innodb-safe-binlog` is not needed from 5.0.3 on.)

Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in the worst case a power outage can even corrupt the `InnoDB` database. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command `hdparm` to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor.

- `innodb_flush_method`

Command-Line Format	`--innodb_flush_method=name`	
System Variable	**Name**	`innodb_flush_method`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (Unix)	**Type**	`string`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	`NULL`
	Valid Values	`fdatasync`
		`O_DSYNC`
		`littlesync`
		`nosync`
		`O_DIRECT`
Permitted Values (Windows)	**Type**	`string`
	Default	`NULL`
	Valid Values	`async_unbuffered`
		`normal`
		`unbuffered`

Defines the method used to flush data to the `InnoDB` data files and log files, which can affect I/O throughput.

If `innodb_flush_method=NULL` on a Unix-like system, the `fdatasync` option is used by default. If `innodb_flush_method=NULL` on Windows, the `async_unbuffered` option is used by default.

The `innodb_flush_method` options for Unix-like systems include:

- `fdatasync`: `InnoDB` uses the `fsync()` system call to flush both the data and log files. `fsync` is the default setting. The `fdatasync` option name should not be confused with the `fdatasync()` system call, which is not used by InnoDB as of MySQL 3.23.41.
- `O_DSYNC`: `InnoDB` uses `O_SYNC` to open and flush the log files, and `fsync()` to flush the data files. `InnoDB` does not use `O_DSYNC` directly because there have been problems with it on many varieties of Unix.
- `O_DIRECT`: `InnoDB` uses `O_DIRECT` (or `directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. This option is available on some GNU/Linux versions, FreeBSD, and Solaris.
- `littlesync`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `nosync`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.

The `innodb_flush_method` options for Windows systems include:

- `async_unbuffered`: `InnoDB` uses Windows asynchronous I/O and non-buffered I/O. `async_unbuffered` is the default setting on Windows systems.
- `normal`: `InnoDB` uses a simulated asynchronous I/O and buffered I/O. This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `unbuffered`: `InnoDB` uses a simulated asynchronous I/O and non-buffered I/O. This option is used for internal performance testing and is currently unsupported. Use at your own risk.

How each settings affects performance depends on hardware configuration and workload. Benchmark your particular configuration to decide which setting to use, or whether to keep the default setting. Examine the `Innodb_data_fsyncs` status variable to see the overall number of `fsync()` calls for

This documentation is for an older version. If you're
This documentation is for an older version. If you're

each setting. The mix of read and write operations in your workload can affect how a setting performs. For example, on a system with a hardware RAID controller and battery-backed write cache, `O_DIRECT` can help to avoid double buffering between the `InnoDB` buffer pool and the operating system's file system cache. On some systems where `InnoDB` data and log files are located on a SAN, the default value or `O_DSYNC` might be faster for a read-heavy workload with mostly `SELECT` statements. Always test this parameter with hardware and workload that reflect your production environment.

- `innodb_force_recovery`

Command-Line Format	`--innodb_force_recovery=#`	
System Variable	**Name**	`innodb_force_recovery`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`6`

The crash recovery mode. Possible values are from 0 to 6. For the meanings of these values and important information about `innodb_force_recovery`, see Section 14.2.6.2, “Forcing InnoDB Recovery”.

Warning

Only set this variable to a value greater than 0 in an emergency situation, so that you can start `InnoDB` and dump your tables. As a safety measure, `InnoDB` prevents `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0.

- `innodb_lock_wait_timeout`

Command-Line Format	`--innodb_lock_wait_timeout=#`	
System Variable	**Name**	`innodb_lock_wait_timeout`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`50`
	Min Value	`1`
	Max Value	`1073741824`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The timeout in seconds an InnoDB transaction may wait for a row lock before giving up. The default value is 50 seconds. A transaction that tries to access a row that is locked by another InnoDB transaction will hang for at most this many seconds before issuing the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

When a lock wait timeout occurs, the current statement is not executed. The current transaction is *not* rolled back. (Until MySQL 5.0.13 InnoDB rolled back the entire transaction if a lock wait timeout happened. You can restore this behavior by starting the server with the `--innodb_rollback_on_timeout` option, available as of MySQL 5.0.32. See also Section 14.2.12, “InnoDB Error Handling”.)

`innodb_lock_wait_timeout` applies to InnoDB row locks only. A MySQL table lock does not happen inside InnoDB and this timeout does not apply to waits for table locks.

InnoDB does detect transaction deadlocks in its own lock table immediately and rolls back one transaction. The lock wait timeout value does not apply to such a wait.

- `innodb_locks_unsafe_for_binlog`

Command-Line Format	`--innodb_locks_unsafe_for_binlog`	
System Variable	Name	`innodb_locks_unsafe_for_binlog`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`boolean`
	Default	`OFF`

This variable affects how InnoDB uses gap locking for searches and index scans. Normally, InnoDB uses an algorithm called *next-key locking* that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order. See Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”.

By default, the value of `innodb_locks_unsafe_for_binlog` is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the variable, set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

Enabling `innodb_locks_unsafe_for_binlog` does not disable the use of gap locking for foreign-key constraint checking or duplicate-key checking.

The effect of enabling `innodb_locks_unsafe_for_binlog` is similar to but not identical to setting the transaction isolation level to `READ COMMITTED`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Enabling `innodb_locks_unsafe_for_binlog` is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- `innodb_locks_unsafe_for_binlog` can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

`READ COMMITTED` therefore offers finer and more flexible control than `innodb_locks_unsafe_for_binlog`. For additional details about the effect of isolation level on gap locking, see Section 13.3.6, “SET TRANSACTION Syntax”.

Enabling `innodb_locks_unsafe_for_binlog` may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled. Suppose that there is an index on the `id` column of the `child` table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where `id` is greater than 100. If the locks set on the index records in that range do not lock out inserts made in the gaps, another session can insert a new row into the table. Consequently, if you were to execute the same `SELECT` again within the same transaction, you would see a new row in the result set returned by the query. This also means that if new items are added to the database, `InnoDB` does not guarantee serializability. Therefore, if `innodb_locks_unsafe_for_binlog` is enabled, `InnoDB` guarantees at most an isolation level of `READ COMMITTED`. (Conflict serializability is still guaranteed.) For additional information about phantoms, see Section 14.2.8.3, “Avoiding the Phantom Problem Using Next-Key Locking”.

Starting from MySQL 5.0.2, enabling `innodb_locks_unsafe_for_binlog` has an additional effect. For `UPDATE` or `DELETE` statements, `InnoDB` holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. This greatly reduces the probability of deadlocks, but they can still happen. Note that enabling this variable still does not permit operations such as `UPDATE` to overtake other similar operations (such as another `UPDATE`) even when they affect different rows.

Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

In this case, table has no indexes, so searches and index scans use the hidden clustered index for record locking (see Section 14.2.10.1, “Clustered and Secondary Indexes”).

Suppose that one client performs an `UPDATE` using these statements:

```
SET autocommit = 0;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second client performs an `UPDATE` by executing these statements following those of the first client:

```
SET autocommit = 0;
UPDATE t SET b = 4 WHERE b = 2;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As InnoDB executes each UPDATE, it first acquires an exclusive lock for each row, and then determines whether to modify it. If InnoDB does not modify the row and innodb_locks_unsafe_for_binlog is enabled, it releases the lock. Otherwise, InnoDB retains the lock until the end of the transaction. This affects transaction processing as follows.

If innodb_locks_unsafe_for_binlog is disabled, the first UPDATE acquires x-locks and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second UPDATE blocks as soon as it tries to acquire any locks (because first update has retained locks on all rows), and does not proceed until the first UPDATE commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If innodb_locks_unsafe_for_binlog is enabled, the first UPDATE acquires x-locks and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

The second UPDATE proceeds part way before it blocks. It begins acquiring x-locks, and blocks when it tries to acquire one for a row still locked by first UPDATE. The second UPDATE does not proceed until the first UPDATE commits or rolls back:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); block and wait for first UPDATE to commit or roll back
```

In this case, the second UPDATE must wait for a commit or rollback of the first UPDATE, even though it affects different rows. The first UPDATE has an exclusive lock on row (2,3) that it has not released. As the second UPDATE scans rows, it tries to acquire an exclusive lock for that same row, which it cannot have.

- innodb_log_arch_dir

  This variable is unused, and is deprecated as of MySQL 5.0.24. It is removed in MySQL 5.1

- innodb_log_archive

  Whether to log InnoDB archive files. This variable is present for historical reasons, but is unused. Recovery from a backup is done by MySQL using its own log files, so there is no need to archive InnoDB log files. The default for this variable is 0.

- innodb_log_buffer_size

Command-Line Format	--innodb_log_buffer_size=#	
System Variable	Name	innodb_log_buffer_size

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`1048576`
	Min Value	`1048576`
	Max Value	`4294967295`

The size in bytes of the buffer that `InnoDB` uses to write to the log files on disk. The default value is 1MB. Sensible values range from 1MB to 8MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

- `innodb_log_file_size`

Command-Line Format	`--innodb_log_file_size=#`	
System Variable	**Name**	`innodb_log_file_size`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`5242880`
	Min Value	`1048576`
	Max Value	`4GB / innodb_log_files_in_group`

The size in bytes of each log file in a log group. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) cannot exceed a maximum value that is slightly less than 4GB. A pair of 2047 MB log files, for example, would allow you to approach the range limit but not exceed it. The default value is 5MB. Sensible values range from 1MB to 1/*N*-th of the size of the buffer pool, where *N* is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

- `innodb_log_files_in_group`

Command-Line Format	`--innodb_log_files_in_group=#`	
System Variable	**Name**	`innodb_log_files_in_group`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	2
	Min Value	2
	Max Value	100

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) value is 2.

- innodb_log_group_home_dir

Command-Line Format	--innodb_log_group_home_dir=dir_name	
System Variable	**Name**	innodb_log_group_home_dir
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	directory name

The directory path to the InnoDB log files. If you do not specify any InnoDB log variables, the default is to create two files named ib_logfile0 and ib_logfile1 in the MySQL data directory. Their size is given by the size of the innodb_log_file_size system variable.

- innodb_max_dirty_pages_pct

Command-Line Format	--innodb_max_dirty_pages_pct=#	
System Variable	**Name**	innodb_max_dirty_pages_pct
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	numeric
	Default	90
	Min Value	0
	Max Value	100

This is an integer in the range from 0 to 100. The default value is 90. The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value.

- innodb_max_purge_lag

Command-Line Format	--innodb_max_purge_lag=#	
System Variable	**Name**	innodb_max_purge_lag
	Variable Scope	Global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`

This variable controls how to delay `INSERT`, `UPDATE`, and `DELETE` operations when purge operations are lagging (see Section 14.2.9, "InnoDB Multi-Versioning"). The default value 0 (no delays).

The `InnoDB` transaction system maintains a list of transactions that have index records delete-marked by `UPDATE` or `DELETE` operations. Let the length of this list be *`purge_lag`*. When *`purge_lag`* exceeds `innodb_max_purge_lag`, each `INSERT`, `UPDATE`, and `DELETE` operation is delayed by ((*`purge_lag`*/`innodb_max_purge_lag`)×10)−5 milliseconds. The delay is computed in the beginning of a purge batch, every ten seconds. The operations are not delayed if purge cannot run because of an old consistent read view that could see the rows to be purged.

A typical setting for a problematic workload might be 1 million, assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged `InnoDB` table rows.

The lag value is displayed as the history list length in the `TRANSACTIONS` section of InnoDB Monitor output. For example, if the output includes the following lines, the lag value is 20:

```
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

- `innodb_mirrored_log_groups`

  The number of identical copies of log groups to keep for the database. This should be set to 1.

- `innodb_open_files`

Command-Line Format	`--innodb_open_files=#`	
System Variable	**Name**	`innodb_open_files`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`integer`
	Default	`300`
	Min Value	`10`
	Max Value	`4294967295`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This variable is relevant only if you use multiple tablespaces in `InnoDB`. It specifies the maximum number of `.ibd` files that `InnoDB` can keep open at one time. The minimum value is 10. The default value is 300.

The file descriptors used for `.ibd` files are for `InnoDB` only. They are independent of those specified by the `--open-files-limit` server option, and do not affect the operation of the table cache.

- `innodb_rollback_on_timeout`

Introduced	5.0.32	
Command-Line Format	`--innodb_rollback_on_timeout`	
System Variable	**Name**	`innodb_rollback_on_timeout`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`boolean`
	Default	`OFF`

In MySQL 5.0.13 and up, `InnoDB` rolls back only the last statement on a transaction timeout by default. If `--innodb_rollback_on_timeout` is specified, a transaction timeout causes `InnoDB` to abort and roll back the entire transaction (the same behavior as before MySQL 5.0.13). This variable was added in MySQL 5.0.32.

- `innodb-safe-binlog`

Deprecated	5.0.3	
Removed	5.0.3	
Command-Line Format	`--innodb-safe-binlog`	
Permitted Values	**Type**	`boolean`

If this option is given, then after a crash recovery by `InnoDB`, `mysqld` truncates the binary log after the last not-rolled-back transaction in the log. The option also causes `InnoDB` to print an error if the binary log is smaller or shorter than it should be. See Section 5.4.3, “The Binary Log”. This variable was removed in MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support. You should set `innodb_support_xa` to `ON` or 1 to ensure consistency. See `innodb_support_xa`.

- `innodb_support_xa`

Introduced	5.0.3	
Command-Line Format	`--innodb_support_xa`	
System Variable	**Name**	`innodb_support_xa`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Enables `InnoDB` support for two-phase commit in XA transactions, causing an extra disk flush for transaction preparation. This setting is the default. The XA mechanism is used internally and is essential for any server that has its binary log turned on and is accepting changes to its data from more than one thread. If you turn it off, transactions can be written to the binary log in a different order from the one in which the live database is committing them. This can produce different data when the binary log is replayed in disaster recovery or on a replication slave. Do not turn it off on a replication master server unless you have an unusual setup where only one thread is able to change data.

For a server that is accepting data changes from only one thread, it is safe and recommended to turn off this option to improve performance for `InnoDB` tables. For example, you can turn it off on replication slaves where only the replication SQL thread is changing data.

You can also turn off this option if you do not need it for safe binary logging or replication, and you also do not use an external XA transaction manager.

This variable was added in MySQL 5.0.3.

- `innodb_sync_spin_loops`

Introduced	5.0.3	
Command-Line Format	`--innodb_sync_spin_loops=#`	
System Variable	**Name**	`innodb_sync_spin_loops`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`20`
	Min Value	`0`
	Max Value	`4294967295`

The number of times a thread waits for an `InnoDB` mutex to be freed before the thread is suspended. The default value is 20. This variable was added in MySQL 5.0.3.

- `innodb_table_locks`

Command-Line Format	`--innodb_table_locks`	
System Variable	**Name**	`innodb_table_locks`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

If `autocommit = 0`, `InnoDB` honors `LOCK TABLES`; MySQL does not return from `LOCK TABLES ... WRITE` until all other threads have released all their locks to the table. The default value of

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`innodb_table_locks` is 1, which means that `LOCK TABLES` causes InnoDB to lock a table internally if `autocommit = 0`.

- `innodb_thread_concurrency`

Command-Line Format	`--innodb_thread_concurrency=#`	
System Variable	**Name**	`innodb_thread_concurrency`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (<= 5.0.7)	**Type**	`integer`
	Default	`8`
	Min Value	`1`
	Max Value	`1000`
Permitted Values (>= 5.0.8, <= 5.0.18)	**Type**	`integer`
	Default	`20`
	Min Value	`1`
	Max Value	`1000`
Permitted Values (>= 5.0.19, <= 5.0.20)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`1000`
Permitted Values (>= 5.0.21)	**Type**	`integer`
	Default	`8`
	Min Value	`0`
	Max Value	`1000`

`InnoDB` tries to keep the number of operating system threads concurrently inside `InnoDB` less than or equal to the limit given by this variable (`InnoDB` uses operating system threads to process user transactions). Once the number of threads reaches this limit, additional threads are placed into a wait state within a “First In, First Out” (FIFO) queue for execution. Threads waiting for locks are not counted in the number of concurrently executing threads.

The range of this variable is 0 to 1000. A value of 20 or higher is interpreted as infinite concurrency before MySQL 5.0.19. From 5.0.19 on, you can disable thread concurrency checking by setting the value to 0. Disabling thread concurrency checking enables `InnoDB` to create as many threads as it needs.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The default value has changed several times: 8 before MySQL 5.0.8, 20 (infinite) from 5.0.8 through 5.0.18, 0 (infinite) from 5.0.19 to 5.0.20, and 8 (finite) from 5.0.21 on.

Consider setting this variable if your MySQL instance shares CPU resources with other applications, or if your workload or number of concurrent users is growing. The correct setting depends on workload, computing environment, and the version of MySQL that you are running. You will need to test a range of values to determine the setting that provides the best performance. `innodb_thread_concurrency` is a dynamic variable, which allows you to experiment with different settings on a live test system. If a particular setting performs poorly, you can quickly set `innodb_thread_concurrency` back to 0.

Use the following guidelines to help find and maintain an appropriate setting:

- If the number of concurrent user threads for a workload is less than 64, set `innodb_thread_concurrency=0`.
- If your workload is consistently heavy or occasionally spikes, start by setting `innodb_thread_concurrency=128`, and lowering the value to 96, 80, 64, and so on, until you find the number of threads that provides the best performance. For example, suppose your system typically has 40 to 50 users, but periodically the number increases to 60, 70, or even 200. You find that performance is stable at 80 concurrent users but starts to show a regression above this number. In this case, you would set `innodb_thread_concurrency=80` to avoid impacting performance.
- If you do not want `InnoDB` to use more than a certain number of vCPUs for user threads (20 vCPUs for example), set `innodb_thread_concurrency` to this number (or possibly lower, depending on performance results). If your goal is to isolate MySQL from other applications, you may consider binding the `mysqld` process exclusively to the vCPUs. Be aware, however, that exclusive binding could result in non-optimal hardware usage if the `mysqld` process is not consistently busy. In this case, you might bind the `mysqld` process to the vCPUs but also allow other applications to use some or all of the vCPUs.

  Note

  From an operating system perspective, using a resource management solution (if available) to manage how CPU time is shared among applications may be preferable to binding the `mysqld` process. For example, you could assign 90% of vCPU time to a given application while other critical process *are not* running, and scale that value back to 40% when other critical processes *are* running.

- `innodb_thread_concurrency` values that are too high can cause performance regression due to increased contention on system internals and resources.
- In some cases, the optimal `innodb_thread_concurrency` setting can be smaller than the number of vCPUs.
- If an operation, such as an `ALTER TABLE` operation, completes in a few minutes on an idle system but takes hours on a busy system (perhaps aborting due to insufficient log space), a lower but still acceptable `innodb_thread_concurrency` value has been shown to allow such operations to complete in times comparable to those on an idle system. A lower but still acceptable `innodb_thread_concurrency` value, in this case, is a value that might reduce overall performance by 10 or 20 percent.
- Monitor and analyze your system regularly. Changes to workload, number of users, or computing environment may require that you adjust the `innodb_thread_concurrency` setting.

- `innodb_thread_sleep_delay`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Introduced	5.0.3	
Command-Line Format	`--innodb_thread_sleep_delay=#`	
System Variable	**Name**	`innodb_thread_sleep_delay`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (>= 5.0.8, <= 5.0.18)	**Type**	`integer`
	Default	`0`
	Min Value	`1000`
	Max Value	`4294967295`
Permitted Values (>= 5.0.19, <= 5.0.20)	**Type**	`integer`
	Default	`0`
	Min Value	`1000`
	Max Value	`4294967295`
Permitted Values (>= 5.0.21)	**Type**	`integer`
	Default	`8`
	Min Value	`1000`
	Max Value	`4294967295`

How long `InnoDB` threads sleep before joining the `InnoDB` queue, in microseconds. The default value is 10,000. A value of 0 disables sleep. This variable was added in MySQL 5.0.3.

- `innodb_use_legacy_cardinality_algorithm`

Introduced	5.0.82	
Command-Line Format	`--innodb_use_legacy_cardinality_algorithm=#`	
System Variable	**Name**	`innodb_use_legacy_cardinality_algorithm`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`ON`

`InnoDB` uses random numbers to generate dives into indexes for calculating index cardinality. However, under certain conditions, the algorithm does not generate random numbers, so `ANALYZE TABLE` sometimes does not update cardinality estimates properly. An alternative algorithm was introduced in MySQL 5.0.82 with better randomization properties, and the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`innodb_use_legacy_cardinality_algorithm`, system variable which algorithm to use. The default value of the variable is 1 (`ON`), to use the original algorithm for compatibility with existing applications. The variable can be set to 0 (`OFF`) to use the new algorithm with improved randomness.

You should also take into consideration the value of `sync_binlog`, which controls synchronization of the binary log to disk.

14.2.3 Creating and Using InnoDB Tables

To create an `InnoDB` table, specify an `ENGINE = InnoDB` option in the `CREATE TABLE` statement:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

The statement creates a table and an index on column `a` in the `InnoDB` tablespace that consists of the data files that you specified in `my.cnf`. In addition, MySQL creates a file `customers.frm` in the `test` directory under the MySQL database directory. Internally, `InnoDB` adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if `test` is the database in which the `customers` table is created, the entry is for `'test/customers'`. This means you can create a table of the same name `customers` in some other database, and the table names do not collide inside `InnoDB`.

You can query the amount of free space in the `InnoDB` tablespace by issuing a `SHOW TABLE STATUS` statement for any `InnoDB` table. The amount of free space in the tablespace appears in the `Comment` section in the output of `SHOW TABLE STATUS`. For example:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

The statistics `SHOW` displays for `InnoDB` tables are only approximate. They are used in SQL optimization. Table and index reserved sizes in bytes are accurate, though.

14.2.3.1 How to Use Transactions in InnoDB with Different APIs

By default, each client that connects to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. To use multiple-statement transactions, you can switch autocommit off with the SQL statement `SET autocommit = 0` and end each transaction with either `COMMIT` or `ROLLBACK`. If you want to leave autocommit on, you can begin your transactions within `START TRANSACTION` and end them with `COMMIT` or `ROLLBACK`. The following example shows two transactions. The first is committed; the second is rolled back.

```
shell> mysql test

mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+------+--------+
| a    | b      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as `COMMIT` to the MySQL server as strings just like any other SQL statements such as `SELECT` or `INSERT`. Some APIs also offer separate special transaction commit and rollback functions or methods.

14.2.3.2 Converting Tables from Other Storage Engines to InnoDB

To convert a non-`InnoDB` table to use `InnoDB` use `ALTER TABLE`:

```
ALTER TABLE t1 ENGINE=InnoDB;
```

Important

Do not convert MySQL system tables in the `mysql` database (such as `user` or `host`) to the `InnoDB` type. This is an unsupported operation. The system tables must always be of the `MyISAM` type.

`InnoDB` does not have a special optimization for separate index creation the way the `MyISAM` storage engine does. Therefore, it does not pay to export and import the table and create indexes afterward. The fastest way to alter a table to `InnoDB` is to do the inserts directly to an `InnoDB` table. That is, use `ALTER TABLE ... ENGINE=INNODB`, or create an empty `InnoDB` table with identical definitions and insert the rows with `INSERT INTO ... SELECT * FROM ...`.

If you have `UNIQUE` constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET unique_checks=0;
... import operation ...
SET unique_checks=1;
```

For big tables, this saves a lot of disk I/O because `InnoDB` can then use its insert buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. `unique_checks` permits but does not require storage engines to ignore duplicate keys.

To get better control over the insertion process, it might be good to insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, you should increase the size of the `InnoDB` buffer pool to reduce disk I/O. Do not use more than 80% of the physical memory, though. You can also increase the sizes of the `InnoDB` log files.

Make sure that you do not fill up the tablespace: `InnoDB` tables require a lot more disk space than `MyISAM` tables. If an `ALTER TABLE` operation runs out of space, it starts a rollback, and that can take hours if it

This documentation is for an older version. If you're
This documentation is for an older version. If you're

is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see Section 14.2.6.2, “Forcing InnoDB Recovery”.

If you want all your (nonsystem) tables to be created as InnoDB tables, add the line default-storage-engine=innodb to the [mysqld] section of your server option file.

14.2.3.3 AUTO_INCREMENT Handling in InnoDB

To use the AUTO_INCREMENT mechanism with an InnoDB table, an AUTO_INCREMENT column *ai_col* must be defined as part of an index such that it is possible to perform the equivalent of an indexed SELECT MAX(*ai_col*) lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

If you specify an AUTO_INCREMENT column for an InnoDB table, the table handle in the InnoDB data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

InnoDB uses the following algorithm to initialize the auto-increment counter for a table t that contains an AUTO_INCREMENT column named ai_col: After a server startup, for the first insert into a table t, InnoDB executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM t FOR UPDATE;
```

InnoDB increments by one the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the table. If the table is empty, InnoDB uses the value 1. If a user invokes a SHOW TABLE STATUS statement that displays output for the table t and the auto-increment counter has not been initialized, InnoDB initializes but does not increment the value and stores it for use by later inserts. This initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

After the auto-increment counter has been initialized, if you do not explicitly specify a value for an AUTO_INCREMENT column, InnoDB increments the counter and assigns the new value to the column. If you insert a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value.

If a user specifies NULL or 0 for the AUTO_INCREMENT column in an INSERT, InnoDB treats the row as if the value had not been specified and generates a new value for it.

The behavior of the auto-increment mechanism is not defined if a user assigns a negative value to the column or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

When accessing the auto-increment counter, InnoDB uses a special table-level AUTO-INC lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an AUTO_INCREMENT column. Nevertheless, two transactions cannot have the AUTO-INC lock on the same table simultaneously, which can have a performance impact if the AUTO-INC lock is held for a long time. That might be the case for a statement such as INSERT INTO t1 ... SELECT ... FROM t2 that inserts all rows from one table into another.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

InnoDB uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, InnoDB reinitializes the counter for each table for the first INSERT to the table, as described earlier.

A server restart also cancels the effect of the AUTO_INCREMENT = *N* table option in CREATE TABLE and ALTER TABLE statements, which you can use with InnoDB tables as of MySQL 5.0.3 to set the initial counter value or alter the current counter value.

You may see gaps in the sequence of values assigned to the AUTO_INCREMENT column if you roll back transactions that have generated numbers using the counter.

14.2.3.4 InnoDB and FOREIGN KEY Constraints

This section describes differences in the InnoDB storage engine's handling of foreign keys as compared with that of the MySQL Server.

Foreign Key Definitions

Foreign key definitions for InnoDB tables are subject to the following conditions:

- InnoDB permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- InnoDB does not currently support foreign keys for tables with user-defined partitioning. This means that no user-partitioned InnoDB table may contain foreign key references or columns referenced by foreign keys.
- InnoDB allows a foreign key constraint to reference a non-unique key. *This is an InnoDB extension to standard SQL.*

Referential Actions

Referential actions for foreign keys of InnoDB tables are subject to the following conditions:

- While SET DEFAULT is allowed by the MySQL Server, it is rejected as invalid by InnoDB. CREATE TABLE and ALTER TABLE statements using this clause are not allowed for InnoDB tables.
- If there are several rows in the parent table that have the same referenced key value, InnoDB acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a RESTRICT type constraint, and there is a child row with several parent rows, InnoDB does not permit the deletion of any of those parent rows.
- InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.
- If ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the *same table* it has previously updated during the cascade, it acts like RESTRICT. This means that you cannot use self-referential ON UPDATE CASCADE or ON UPDATE SET NULL operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, is possible, as is a self-referential ON DELETE CASCADE. Cascading operations may not be nested more than 15 levels deep.
- Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, InnoDB checks UNIQUE and FOREIGN KEY constraints row-by-row. When performing foreign key checks, InnoDB sets shared row-level locks on child or parent records it has to look at. InnoDB checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. Until InnoDB implements deferred constraint checking, some things will be impossible, such as deleting a record that refers to itself using a foreign key.

Foreign Key Usage and Error Information

You can obtain general information about foreign keys and their usage from querying the INFORMATION_SCHEMA.KEY_COLUMN_USAGE table. See also Section 13.1.10.3, “Using FOREIGN KEY Constraints”.

In addition to SHOW ERRORS, in the event of a foreign key error involving InnoDB tables (usually Error 150 in the MySQL Server), you can obtain a detailed explanation of the most recent InnoDB foreign key error by checking the output of SHOW ENGINE INNODB STATUS.

14.2.3.5 InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, you have to make a copy of the InnoDB tablespace and the log files, as well as the .frm files of the InnoDB tables, and move the copies to the slave. If the innodb_file_per_table variable is enabled, you must also copy the .ibd files as well. For the proper procedure to do this, see Section 14.2.6, “Backing Up and Recovering an InnoDB Database”.

If you can shut down the master or an existing slave, you can take a cold backup of the InnoDB tablespace and log files and use that to set up a slave. To make a new slave without taking down any server you can also use the commercial MySQL Enterprise Backup tool.

You cannot set up replication for InnoDB using the LOAD TABLE FROM MASTER statement, which works only for MyISAM tables. There are two possible workarounds:

- Dump the table on the master and import the dump file into the slave.
- Use ALTER TABLE *tbl_name* ENGINE=MyISAM on the master before setting up replication with LOAD TABLE *tbl_name* FROM MASTER, and then use ALTER TABLE to convert the master table back to InnoDB afterward. However, this should not be done for tables that have foreign key definitions because the definitions will be lost.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

Replication and CASCADE. Cascading actions for InnoDB tables on the master are replicated on the slave *only* if the tables sharing the foreign key relation use InnoDB on both the master and slave. Suppose that you have started replication, and then create two tables on the master using the following CREATE TABLE statements:

```
CREATE TABLE fc1 (
    i INT PRIMARY KEY,
    j INT
) ENGINE = InnoDB;

CREATE TABLE fc2 (
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    m INT PRIMARY KEY,
    n INT,
    FOREIGN KEY ni (n) REFERENCES fc1 (i)
        ON DELETE CASCADE
) ENGINE = InnoDB;
```

Suppose that the slave does not have `InnoDB` support enabled. If this is the case, then the tables on the slave are created, but they use the `MyISAM` storage engine, and the `FOREIGN KEY` option is ignored. Now we insert some rows into the tables on the master:

```
master> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2  Duplicates: 0  Warnings: 0

master> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

At this point, on both the master and the slave, table `fc1` contains 2 rows, and table `fc2` contains 3 rows, as shown here:

```
master> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

master> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)

slave> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

slave> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)
```

Now suppose that you perform the following `DELETE` statement on the master:

```
master> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Due to the cascade, table `fc2` on the master now contains only 1 row:

```
master> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 2 | 2 |
+---+---+
1 row in set (0.00 sec)
```

However, the cascade does not propagate on the slave because on the slave the `DELETE` for `fc1` deletes no rows from `fc2`. The slave's copy of `fc2` still contains all of the rows that were originally inserted:

```
slave> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+---+---+
3 rows in set (0.00 sec)
```

This difference is due to the fact that the cascading deletes are handled internally by the `InnoDB` storage engine, which means that none of the changes are logged.

14.2.4 Changing the Number or Size of InnoDB Redo Log Files

To change the number or the size of your `InnoDB` redo log files, perform the following steps:

1. If `innodb_fast_shutdown` is set to 2, set `innodb_fast_shutdown` to 1:

   ```
   mysql> SET GLOBAL innodb_fast_shutdown = 1;
   ```

2. After ensuring that `innodb_fast_shutdown` is not set to 2, stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the log).

3. Copy the old log files into a safe place in case something went wrong during the shutdown and you need them to recover the tablespace.

4. Delete the old log files from the log file directory.

5. Edit `my.cnf` to change the log file configuration.

6. Start the MySQL server again. `mysqld` sees that no `InnoDB` log files exist at startup and creates new ones.

14.2.5 Resizing the InnoDB System Tablespace

This section describes how to increase or decrease the size of the `InnoDB` system tablespace.

Increasing the Size of the InnoDB System Tablespace

The easiest way to increase the size of the `InnoDB` tablespace is to configure it from the beginning to be auto-extending. Specify the `autoextend` attribute for the last data file in the tablespace definition. Then

This documentation is for an older version. If you're
This documentation is for an older version. If you're

InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. The increment size can be changed by setting the value of the innodb_autoextend_increment system variable, which is measured in MB.

You can expand the system tablespace by a defined amount by adding another data file:

1. Shut down the MySQL server.
2. If the previous last data file is defined with the keyword autoextend, change its definition to use a fixed size, based on how large it has actually grown. Check the size of the data file, round it down to the closest multiple of 1024 × 1024 bytes (= 1MB), and specify this rounded size explicitly in innodb_data_file_path.
3. Add a new data file to the end of innodb_data_file_path, optionally making that file auto-extending. Only the last data file in the innodb_data_file_path can be specified as auto-extending.
4. Start the MySQL server again.

For example, this tablespace has just one auto-extending data file ibdata1:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Here is the configuration line after modifying the original data file to not be auto-extending and adding another auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When you add a new file to the tablespace configuration, make sure that it does not exist. InnoDB will create and initialize the file when you restart the server.

Decreasing the Size of the InnoDB System Tablespace

You cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this procedure:

1. Use mysqldump to dump all your InnoDB tables.
2. Stop the server.
3. Remove all the existing tablespace files, including the ibdata and ib_log files. If you want to keep a backup copy of the information, then copy all the ib* files to another location before the removing the files in your MySQL installation.
4. Remove any .frm files for InnoDB tables.
5. Configure a new tablespace.
6. Restart the server.
7. Import the dump files.

14.2.6 Backing Up and Recovering an InnoDB Database

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The key to safe database management is making regular backups.

MySQL Enterprise Backup enables you to back up a running MySQL database, including `InnoDB` and `MyISAM` tables, with minimal disruption to operations while producing a consistent snapshot of the database. When MySQL Enterprise Backup is copying `InnoDB` tables, reads and writes to both `InnoDB` and `MyISAM` tables can continue. During the copying of `MyISAM` tables, reads (but not writes) to those tables are permitted. In addition, MySQL Enterprise Backup supports creating compressed backup files, and performing backups of subsets of `InnoDB` tables. In conjunction with MySQL's binary log, users can perform point-in-time recovery. MySQL Enterprise Backup is commercially licensed. For a more complete description of MySQL Enterprise Backup, see Section 22.2, "MySQL Enterprise Backup Overview".

If you are able to shut down your MySQL server, you can make a binary backup that consists of all files used by `InnoDB` to manage its tables. Use the following procedure:

1. Shut down the MySQL server and make sure that it stops without errors.
2. Copy all `InnoDB` data files (`ibdata` files and `.ibd` files) into a safe place.
3. Copy all the `.frm` files for `InnoDB` tables to a safe place.
4. Copy all `InnoDB` log files (`ib_logfile` files) to a safe place.
5. Copy your `my.cnf` configuration file or files to a safe place.

In addition to making binary backups as just described, you should also regularly make dumps of your tables with `mysqldump`. The reason for this is that a binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. `mysqldump` also has a `--single-transaction` option for making a consistent snapshot without locking out other clients. See Section 7.3.1, "Establishing a Backup Policy".

Replication works with `InnoDB` tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

To be able to recover your `InnoDB` database to the present from the time at which the binary backup was made, you must run your MySQL server with binary logging turned on. To achieve point-in-time recovery after restoring a backup, you can apply changes from the binary log that occurred after the backup was made. See Section 7.5, "Point-in-Time (Incremental) Recovery Using the Binary Log".

To recover from a crash of your MySQL server, the only requirement is to restart it. `InnoDB` automatically checks the logs and performs a roll-forward of the database to the present. `InnoDB` automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, `mysqld` displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database becomes corrupted or disk failure occurs, you must perform the recovery using a backup. In the case of corruption, you should first find a backup that is not corrupted. After restoring the base backup, do a point-in-time recovery from the binary log files using `mysqlbinlog` and `mysql` to restore the changes that occurred after the backup was made.

In some cases of database corruption it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE` SQL statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use the Tablespace Monitor to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. Doing so may eliminate errors that appeared to be database page corruption.

14.2.6.1 The InnoDB Recovery Process

`InnoDB` crash recovery consists of several steps:

- Applying the redo log: Redo log application is the first step and is performed during initialization, before accepting any connections. If all changes were flushed from the buffer pool to the tablespaces (`ibdata*` and `*.ibd` files) at the time of the shutdown or crash, the redo log application can be skipped. If the redo log files are missing at startup, `InnoDB` skips the redo log application.

  Removing redo logs to speed up the recovery process is not recommended, even if some data loss is acceptable. Removing redo logs should only be considered an option after a clean shutdown is performed, with `innodb_fast_shutdown` set to `0` or `1`.

- Rolling back incomplete transactions: Any transactions that were active at the time of crash or fast shutdown. The time it takes to roll back an incomplete transaction can be three or four times the amount of time a transaction is active before it is interrupted, depending on server load.

  You cannot cancel transactions that are in the process of being rolled back. In extreme cases, when rolling back transactions is expected to take an exceptionally long time, it may be faster to start `InnoDB` with an `innodb_force_recovery` setting of `3` or greater. See Section 14.2.6.2, “Forcing InnoDB Recovery” for more information.

- Insert buffer merge: Applying changes from the insert buffer (part of the system tablespace) to leaf pages of secondary indexes, as the index pages are read to the buffer pool.

- Purge: Deleting delete-marked records that are no longer visible for any active transaction.

The steps that follow redo log application do not depend on the redo log (other than for logging the writes) and are performed in parallel with normal processing. Of these, only rollback of incomplete transactions is special to crash recovery. The insert buffer merge and the purge are performed during normal processing.

After redo log application, `InnoDB` attempts to accept connections as early as possible, to reduce downtime. As part of crash recovery, `InnoDB` rolls back any transactions that were not committed or in `XA PREPARE` state when the server crashed. The rollback is performed by a background thread, executed in parallel with transactions from new connections. Until the rollback operation is completed, new connections may encounter locking conflicts with recovered transactions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In most situations, even if the MySQL server was killed unexpectedly in the middle of heavy activity, the recovery process happens automatically and no action is needed from the DBA. If a hardware failure or severe system error corrupted `InnoDB` data, MySQL might refuse to start. In that case, see Section 14.2.6.2, “Forcing InnoDB Recovery” for the steps to troubleshoot such an issue.

For information about the binary log and `InnoDB` crash recovery, see Section 5.4.3, “The Binary Log”.

14.2.6.2 Forcing InnoDB Recovery

If there is database page corruption, you may want to dump your tables from the database with `SELECT ... INTO OUTFILE`. Usually, most of the data obtained in this way is intact. However, it is possible that the corruption might cause `SELECT * FROM tbl_name` statements or `InnoDB` background operations to crash or assert, or even cause `InnoDB` roll-forward recovery to crash. In such cases, you can use the `innodb_force_recovery` option to force the `InnoDB` storage engine to start up while preventing background operations from running, so that you are able to dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 1
```

Warning

Only set `innodb_force_recovery` to a value greater than 0 in an emergency situation, so that you can start `InnoDB` and dump your tables. Before doing so, ensure that you have a backup copy of your database in case you need to recreate it. Values of 4 or greater can permanently corrupt data files. Only use an `innodb_force_recovery` setting of 4 or greater on a production server instance after you have successfully tested the setting on separate physical copy of your database. When forcing `InnoDB` recovery, you should always start with `innodb_force_recovery=1` and only increase the value incrementally, as necessary.

`innodb_force_recovery` is 0 by default (normal startup without forced recovery). The permissible nonzero values for `innodb_force_recovery` are 1 to 6. A larger value includes the functionality of lesser values. For example, a value of 3 includes all of the functionality of values 1 and 2.

If you are able to dump your tables with an `innodb_force_recovery` value of 3 or less, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 4 or greater is considered dangerous because data files can be permanently corrupted. A value of 6 is considered drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

As a safety measure, `InnoDB` prevents users from performing `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0.

- `1` (`SRV_FORCE_IGNORE_CORRUPT`)

  Let the server run even if it detects a corrupt page. Try to make `SELECT * FROM tbl_name` jump over corrupt index records and pages, which helps in dumping tables.

- `2` (`SRV_FORCE_NO_BACKGROUND`)

  Prevent the main thread from running. If a crash would occur during the purge operation, this recovery value prevents it.

- `3` (`SRV_FORCE_NO_TRX_UNDO`)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Do not run transaction rollbacks after recovery.

- `4` (`SRV_FORCE_NO_IBUF_MERGE`)

  Prevent insert buffer merge operations. If they would cause a crash, do not do them. Do not calculate table statistics. This value can permanently corrupt data files. After using this value, be prepared to drop and recreate all secondary indexes.

- `5` (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

  Do not look at undo logs when starting the database: `InnoDB` treats even incomplete transactions as committed. This value can permanently corrupt data files.

- `6` (`SRV_FORCE_NO_LOG_REDO`)

  Do not do the log roll-forward in connection with recovery. This value can permanently corrupt data files. Leaves database pages in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

You can `SELECT` from tables to dump them, or `DROP` or `CREATE` tables even if forced recovery is used. If you know that a given table is causing a crash on rollback, you can drop it. You can also use this to stop a runaway rollback caused by a failing mass import or `ALTER TABLE`. You can kill the `mysqld` process and set `innodb_force_recovery` to `3` to bring the database up without the rollback, then `DROP` the table that is causing the runaway rollback.

14.2.6.3 InnoDB Checkpoints

`InnoDB` implements a checkpoint mechanism known as “fuzzy” checkpointing. `InnoDB` flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would in practice stop processing of user SQL statements during the checkpointing process.

During crash recovery, `InnoDB` looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then `InnoDB` scans the log files forward from the checkpoint, applying the logged modifications to the database.

`InnoDB` writes to its log files on a rotating basis. It also writes checkpoint information to the first log file at each checkpoint. All committed modifications that make the database pages in the buffer pool different from the images on disk must be available in the log files in case `InnoDB` has to do a recovery. This means that when `InnoDB` starts to reuse a log file, it has to make sure that the database page images on disk contain the modifications logged in the log file that `InnoDB` is going to reuse. In other words, `InnoDB` must create a checkpoint and this often involves flushing of modified database pages to disk.

The preceding description explains why making your log files very large may reduce disk I/O in checkpointing. It often makes sense to set the total size of the log files as large as the buffer pool or even larger. The disadvantage of using large log files is that crash recovery can take longer because there is more logged information to apply to the database.

14.2.7 Moving an InnoDB Database to Another Machine

On Windows, `InnoDB` always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names. A convenient way to accomplish this is to add the following line to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating any databases or tables:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
[mysqld]
lower_case_table_names=1
```

Like MyISAM data files, InnoDB data and log files are binary-compatible on all platforms having the same floating-point number format. You can move an InnoDB database simply by copying all the relevant files listed in Section 14.2.6, “Backing Up and Recovering an InnoDB Database”. If the floating-point formats differ but you have not used FLOAT or DOUBLE data types in your tables, then the procedure is the same: simply copy the relevant files. If you use mysqldump to dump your tables on one machine and then import the dump files on the other machine, it does not matter whether the formats differ or your tables contain floating-point data.

One way to increase performance is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

14.2.8 InnoDB Transaction Model and Locking

To implement a large-scale, busy, or highly reliable database application, to port substantial code from a different database system, or to tune MySQL performance, you must understand the notions of transactions and locking as they relate to the InnoDB storage engine.

In the InnoDB transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. InnoDB does locking on the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle. The lock information in InnoDB is stored so space-efficiently that lock escalation is not needed: Typically, several users are permitted to lock every row in InnoDB tables, or any random subset of the rows, without causing InnoDB memory exhaustion.

In InnoDB, all user activity occurs inside a transaction. If autocommit mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts the session for each new connection with autocommit enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 14.2.12, “InnoDB Error Handling”.

A session that has autocommit enabled can perform a multiple-statement transaction by starting it with an explicit START TRANSACTION or BEGIN statement and ending it with a COMMIT or ROLLBACK statement. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

If autocommit mode is disabled within a session with SET autocommit = 0, the session always has a transaction open. A COMMIT or ROLLBACK statement ends the current transaction and a new one starts.

A COMMIT means that the changes made in the current transaction are made permanent and become visible to other sessions. A ROLLBACK statement, on the other hand, cancels all modifications made by the current transaction. Both COMMIT and ROLLBACK release all InnoDB locks that were set during the current transaction.

In terms of the SQL:1992 transaction isolation levels, the default InnoDB level is REPEATABLE READ. InnoDB offers all four transaction isolation levels described by the SQL standard: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE.

A user can change the isolation level for a single session or for all subsequent connections with the SET TRANSACTION statement. To set the server's default isolation level for all connections, use the --transaction-isolation option on the command line or in an option file. For detailed information about isolation levels and level-setting syntax, see Section 13.3.6, “SET TRANSACTION Syntax”.

In row-level locking, InnoDB normally uses next-key locking. That means that besides index records, InnoDB can also lock the “gap” preceding an index record to block insertions by other sessions in the gap

This documentation is for an older version. If you're
This documentation is for an older version. If you're

immediately before the index record. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that locks only the gap before some index record.

For more information about row-level locking, and the circumstances under which gap locking is disabled, see Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”.

14.2.8.1 InnoDB Lock Modes

`InnoDB` implements standard row-level locking where there are two types of locks, shared (*S*) locks and exclusive (*X*) locks. For information about record, gap, and next-key lock types, see Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”.

- A shared (*S*) lock permits a transaction to read a row.
- An exclusive (*X*) lock permits a transaction to update or delete a row.

If transaction `T1` holds a shared (*S*) lock on row `r`, then requests from some distinct transaction `T2` for a lock on row `r` are handled as follows:

- A request by `T2` for an *S* lock can be granted immediately. As a result, both `T1` and `T2` hold an *S* lock on `r`.
- A request by `T2` for an *X* lock cannot be granted immediately.

If a transaction `T1` holds an exclusive (*X*) lock on row `r`, a request from some distinct transaction `T2` for a lock of either type on `r` cannot be granted immediately. Instead, transaction `T2` has to wait for transaction `T1` to release its lock on row `r`.

Intention Locks

Additionally, `InnoDB` supports *multiple granularity locking* which permits coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called *intention locks* are used. Intention locks are table locks in `InnoDB`. The idea behind intention locks is for a transaction to indicate which type of lock (shared or exclusive) it will require later for a row in that table. There are two types of intention locks used in `InnoDB` (assume that transaction `T` has requested a lock of the indicated type on table `t`):

- Intention shared (*IS*): Transaction `T` intends to set *S* locks on individual rows in table `t`.
- Intention exclusive (*IX*): Transaction `T` intends to set *X* locks on those rows.

For example, `SELECT ... LOCK IN SHARE MODE` sets an *IS* lock and `SELECT ... FOR UPDATE` sets an *IX* lock.

The intention locking protocol is as follows:

- Before a transaction can acquire an *S* lock on a row in table `t`, it must first acquire an *IS* or stronger lock on `t`.
- Before a transaction can acquire an *X* lock on a row, it must first acquire an *IX* lock on `t`.

These rules can be conveniently summarized by means of the following *lock type compatibility matrix*.

	X	***IX***	***S***	***IS***
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	X	*IX*	*S*	*IS*
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks, but not if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Thus, intention locks do not block anything except full table requests (for example, `LOCK TABLES ... WRITE`). The main purpose of *IX* and *IS* locks is to show that someone is locking a row, or going to lock a row in the table.

Deadlock Example

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an *S* lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+------+
| i    |
+------+
|    1 |
+------+
1 row in set (0.10 sec)
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an *X* lock. The lock cannot be granted because it is incompatible with the *S* lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an *X* lock to delete the row. However, that lock request cannot be granted because client B already has a request for an *X* lock and is waiting for client A to release its *S* lock. Nor can the *S* lock held by A be upgraded to an *X* lock because of the prior request by B for an *X*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

lock. As a result, InnoDB generates an error for one of the clients and releases its locks. The client returns this error:

```
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

At that point, the lock request for the other client can be granted and it deletes the row from the table.

Note

If the LATEST DETECTED DEADLOCK section of InnoDB Monitor output includes a message stating, "TOO DEEP OR LONG SEARCH IN THE LOCK TABLE WAITS-FOR GRAPH, WE WILL ROLL BACK FOLLOWING TRANSACTION," this indicates that the number of transactions on the wait-for list has reached a limit of 200, which is defined by LOCK_MAX_DEPTH_IN_DEADLOCK_CHECK. A wait-for list that exceeds 200 transactions is treated as a deadlock and the transaction attempting to check the wait-for list is rolled back.

The same error may also occur if the locking thread must look at more than 1,000,000 locks owned by the transactions on the wait-for list. The limit of 1,000,000 locks is defined by LOCK_MAX_N_STEPS_IN_DEADLOCK_CHECK.

14.2.8.2 InnoDB Record, Gap, and Next-Key Locks

InnoDB has several types of record-level locks including record locks, gap locks, and next-key locks. For information about shared locks, exclusive locks, and intention locks, see Section 14.2.8.1, "InnoDB Lock Modes".

- Record lock: This is a lock on an index record.
- Gap lock: This is a lock on a gap between index records, or a lock on the gap before the first or after the last index record.
- Next-key lock: This is a combination of a record lock on the index record and a gap lock on the gap before the index record.

Record Locks

Record locks always lock index records, even if a table is defined with no indexes. For such cases, InnoDB creates a hidden clustered index and uses this index for record locking. See Section 14.2.10.1, "Clustered and Secondary Indexes".

Next-key Locks

By default, InnoDB operates in REPEATABLE READ transaction isolation level and with the innodb_locks_unsafe_for_binlog system variable disabled. In this case, InnoDB uses next-key locks for searches and index scans, which prevents phantom rows (see Section 14.2.8.3, "Avoiding the Phantom Problem Using Next-Key Locking").

Next-key locking combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the "gap" before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Suppose that an index contains the values 10, 11, 13, and 20. The possible next-key locks for this index cover the following intervals, where (or) denote exclusion of the interval endpoint and [or] denote inclusion of the endpoint:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

For the last interval, the next-key lock locks the gap above the largest value in the index and the "supremum" pseudo-record having a value higher than any value actually in the index. The supremum is not a real index record, so, in effect, this next-key lock locks only the gap following the largest index value.

Gap Locks

The next-key locking example in the previous section shows that a gap might span a single index value, multiple index values, or even be empty.

Gap locking is not needed for statements that lock rows using a unique index to search for a unique row. (This does not include the case that the search condition includes only some columns of a multiple-column unique index; in that case, gap locking does occur.) For example, if the `id` column has a unique index, the following statement uses only an index-record lock for the row having `id` value 100 and it does not matter whether other sessions insert rows in the preceding gap:

```
SELECT * FROM child WHERE id = 100;
```

If `id` is not indexed or has a nonunique index, the statement does lock the preceding gap.

A type of gap lock called an insert intention gap lock is set by `INSERT` operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

It is also worth noting here that conflicting locks can be held on a gap by different transactions. For example, transaction A can hold a shared gap lock (gap S-lock) on a gap while transaction B holds an exclusive gap lock (gap X-lock) on the same gap. The reason conflicting gap locks are allowed is that if a record is purged from an index, the gap locks held on the record by different transactions must be merged.

Gap locks in `InnoDB` are "purely inhibitive", which means they only stop other transactions from inserting to the gap. Thus, a gap X-lock has the same effect as a gap S-lock.

Disabling Gap Locking

Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to `READ COMMITTED` or enable the `innodb_locks_unsafe_for_binlog` system variable. Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

There is also another effect of using the `READ COMMITTED` isolation level or enabling `innodb_locks_unsafe_for_binlog`: Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition.

14.2.8.3 Avoiding the Phantom Problem Using Next-Key Locking

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The so-called *phantom* problem occurs within a transaction when the same query produces different sets of rows at different times. For example, if a `SELECT` is executed twice, but returns a row the second time that was not returned the first time, the row is a “phantom” row.

Suppose that there is an index on the `id` column of the `child` table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where `id` is bigger than 100. Let the table contain rows having `id` values of 90 and 102. If the locks set on the index records in the scanned range do not lock out inserts made in the gaps (in this case, the gap between 90 and 102), another session can insert a new row into the table with an `id` of 101. If you were to execute the same `SELECT` within the same transaction, you would see a new row with an `id` of 101 (a “phantom”) in the result set returned by the query. If we regard a set of rows as a data item, the new phantom child would violate the isolation principle of transactions that a transaction should be able to run so that the data it has read does not change during the transaction.

To prevent phantoms, `InnoDB` uses an algorithm called *next-key locking* that combines index-row locking with gap locking. `InnoDB` performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order.

When `InnoDB` scans an index, it can also lock the gap after the last record in the index. Just that happens in the preceding example: To prevent any insert into the table where `id` would be bigger than 100, the locks set by `InnoDB` include a lock on the gap following `id` value 102.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking enables you to “lock” the nonexistence of something in your table.

Gap locking can be disabled as discussed in Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”. This may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled.

14.2.8.4 Consistent Nonlocking Reads

A consistent read means that `InnoDB` uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. This exception causes the following anomaly: If you update some rows in a table, a `SELECT` sees the latest version of the updated rows, but it might also see older versions of any rows. If other sessions simultaneously update the same table, the anomaly means that you might see the table in a state that never existed in the database.

If the transaction isolation level is `REPEATABLE READ` (the default level), all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

With `READ COMMITTED` isolation level, each consistent read within a transaction sets and reads its own fresh snapshot.

Consistent read is the default mode in which `InnoDB` processes `SELECT` statements in `READ COMMITTED` and `REPEATABLE READ` isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other sessions are free to modify those tables at the same time a consistent read is being performed on the table.

Suppose that you are running in the default `REPEATABLE READ` isolation level. When you issue a consistent read (that is, an ordinary `SELECT` statement), `InnoDB` gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

You can advance your timepoint by committing your transaction and then doing another `SELECT` or `START TRANSACTION WITH CONSISTENT SNAPSHOT`.

This is called *multi-versioned concurrency control*.

In the following example, session A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

```
             Session A              Session B

           SET autocommit=0;      SET autocommit=0;
time
|          SELECT * FROM t;
|          empty set
|                                 INSERT INTO t VALUES (1, 2);
|
v          SELECT * FROM t;
           empty set
                                  COMMIT;

           SELECT * FROM t;
           empty set

           COMMIT;

           SELECT * FROM t;
           ---------------------
           |    1    |    2    |
           ---------------------
           1 row in set
```

If you want to see the “freshest” state of the database, you should use either the `READ COMMITTED` isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

With `READ COMMITTED` isolation level, each consistent read within a transaction sets and reads its own fresh snapshot. With `LOCK IN SHARE MODE`, a locking read occurs instead: A `SELECT` blocks until the transaction containing the freshest rows ends (see Section 14.2.8.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”).

Consistent read does not work over certain DDL statements:

- Consistent read does not work over `DROP TABLE`, because MySQL cannot use a table that has been dropped and `InnoDB` destroys the table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Consistent read does not work over `ALTER TABLE`, because that statement makes a temporary copy of the original table and deletes the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken.

`InnoDB` uses a consistent read for select in clauses like `INSERT INTO ... SELECT`, `UPDATE ... (SELECT)`, and `CREATE TABLE ... SELECT` that do not specify `FOR UPDATE` or `LOCK IN SHARE MODE` if the `innodb_locks_unsafe_for_binlog` option is set and the isolation level of the transaction is not set to `SERIALIZABLE`. Thus, no locks are set on rows read from the selected table. Otherwise, `InnoDB` uses stronger locks and the `SELECT` part acts like `READ COMMITTED`, where each consistent read, even within the same transaction, sets and reads its own fresh snapshot.

14.2.8.5 SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads

In some circumstances, a consistent (nonlocking) read is not convenient and a locking read is required instead. `InnoDB` supports two types of locking reads:

- `SELECT ... LOCK IN SHARE MODE` sets a shared mode lock on any rows that are read. Other sessions can read the rows, but cannot modify them until your transaction commits. If any of these rows were changed by another transaction that has not yet committed, your query waits until that transaction ends and then uses the latest values.

- For index records the search encounters, `SELECT ... FOR UPDATE` blocks other sessions from doing `SELECT ... LOCK IN SHARE MODE` or from reading in certain transaction isolation levels. Consistent reads will ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they will be reconstructed by applying undo logs on an in-memory copy of the record.)

These clauses are primarily useful when dealing with tree-structured or graph-structured data, either in a single table or split across multiple tables.

Locks set by `LOCK IN SHARE MODE` and `FOR UPDATE` reads are released when the transaction is committed or rolled back.

As an example of a situation in which a locking read is useful, suppose that you want to insert a new row into a table `child`, and make sure that the child row has a parent row in table `parent`. The following discussion describes how to implement referential integrity in application code.

Suppose that you use a consistent read to read the table `parent` and indeed see the parent row of the to-be-inserted child row in the table. Can you safely insert the child row to table `child`? No, because it is possible for some other session to delete the parent row from the table `parent` in the meantime without you being aware of it.

The solution is to perform the `SELECT` in a locking mode using `LOCK IN SHARE MODE`:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

A read performed with `LOCK IN SHARE MODE` reads the latest available data and sets a shared mode lock on the rows read. A shared mode lock prevents others from updating or deleting the row read. Also, if the latest data belongs to a yet uncommitted transaction of another session, we wait until that transaction ends. After we see that the `LOCK IN SHARE MODE` query returns the parent `'Jones'`, we can safely add the child record to the `child` table and commit our transaction.

Let us look at another example: We have an integer counter field in a table `child_codes` that we use to assign a unique identifier to each child added to table `child`. It is not a good idea to use either consistent read or a shared mode read to read the present value of the counter because two users of the database

This documentation is for an older version. If you're
This documentation is for an older version. If you're

may then see the same value for the counter, and a duplicate-key error occurs if two users attempt to add children with the same identifier to the table.

Here, `LOCK IN SHARE MODE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when it attempts to update the counter.

To implement reading and incrementing the counter, first perform a locking read of the counter using `FOR UPDATE`, and then increment the counter. For example:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how `SELECT ... FOR UPDATE` works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The `SELECT` statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

Note

Locking of rows for update using `SELECT FOR UPDATE` only applies when autocommit is disabled (either by beginning transaction with `START TRANSACTION` or by setting `autocommit` to 0. If autocommit is enabled, the rows matching the specification are not locked.

14.2.8.6 Locks Set by Different SQL Statements in InnoDB

A locking read, an `UPDATE`, or a `DELETE` generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter whether there are `WHERE` conditions in the statement that would exclude the row. `InnoDB` does not remember the exact `WHERE` condition, but only knows which index ranges were scanned. The locks are normally next-key locks that also block inserts into the “gap” immediately before the record. However, gap locking can be disabled explicitly, which causes next-key locking not to be used. For more information, see Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”. The transaction isolation level also can affect which locks are set; see Section 13.3.6, “SET TRANSACTION Syntax”.

If a secondary index is used in a search and index record locks to be set are exclusive, `InnoDB` also retrieves the corresponding clustered index records and sets locks on them.

Differences between shared and exclusive locks are described in Section 14.2.8.1, “InnoDB Lock Modes”.

If you have no indexes suitable for your statement and MySQL must scan the entire table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily scan many rows.

For `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`, locks are acquired for scanned rows, and expected to be released for rows that do not qualify for inclusion in the result set (for example, if they do not meet the criteria given in the `WHERE` clause). However, in some cases, rows might not be unlocked immediately because the relationship between a result row and its original source is lost during query execution. For example, in a `UNION`, scanned (and locked) rows from a table might be inserted

This documentation is for an older version. If you're
This documentation is for an older version. If you're

into a temporary table before evaluation whether they qualify for the result set. In this circumstance, the relationship of the rows in the temporary table to the rows in the original table is lost and the latter rows are not unlocked until the end of query execution.

`InnoDB` sets specific types of locks as follows.

- `SELECT ... FROM` is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to `SERIALIZABLE`. For `SERIALIZABLE` level, the search sets shared next-key locks on the index records it encounters.
- `SELECT ... FROM ... LOCK IN SHARE MODE` sets shared next-key locks on all index records the search encounters.
- For index records the search encounters, `SELECT ... FROM ... FOR UPDATE` blocks other sessions from doing `SELECT ... FROM ... LOCK IN SHARE MODE` or from reading in certain transaction isolation levels. Consistent reads will ignore any locks set on the records that exist in the read view.
- `UPDATE ... WHERE ...` sets an exclusive next-key lock on every record the search encounters.
- `DELETE FROM ... WHERE ...` sets an exclusive next-key lock on every record the search encounters.
- `INSERT` sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

  Prior to inserting the row, a type of gap lock called an insertion intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

  If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an `InnoDB` table `t1` has the following structure:

  ```
  CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
  ```

  Now suppose that three sessions perform the following operations in order:

  Session 1:

  ```
  START TRANSACTION;
  INSERT INTO t1 VALUES(1);
  ```

  Session 2:

  ```
  START TRANSACTION;
  INSERT INTO t1 VALUES(1);
  ```

  Session 3:

  ```
  START TRANSACTION;
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value 1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

- `INSERT ... ON DUPLICATE KEY UPDATE` differs from a simple `INSERT` in that an exclusive next-key lock rather than a shared lock is placed on the row to be updated when a duplicate-key error occurs.
- `REPLACE` is done like an `INSERT` if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.
- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive index record without a gap lock on each row inserted into `T`. If `innodb_locks_unsafe_for_binlog` is enabled and the transaction isolation level is not `SERIALIZABLE`, `InnoDB` does the search on `S` as a consistent read (no locks). Otherwise, `InnoDB` sets shared next-key locks on rows from `S`. `InnoDB` has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement must be executed in exactly the same way it was done originally.

  `CREATE TABLE ... SELECT ...` performs the `SELECT` with shared next-key locks or as a consistent read, as for `INSERT ... SELECT`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For `REPLACE INTO T SELECT ... FROM S WHERE ...`, `InnoDB` sets shared next-key locks on rows from `S`.

- While initializing a previously specified `AUTO_INCREMENT` column on a table, `InnoDB` sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. In accessing the auto-increment counter, `InnoDB` uses a specific `AUTO-INC` table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other sessions cannot insert into the table while the `AUTO-INC` table lock is held; see Section 14.2.8, “InnoDB Transaction Model and Locking”.

  `InnoDB` fetches the value of a previously initialized `AUTO_INCREMENT` column without setting any locks.

- If a `FOREIGN KEY` constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. `InnoDB` also sets these locks in the case where the constraint fails.

- `LOCK TABLES` sets table locks, but it is the higher MySQL layer above the `InnoDB` layer that sets these locks. `InnoDB` is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0`, and the MySQL layer above `InnoDB` knows about row-level locks.

  Otherwise, `InnoDB`'s automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because in this case the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another session currently has row-level locks. However, this does not endanger transaction integrity, as discussed in Section 14.2.8.8, “Deadlock Detection and Rollback”. See also Section 14.2.14, “Limits on InnoDB Tables”.

14.2.8.7 Implicit Transaction Commit and Rollback

By default, MySQL starts the session for each new connection with autocommit mode enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 14.2.12, “InnoDB Error Handling”.

If a session that has autocommit disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

Some statements implicitly end a transaction, as if you had done a `COMMIT` before executing the statement. For details, see Section 13.3.3, “Statements That Cause an Implicit Commit”.

14.2.8.8 Deadlock Detection and Rollback

`InnoDB` automatically detects transaction deadlocks and rolls back a transaction or transactions to break the deadlock. `InnoDB` tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

`InnoDB` is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0`, and the MySQL layer above it knows about row-level locks. Otherwise, `InnoDB` cannot detect deadlocks where a table lock set by a MySQL `LOCK TABLES` statement or a lock set by a storage engine other than `InnoDB` is involved. Resolve these situations by setting the value of the `innodb_lock_wait_timeout` system variable.

When `InnoDB` performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because `InnoDB` stores row locks in a format such that it cannot know afterward which lock was set by which statement.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

14.2.8.9 How to Cope with Deadlocks

This section builds on the conceptual information about deadlocks in Section 14.2.8.8, “Deadlock Detection and Rollback”. It explains how to organize database operations to minimize deadlocks and the subsequent error handling required in applications.

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

`InnoDB` uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really “atomic”; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- Use `SHOW ENGINE INNODB STATUS` to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Keep transactions small and short in duration to make them less prone to collision.
- Commit transactions immediately after making a set of related changes to make them less prone to collision. In particular, do not leave an interactive `mysql` session open for a long time with an uncommitted transaction.
- If you are using locking reads (`SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`), try using a lower isolation level such as `READ COMMITTED`.
- When modifying multiple tables within a transaction, or different sets of rows in the same table, do those operations in a consistent order each time. Then transactions form well-defined queues and do not deadlock. For example, organize database operations into functions within your application, or call stored routines, rather than coding multiple similar sequences of `INSERT`, `UPDATE`, and `DELETE` statements in different places.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use `EXPLAIN SELECT` to determine which indexes the MySQL server regards as the most appropriate for your queries.
- Use less locking. If you can afford to permit a `SELECT` to return data from an old snapshot, do not add the clause `FOR UPDATE` or `LOCK IN SHARE MODE` to it. Using the `READ COMMITTED` isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot. You should also set the value of `innodb_support_xa` to 0, which will reduce the number of disk flushes due to synchronizing on disk data and the binary log.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use `LOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
COMMIT;
UNLOCK TABLES;
```

Table-level locks prevent concurrent updates to the table, avoiding deadlocks at the expense of less responsiveness for a busy system.

- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the `InnoDB` instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

14.2.9 InnoDB Multi-Versioning

Because `InnoDB` is a multi-versioned storage engine, it must keep information about old versions of rows in the tablespace. This information is stored in a data structure called a *rollback segment* (after an analogous data structure in Oracle).

Internally, `InnoDB` adds three fields to each row stored in the database. A 6-byte `DB_TRX_ID` field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte `DB_ROLL_PTR` field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated. A 6-byte `DB_ROW_ID` field contains a row ID that increases monotonically as new rows are inserted. If `InnoDB` generates a clustered index automatically, the index contains row ID values. Otherwise, the `DB_ROW_ID` column does not appear in any index.

`InnoDB` uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which `InnoDB` has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

Commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, `InnoDB` cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space needed for your rollback segment.

In the `InnoDB` multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. `InnoDB` only physically removes the corresponding row and its index records when it discards the update undo log record written for the deletion. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

If you insert and delete rows in smallish batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow. In such a case, throttle new row operations, and allocate more resources to the purge thread by tuning the `innodb_max_purge_lag` system variable. See Section 14.2.2, “InnoDB Startup Options and System Variables” for more information.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Multi-Versioning and Secondary Indexes

`InnoDB` multiversion concurrency control (MVCC) treats secondary indexes differently than clustered indexes. Records in a clustered index are updated in-place, and their hidden system columns point undo log entries from which earlier versions of records can be reconstructed. Unlike clustered index records, secondary index records do not contain hidden system columns nor are they updated in-place.

When a secondary index column is updated, old secondary index records are delete-marked, new records are inserted, and delete-marked records are eventually purged. When a secondary index record is delete-marked or the secondary index page is updated by a newer transaction, `InnoDB` looks up the database record in the clustered index. In the clustered index, the record's `DB_TRX_ID` is checked, and the correct version of the record is retrieved from the undo log if the record was modified after the reading transaction was initiated.

If a secondary index record is marked for deletion or the secondary index page is updated by a newer transaction, the covering index technique is not used. Instead of returning values from the index structure, `InnoDB` looks up the record in the clustered index.

14.2.10 InnoDB Table and Index Structures

Role of the .frm File

MySQL stores its data dictionary information for tables in `.frm` files in database directories. This is true for all MySQL storage engines, but every `InnoDB` table also has its own entry in the `InnoDB` internal data dictionary inside the tablespace. When MySQL drops a table or a database, it has to delete one or more `.frm` files as well as the corresponding entries inside the `InnoDB` data dictionary. Consequently, you cannot move `InnoDB` tables between databases simply by moving the `.frm` files.

14.2.10.1 Clustered and Secondary Indexes

Every `InnoDB` table has a special index called the *clustered index* where the data for the rows is stored. Typically, the clustered index is synonymous with the primary key. To get the best performance from queries, inserts, and other database operations, you must understand how InnoDB uses the clustered index to optimize the most common lookup and DML operations for each table.

- If you define a `PRIMARY KEY` on your table, `InnoDB` uses it as the clustered index.
- If you do not define a `PRIMARY KEY` for your table, MySQL picks the first `UNIQUE` index that has only `NOT NULL` columns as the primary key and `InnoDB` uses it as the clustered index.
- If the table has no `PRIMARY KEY` or suitable `UNIQUE` index, `InnoDB` internally generates a hidden clustered index on a synthetic column containing row ID values. The rows are ordered by the ID that `InnoDB` assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

How the Clustered Index Speeds Up Queries

Accessing a row through the clustered index is fast because the row data is on the same page where the index search leads. If a table is large, the clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record. (For example, `MyISAM` uses one file for data rows and another for index records.)

How Secondary Indexes Relate to the Clustered Index

All indexes other than the clustered index are known as secondary indexes. In `InnoDB`, each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. `InnoDB` uses this primary key value to search for the row in the clustered index.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

14.2.10.2 Physical Structure of an InnoDB Index

All InnoDB indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, InnoDB tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fill factor of an index page drops below 1/2, InnoDB tries to contract the index tree to free the page.

Note

Changing the page size is not a supported operation and there is no guarantee that InnoDB will function normally with a page size other than 16KB. Problems compiling or running InnoDB may occur.

A version of InnoDB built for one page size cannot use data files or log files from a version built for a different page size.

14.2.10.3 Insert Buffering

It is a common situation in database applications that the primary key is a unique identifier and new rows are inserted in the ascending order of the primary key. Thus, insertions into the clustered index do not require random reads from a disk.

On the other hand, secondary indexes are usually nonunique, and insertions into secondary indexes happen in a relatively random order. This would cause a lot of random disk I/O operations without a special mechanism used in InnoDB.

If an index record should be inserted into a nonunique secondary index, InnoDB checks whether the secondary index page is in the buffer pool. If that is the case, InnoDB does the insertion directly to the index page. If the index page is not found in the buffer pool, InnoDB inserts the record to a special insert buffer structure. The insert buffer is kept so small that it fits entirely in the buffer pool, and insertions can be done very fast.

Periodically, the insert buffer is merged into the secondary index trees in the database. Often it is possible to merge several insertions into the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer can speed up insertions into a table up to 15 times.

The insert buffer merging may continue to happen *after* the transaction has been committed. In fact, it may continue to happen after a server shutdown and restart (see Section 14.2.6.2, “Forcing InnoDB Recovery”).

Insert buffer merging may take many hours when many secondary indexes must be updated and many rows have been inserted. During this time, disk I/O will be increased, which can cause significant slowdown on disk-bound queries. Another significant background I/O operation is the purge thread (see Section 14.2.9, “InnoDB Multi-Versioning”).

14.2.10.4 Adaptive Hash Indexes

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes. InnoDB has a mechanism that monitors index searches made to the indexes defined for a table. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The hash index is always built based on an existing B-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that InnoDB observes for the B-tree index. A hash index can be partial: It is not required that the whole B-tree index is cached in the buffer pool. InnoDB builds hash indexes on demand for those pages of the index that are often accessed.

In a sense, InnoDB tailors itself through the adaptive hash index mechanism to ample main memory, coming closer to the architecture of main-memory databases.

14.2.10.5 Physical Row Structure

The physical row structure for an InnoDB table depends on the MySQL version and the optional ROW_FORMAT option used when the table was created. For InnoDB tables in MySQL 5.0.3 and earlier, only the REDUNDANT row format was available. For MySQL 5.0.3 and later, the default is to use the COMPACT row format, but you can use the REDUNDANT format to retain compatibility with older versions of InnoDB tables. To check the row format of an InnoDB table use SHOW TABLE STATUS.

The compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed, compact format is likely to be faster. If the workload is a rare case that is limited by CPU speed, compact format might be slower.

Rows in InnoDB tables that use REDUNDANT row format have the following characteristics:

- Each index record contains a 6-byte header. The header is used to link together consecutive records, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index.
- A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as CHAR(10) in a fixed-length format. Before MySQL 5.0.3, InnoDB truncates trailing spaces from VARCHAR columns.
- An SQL NULL value reserves one or two bytes in the record directory. Besides that, an SQL NULL value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column in the data part of the record. Reserving the fixed space for NULL values enables an update of the column from NULL to a non-NULL value to be done in place without causing fragmentation of the index page.

Rows in InnoDB tables that use COMPACT row format have the following characteristics:

- Each index record contains a 5-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and also in row-level locking.
- The variable-length part of the record header contains a bit vector for indicating NULL columns. If the number of columns in the index that can be NULL is *N*, the bit vector occupies CEILING(*N*/8) bytes. (For example, if there are anywhere from 9 to 15 columns that can be NULL, the bit vector uses two bytes.) Columns that are NULL do not occupy space other than the bit in this vector. The variable-length

This documentation is for an older version. If you're
This documentation is for an older version. If you're

part of the header also contains the lengths of variable-length columns. Each length takes one or two bytes, depending on the maximum length of the column. If all columns in the index are `NOT NULL` and have a fixed length, the record header has no variable-length part.

- For each non-`NULL` variable-length field, the record header contains the length of the column in one or two bytes. Two bytes will only be needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes. For an externally stored column, the 2-byte length indicates the length of the internally stored part plus the 20-byte pointer to the externally stored part. The internal part is 768 bytes, so the length is 768+20. The 20-byte pointer stores the true length of the column.
- The record header is followed by the data contents of the non-`NULL` columns.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index. If any of these primary key fields are variable length, the record header for each secondary index will have a variable-length part to record their lengths, even if the secondary index is defined on fixed-length columns.
- Internally, `InnoDB` stores fixed-length, fixed-width character columns such as `CHAR(10)` in a fixed-length format. Before MySQL 5.0.3, `InnoDB` truncates trailing spaces from `VARCHAR` columns.
- Internally, `InnoDB` attempts to store UTF-8 `CHAR(N)` columns in *`N`* bytes by trimming trailing spaces. (With `REDUNDANT` row format, such columns occupy 3 × *`N`* bytes.) Reserving the minimum space *`N`* in many cases enables column updates to be done in place without causing fragmentation of the index page.

14.2.11 InnoDB Disk I/O and File Space Management

14.2.11.1 InnoDB Disk I/O

`InnoDB` uses simulated asynchronous disk I/O: `InnoDB` creates a number of threads to take care of I/O operations, such as read-ahead.

There are two read-ahead heuristics in `InnoDB`:

- In sequential read-ahead, if `InnoDB` notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if `InnoDB` notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

`InnoDB` uses a novel file flush technique involving a structure called the doublewrite buffer, which is enabled by default (`innodb_doublewrite=ON`). It adds safety to recovery following a crash or power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations.

Doublewrite means that before writing pages to a data file, `InnoDB` first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does `InnoDB` write the pages to their proper positions in the data file. If there is an operating system, storage subsystem, or `mysqld` process crash in the middle of a page write (causing a torn page condition), `InnoDB` can later find a good copy of the page from the doublewrite buffer during recovery.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

14.2.11.2 File Space Management

The data files that you define in the configuration file form the `InnoDB` tablespace. The files are logically concatenated to form the tablespace. There is no striping in use. You cannot define where within the tablespace your tables are allocated. However, in a newly created tablespace, `InnoDB` allocates space starting from the first data file.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of size 1MB (64 consecutive pages). The "files" inside a tablespace are called *segments* in `InnoDB`. The term "rollback segment" is somewhat confusing because it actually contains many tablespace segments.

When a segment grows inside the tablespace, `InnoDB` allocates the first 32 pages to it individually. After that, `InnoDB` starts to allocate whole extents to the segment. `InnoDB` can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in `InnoDB`. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. The idea here is to achieve better sequentiality for the leaf nodes, which contain the data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an `InnoDB` tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a `SHOW TABLE STATUS` statement, `InnoDB` reports the extents that are definitely free in the tablespace. `InnoDB` always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, `InnoDB` contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only in an (automatic) purge operation after they are no longer needed for transaction rollbacks or consistent reads. (See Section 14.2.9, "InnoDB Multi-Versioning".)

To see information about the tablespace, use the Tablespace Monitor. See Section 14.2.13.1, "SHOW ENGINE INNODB STATUS and the InnoDB Monitors".

The maximum row length, except for variable-length columns (`VARBINARY`, `VARCHAR`, `BLOB` and `TEXT`), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. `LONGBLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including `BLOB` and `TEXT` columns, must be less than 4GB.

If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page. For a column chosen for off-page storage, `InnoDB` stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored.

14.2.11.3 Defragmenting a Table

If there are random insertions into or deletions from the indexes of a table, the indexes may become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

One symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

(In the preceding query, we are “fooling” the SQL optimizer into scanning the clustered index rather than a secondary index.) Most disks can read 10MB/s to 50MB/s, which can be used to estimate how fast a table scan should be.

It can speed up index scans if you periodically perform a “null” ALTER TABLE operation, which causes MySQL to rebuild the table:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

Another way to perform a defragmentation operation is to use mysqldump to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the InnoDB filespace management algorithm guarantees that fragmentation in the index does not occur.

14.2.12 InnoDB Error Handling

The following items describe how InnoDB performs error handling. InnoDB sometimes rolls back only the statement that failed, other times it rolls back the entire transaction.

- If you run out of file space in the tablespace, a MySQL Table is full error occurs and InnoDB rolls back the SQL statement.
- A transaction deadlock causes InnoDB to roll back the entire transaction. Retry the whole transaction when this happens.

  A lock wait timeout causes InnoDB to roll back only the single statement that was waiting for the lock and encountered the timeout. (Until MySQL 5.0.13 InnoDB rolled back the entire transaction if a lock wait timeout happened. You can restore this behavior by starting the server with the --innodb_rollback_on_timeout option, available as of MySQL 5.0.32.) You should normally retry the statement if using the current behavior or the entire transaction if using the old behavior.

  Both deadlocks and lock wait timeouts are normal on busy servers and it is necessary for applications to be aware that they may happen and handle them by retrying. You can make them less likely by doing as little work as possible between the first change to data during a transaction and the commit, so the locks are held for the shortest possible time and for the smallest possible number of rows. Sometimes splitting work between different transactions may be practical and helpful.

  When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was START TRANSACTION or BEGIN statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of COMMIT, ROLLBACK, or some SQL statement that causes an implicit commit.
- A duplicate-key error rolls back the SQL statement, if you have not specified the IGNORE option in your statement.
- A row too long error rolls back the SQL statement.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Other errors are mostly detected by the MySQL layer of code (above the InnoDB storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit ROLLBACK SQL statement, SHOW PROCESSLIST displays Rolling back in the State column for the relevant connection.

14.2.13 InnoDB Troubleshooting

14.2.13.1 SHOW ENGINE INNODB STATUS and the InnoDB Monitors

InnoDB Monitors provide information about the InnoDB internal state. This information is useful for performance tuning. Each Monitor can be enabled by creating a table with a special name, which causes InnoDB to write Monitor output periodically. Output for the standard InnoDB Monitor is also available on demand through the SHOW ENGINE INNODB STATUS SQL statement. Additionally, to assist with troubleshooting, InnoDB temporarily enables standard InnoDB Monitor output under certain conditions. For more information, see Section 14.2.13, “InnoDB Troubleshooting”.

There are several types of InnoDB Monitors:

- The standard InnoDB Monitor displays the following types of information:
  - Table and record locks held by each active transaction
  - Lock waits of a transactions
  - Semaphore waits of threads
  - Pending file I/O requests
  - Buffer pool statistics
  - Purge and insert buffer merge activity of the main InnoDB thread

  For a discussion of InnoDB lock modes, see Section 14.2.8.1, “InnoDB Lock Modes”.

  To enable the standard InnoDB Monitor for periodic output, create a table named innodb_monitor. To obtain Monitor output on demand, use the SHOW ENGINE INNODB STATUS SQL statement to fetch the output to your client program. If you are using the mysql interactive client, the output is more readable if you replace the usual semicolon statement terminator with \G:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

- The InnoDB Lock Monitor prints additional lock information as part of the standard InnoDB Monitor output. To enable the InnoDB Lock Monitor, create a table named innodb_lock_monitor.
- The InnoDB Tablespace Monitor prints a list of file segments in the shared tablespace and validates the tablespace allocation data structures. To enable this Monitor for periodic output, create a table named innodb_tablespace_monitor.
- The InnoDB Table Monitor prints the contents of the InnoDB internal data dictionary. To enable this Monitor for periodic output, create a table named innodb_table_monitor.

To enable an InnoDB Monitor for periodic output, use a CREATE TABLE statement to create the table associated with the Monitor. For example, to enable the standard InnoDB Monitor, create the innodb_monitor table:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

To stop the Monitor, drop the table:

```
DROP TABLE innodb_monitor;
```

The `CREATE TABLE` syntax is just a way to pass a command to the `InnoDB` engine through MySQL's SQL parser: The only things that matter are the table name `innodb_monitor` and that it be an `InnoDB` table. The structure of the table is not relevant at all for the `InnoDB` Monitor. If you shut down the server, the Monitor does not restart automatically when you restart the server. Drop the Monitor table and issue a new `CREATE TABLE` statement to start the Monitor. (This syntax may change in a future release.)

When you enable `InnoDB` Monitors for periodic output, `InnoDB` writes their output to the `mysqld` server standard error output (`stderr`). In this case, no output is sent to clients. When switched on, `InnoDB` Monitors print data about every 15 seconds. Server output usually is directed to the error log (see Section 5.4.1, “The Error Log”). This data is useful in performance tuning. On Windows, start the server from a command prompt in a console window with the `--console` option if you want to direct the output to the window rather than to the error log.

`InnoDB` sends diagnostic output to `stderr` or to files rather than to `stdout` or fixed-size memory buffers, to avoid potential buffer overflows. As a side effect, the output of `SHOW ENGINE INNODB STATUS` is written to a status file in the MySQL data directory every fifteen seconds. The name of the file is `innodb_status.pid`, where *`pid`* is the server process ID. `InnoDB` removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see whether they contain useful information about the cause of abnormal shutdowns. The `innodb_status.pid` file is created only if the configuration option `innodb-status-file=1` is set.

`InnoDB` Monitors should be enabled only when you actually want to see Monitor information because output generation does result in some performance decrement. Also, if you enable monitor output by creating the associated table, your error log may become quite large if you forget to remove the table later.

For additional information about `InnoDB` monitors, see:

- Mark Leith: InnoDB Table and Tablespace Monitors
- MySQL Performance Blog: SHOW INNODB STATUS walk through

Each monitor begins with a header containing a timestamp and the monitor name. For example:

```
=====================================
141017  8:11:59 INNODB MONITOR OUTPUT
=====================================
```

The header for the standard Monitor (`INNODB MONITOR OUTPUT`) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

The following sections describe the output for each Monitor.

InnoDB Standard Monitor and Lock Monitor Output

The Lock Monitor is the same as the standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output by creating the associated `InnoDB` table turns on the same output stream, but the stream includes the extra information if the Lock Monitor is enabled. For example, if you create the `innodb_monitor` and `innodb_lock_monitor` tables, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor by removing the `innodb_lock_monitor` table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Example InnoDB Monitor output (as of MySQL 5.0.96):

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
141017  8:11:59 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 4 seconds
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 96, signal count 96
Mutex spin waits 0, rounds 1620, OS waits 46
RW-shared spins 92, OS waits 46; RW-excl spins 4, OS waits 4
------------------------
LATEST FOREIGN KEY ERROR
------------------------
141017  8:02:40 Transaction:
TRANSACTION 0 1799, ACTIVE 0 sec, process no 3857, OS thread id 140048508237568
inserting, thread declared inside InnoDB 497
mysql tables in use 1, locked 1
5 lock struct(s), heap size 1216, undo log entries 3
MySQL thread id 1, query id 33 localhost msandbox update
INSERT INTO child VALUES
    (NULL, 1)
    , (NULL, 2)
    , (NULL, 3)
    , (NULL, 4)
    , (NULL, 5)
    , (NULL, 6)
Foreign key constraint fails for table `mysql/child`:
,
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`)
 ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index `par_ind` tuple:
DATA TUPLE: 2 fields;
 0: len 4; hex 80000003; asc     ;; 1: len 4; hex 80000003; asc     ;;

But in parent table `mysql/parent`, in index `PRIMARY`,
the closest match we can find is record:
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000004; asc     ;; 1: len 6; hex 000000000703; asc       ;; 2:
len 7; hex 80000000320134; asc     2 4;;

------------------------
LATEST DETECTED DEADLOCK
------------------------
141017  8:04:35
*** (1) TRANSACTION:
TRANSACTION 0 1803, ACTIVE 11 sec, process no 3857, OS thread id 140048507971328
 starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 368
MySQL thread id 2, query id 89 localhost msandbox updating
DELETE FROM t WHERE i = 1
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 4555 n bits 72 index `GEN_CLUST_INDEX` of table
`mysql/t` trx id 0 1803 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000000200; asc       ;; 1: len 6; hex 000000000709; asc
;; 2: len 7; hex 80000000320110; asc     2  ;; 3: len 4; hex 80000001; asc     ;
;

*** (2) TRANSACTION:
TRANSACTION 0 1802, ACTIVE 33 sec, process no 3857, OS thread id 140048508237568
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
 starting index read, thread declared inside InnoDB 500
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1216
MySQL thread id 1, query id 90 localhost msandbox updating
DELETE FROM t WHERE i = 1
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 4555 n bits 72 index `GEN_CLUST_INDEX` of table
`mysql/t` trx id 0 1802 lock mode S
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
 0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000000200; asc       ;; 1: len 6; hex 000000000709; asc
;; 2: len 7; hex 80000000320110; asc     2  ;; 3: len 4; hex 80000001; asc     ;
;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 4555 n bits 72 index `GEN_CLUST_INDEX` of table
`mysql/t` trx id 0 1802 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000000200; asc       ;; 1: len 6; hex 000000000709; asc
;; 2: len 7; hex 80000000320110; asc     2  ;; 3: len 4; hex 80000001; asc     ;
;

*** WE ROLL BACK TRANSACTION (2)
------------
TRANSACTIONS
------------
Trx id counter 0 2119
Purge done for trx's n:o < 0 2048 undo n:o < 0 0
History list length 6
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 2118, not started, process no 3857, OS thread id 14004850770508
8
MySQL thread id 4, query id 430 localhost msandbox end
INSERT INTO `salaries` VALUES (53000,67817,'1998-12-05','1999-12-05'),(53000,689
97,'1999-12-05','2000-12-04'),(53000,71070,'2000-12-04','2001-12-04'),(53000,752
14,'2001-12-04','9999-01-01'),(53001,49513,'1985-03-18','1986-03-18'),(53001,493
81,'1986-03-18','1987-03-18'),(53001,49006,'1987-03-18','1988-03-17'),(53001,523
22,'1988-03-17','1989-03-17'),(53001,54980,'1989-03-17','1990-03-17'),(53001,553
51,'1990-03-17','1991-03-17'),(53001,55221,'1991-03-17','1991-08-06'),(53002,400
00,'1998-03-26','1999-03-26'),(53002,41830,'1999-03-26','2000-03-25'),(53002,445
10,'2000-03-25','2001-03-25'),(53002,480
---TRANSACTION 0 1803, not started, process no 3857, OS thread id 14004850797132
8
MySQL thread id 2, query id 336 localhost msandbox
---TRANSACTION 0 1802, not started, process no 3857, OS thread id 14004850823756
8
MySQL thread id 1, query id 431 localhost msandbox
SHOW ENGINE INNODB STATUS
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
2454 OS file reads, 5144 OS file writes, 1437 OS fsyncs
42.74 reads/s, 26444 avg bytes/read, 86.73 writes/s, 25.24 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
0 inserts, 0 merged recs, 0 merges
Hash table size 17393, used cells 8885, node heap has 20 buffer(s)
329491.13 hash searches/s, 2940.76 non-hash searches/s
---
LOG
---
Log sequence number 0 927897901
Log flushed up to   0 927897901
Last checkpoint at  0 923527492
0 pending log writes, 0 pending chkp writes
1837 log i/o's done, 33.49 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 20641866; in additional pool allocated 951040
Buffer pool size   512
Free buffers       1
Database pages     491
Modified db pages  161
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 6690, created 17907, written 21574
68.98 reads/s, 325.42 creates/s, 394.65 writes/s
Buffer pool hit rate 1000 / 1000
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread process no. 3857, id 140048477996800, state: sleeping
Number of rows inserted 5427964, updated 0, deleted 3, read 4
110594.85 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

`InnoDB` Monitor output is limited to 64,000 bytes when produced using the `SHOW ENGINE INNODB STATUS` statement. This limit does not apply to output written to the server's error output.

Some notes on the output sections:

Status

This section shows the timestamp, the monitor name, and the number of seconds that per-second averages are based on. The number of seconds is the elapsed time between the current time and the last time `InnoDB` Monitor output was printed.

SEMAPHORES

This section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside `InnoDB`. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting the `innodb_thread_concurrency` system variable smaller than the default value might help in such situations.

LATEST FOREIGN KEY ERROR

This section provides information about the most recent foreign key constraint error. It is not present if no such error has occurred. The contents include the statement that failed as well as information about the constraint that failed and the referenced and referencing tables.

LATEST DETECTED DEADLOCK

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This section provides information about the most recent deadlock. It is not present if no deadlock has occurred. The contents show which transactions are involved, the statement each was attempting to execute, the locks they have and need, and which transaction InnoDB decided to roll back to break the deadlock. The lock modes reported in this section are explained in Section 14.2.8.1, “InnoDB Lock Modes”.

TRANSACTIONS

If this section reports lock waits, your applications might have lock contention. The output can also help to trace the reasons for transaction deadlocks.

FILE I/O

This section provides information about threads that InnoDB uses to perform various types of I/O. The first few of these are dedicated to general InnoDB processing. The contents also display information for pending I/O operations and statistics for I/O performance.

On Unix, the number of threads is always 4. On Windows, the number depends on the setting of the innodb_file_io_threads system variable.

INSERT BUFFER AND ADAPTIVE HASH INDEX

This section shows the status of the InnoDB insert buffer and adaptive hash index. (See Section 14.2.10.3, “Insert Buffering”, and Section 14.2.10.4, “Adaptive Hash Indexes”.) The contents include the number of operations performed for each, plus statistics for hash index performance.

LOG

This section displays information about the InnoDB log. The contents include the current log sequence number, how far the log has been flushed to disk, and the position at which InnoDB last took a checkpoint. (See Section 14.2.6.3, “InnoDB Checkpoints”.) The section also displays information about pending writes and write performance statistics.

BUFFER POOL AND MEMORY

This section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.

For additional information about the operation of the buffer pool, see Section 8.10.2, “The InnoDB Buffer Pool”.

ROW OPERATIONS

This section shows what the main thread is doing, including the number and performance rate for each type of row operation.

InnoDB Tablespace Monitor Output

The InnoDB Tablespace Monitor prints information about the file segments in the shared tablespace and validates the tablespace allocation data structures. If you use individual tablespaces by enabling innodb_file_per_table, the Tablespace Monitor does not describe those tablespaces.

Example InnoDB Tablespace Monitor output:

```
=====================================
090408 21:28:09 INNODB TABLESPACE MONITOR OUTPUT
=====================================
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
first seg id not used 0 23845
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
SEGMENT id 0 2 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 3 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
SEGMENT id 0 488 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 17 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 171 space 0; page 2; res 592 used 481; full ext 7
fragm pages 16; free extents 0; not full extents 2: pages 17
SEGMENT id 0 172 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 173 space 0; page 2; res 96 used 44; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 12
...
SEGMENT id 0 601 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
NUMBER of file segments: 73
Validating tablespace
Validation ok
---------------------------------------
END OF INNODB TABLESPACE MONITOR OUTPUT
=======================================
```

The Tablespace Monitor output includes information about the shared tablespace as a whole, followed by a list containing a breakdown for each segment within the tablespace.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of size 1MB (64 consecutive pages).

The initial part of the output that displays overall tablespace information has this format:

```
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
```

Overall tablespace information includes these values:

- `id`: The tablespace ID. A value of 0 refers to the shared tablespace.
- `size`: The current tablespace size in pages.
- `free limit`: The minimum page number for which the free list has not been initialized. Pages at or above this limit are free.
- `free extents`: The number of free extents.
- `not full frag extents`, `used pages`: The number of fragment extents that are not completely filled, and the number of pages in those extents that have been allocated.
- `full frag extents`: The number of completely full fragment extents.
- `first seg id not used`: The first unused segment ID.

Individual segment information has this format:

This documentation is for an older version. If you're

```
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
```

Segment information includes these values:

`id`: The segment ID.

`space`, `page`: The tablespace number and page within the tablespace where the segment “inode” is located. A tablespace number of 0 indicates the shared tablespace. `InnoDB` uses inodes to keep track of segments in the tablespace. The other fields displayed for a segment (`id`, `res`, and so forth) are derived from information in the inode.

`res`: The number of pages allocated (reserved) for the segment.

`used`: The number of allocated pages in use by the segment.

`full ext`: The number of extents allocated for the segment that are completely used.

`fragm pages`: The number of initial pages that have been allocated to the segment.

`free extents`: The number of extents allocated for the segment that are completely unused.

`not full extents`: The number of extents allocated for the segment that are partially used.

`pages`: The number of pages used within the not-full extents.

When a segment grows, it starts as a single page, and `InnoDB` allocates the first pages for it individually, up to 32 pages (this is the `fragm pages` value). After that, `InnoDB` allocates complete 64-page extents. `InnoDB` can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

For the example segment shown earlier, it has 32 fragment pages, plus 2 full extents (64 pages each), for a total of 160 pages used out of 160 pages allocated. The following segment has 32 fragment pages and one partially full extent using 14 pages for a total of 46 pages used out of 96 pages allocated:

```
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
```

It is possible for a segment that has extents allocated to it to have a `fragm pages` value less than 32 if some of the individual pages have been deallocated subsequent to extent allocation.

InnoDB Table Monitor Output

The `InnoDB` Table Monitor prints the contents of the `InnoDB` internal data dictionary.

The output contains one section per table. The `SYS_FOREIGN` and `SYS_FOREIGN_COLS` sections are for internal data dictionary tables that maintain information about foreign keys. There are also sections for the Table Monitor table and each user-created `InnoDB` table. Suppose that the following two tables have been created in the `test` database:

```
CREATE TABLE parent
(
  par_id    INT NOT NULL,
  fname      CHAR(20),
  lname      CHAR(20),
  PRIMARY KEY (par_id),
  UNIQUE INDEX (lname, fname)
) ENGINE = INNODB;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE child
(
  par_id      INT NOT NULL,
  child_id    INT NOT NULL,
  name        VARCHAR(40),
  birth       DATE,
  weight      DECIMAL(10,2),
  misc_info   VARCHAR(255),
  last_update TIMESTAMP,
  PRIMARY KEY (par_id, child_id),
  INDEX (name),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = INNODB;
```

Then the Table Monitor output will look something like this (reformatted slightly):

```

===========================================
090420 12:05:26 INNODB TABLE MONITOR OUTPUT
===========================================
--------------------------------------
TABLE: name SYS_FOREIGN, id 0 11, columns 8, indexes 3, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           FOR_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           REF_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           N_COLS: DATA_INT len 4 prec 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6 prec 0;
           DB_TRX_ID: DATA_SYS prtype 257 len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS prtype 258 len 7 prec 0;
  INDEX: name ID_IND, id 0 11, fields 1/6, type 3
   root page 46, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID DB_TRX_ID DB_ROLL_PTR FOR_NAME REF_NAME N_COLS
  INDEX: name FOR_IND, id 0 12, fields 1/2, type 0
   root page 47, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  FOR_NAME ID
  INDEX: name REF_IND, id 0 13, fields 1/2, type 0
   root page 48, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  REF_NAME ID
--------------------------------------
TABLE: name SYS_FOREIGN_COLS, id 0 12, columns 8, indexes 1, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           POS: DATA_INT len 4 prec 0;
           FOR_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           REF_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6 prec 0;
           DB_TRX_ID: DATA_SYS prtype 257 len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS prtype 258 len 7 prec 0;
  INDEX: name ID_IND, id 0 14, fields 2/6, type 3
   root page 49, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID POS DB_TRX_ID DB_ROLL_PTR FOR_COL_NAME REF_COL_NAME
--------------------------------------
TABLE: name test/child, id 0 14, columns 11, indexes 2, appr.rows 210
  COLUMNS: par_id: DATA_INT len 4 prec 0;
           child_id: DATA_INT len 4 prec 0;
           name: DATA_VARCHAR prtype 524303 len 40 prec 0;
           birth: DATA_INT len 3 prec 0;
           weight: type 3 len 5 prec 0;
           misc_info: DATA_VARCHAR prtype 524303 len 255 prec 0;
           last_update: DATA_INT len 4 prec 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6 prec 0;
           DB_TRX_ID: DATA_SYS prtype 257 len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS prtype 258 len 7 prec 0;
  INDEX: name PRIMARY, id 0 17, fields 2/9, type 3
   root page 52, appr.key vals 210, leaf pages 1, size pages 1
   FIELDS:  par_id child_id DB_TRX_ID DB_ROLL_PTR name birth weight misc_info last_update
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  INDEX: name name, id 0 18, fields 1/3, type 0
   root page 53, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  name par_id child_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
              REFERENCES test/parent ( par_id )
-------------------------------------
TABLE: name test/innodb_table_monitor, id 0 15, columns 5, indexes 1, appr.rows 0
  COLUMNS: i: DATA_INT len 4 prec 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6 prec 0;
           DB_TRX_ID: DATA_SYS prtype 257 len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS prtype 258 len 7 prec 0;
  INDEX: name GEN_CLUST_INDEX, id 0 19, fields 0/4, type 1
   root page 54, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  DB_ROW_ID DB_TRX_ID DB_ROLL_PTR i
-------------------------------------
TABLE: name test/parent, id 0 13, columns 7, indexes 2, appr.rows 299
  COLUMNS: par_id: DATA_INT len 4 prec 0;
           fname: DATA_CHAR prtype 524542 len 20 prec 0;
           lname: DATA_CHAR prtype 524542 len 20 prec 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6 prec 0;
           DB_TRX_ID: DATA_SYS prtype 257 len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS prtype 258 len 7 prec 0;
  INDEX: name PRIMARY, id 0 15, fields 1/5, type 3
   root page 50, appr.key vals 299, leaf pages 2, size pages 3
   FIELDS:  par_id DB_TRX_ID DB_ROLL_PTR fname lname
  INDEX: name lname, id 0 16, fields 2/3, type 2
   root page 51, appr.key vals 300, leaf pages 1, size pages 1
   FIELDS:  lname fname par_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
              REFERENCES test/parent ( par_id )
-----------------------------------
END OF INNODB TABLE MONITOR OUTPUT
===================================
```

For each table, Table Monitor output contains a section that displays general information about the table and specific information about its columns, indexes, and foreign keys.

The general information for each table includes the table name (in *`db_name/tbl_name`* format except for internal tables), its ID, the number of columns and indexes, and an approximate row count.

The `COLUMNS` part of a table section lists each column in the table. Information for each column indicates its name and data type characteristics. Some internal columns are added by `InnoDB`, such as `DB_ROW_ID` (row ID), `DB_TRX_ID` (transaction ID), and `DB_ROLL_PTR` (a pointer to the rollback/undo data).

- `DATA_`*`xxx`*: These symbols indicate the data type. There may be multiple `DATA_`*`xxx`* symbols for a given column.
- `prtype`: The column's “precise” type. This field includes information such as the column data type, character set code, nullability, signedness, and whether it is a binary string. This field is described in the `innobase/include/data0type.h` source file.
- `len`: The column length in bytes.
- `prec`: The precision of the type.

Each `INDEX` part of the table section provides the name and characteristics of one table index:

- `name`: The index name. If the name is `PRIMARY`, the index is a primary key. If the name is `GEN_CLUST_INDEX`, the index is the clustered index that is created automatically if the table definition doesn't include a primary key or non-`NULL` unique index. See Section 14.2.10.1, “Clustered and Secondary Indexes”.
- `id`: The index ID.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `fields`: The number of fields in the index, as a value in *`m/n`* format:
  - *`m`* is the number of user-defined columns; that is, the number of columns you would see in the index definition in a `CREATE TABLE` statement.
  - *`n`* is the total number of index columns, including those added internally. For the clustered index, the total includes the other columns in the table definition, plus any columns added internally. For a secondary index, the total includes the columns from the primary key that are not part of the secondary index.
- `type`: The index type. This is a bit field. For example, 1 indicates a clustered index and 2 indicates a unique index, so a clustered index (which always contains unique values), will have a `type` value of 3. An index with a `type` value of 0 is neither clustered nor unique. The flag values are defined in the `innobase/include/dict0mem.h` source file.
- `root page`: The index root page number.
- `appr. key vals`: The approximate index cardinality.
- `leaf pages`: The approximate number of leaf pages in the index.
- `size pages`: The approximate total number of pages in the index.
- `FIELDS`: The names of the fields in the index. For a clustered index that was generated automatically, the field list begins with the internal `DB_ROW_ID` (row ID) field. `DB_TRX_ID` and `DB_ROLL_PTR` are always added internally to the clustered index, following the fields that comprise the primary key. For a secondary index, the final fields are those from the primary key that are not part of the secondary index.

The end of the table section lists the `FOREIGN KEY` definitions that apply to the table. This information appears whether the table is a referencing or referenced table.

14.2.13.2 InnoDB General Troubleshooting

The following general guidelines apply to troubleshooting `InnoDB` problems:

- When an operation fails or you suspect a bug, look at the MySQL server error log (see Section 5.4.1, “The Error Log”). Section B.3, “Server Error Codes and Messages” provides troubleshooting information for some of the common `InnoDB`-specific errors that you may encounter.
- Issues relating to the `InnoDB` data dictionary include failed `CREATE TABLE` statements (orphan table files), inability to open `.InnoDB` files, and `system cannot find the path specified` errors. For information about these sorts of problems and errors, see Section 14.2.13.3, “Troubleshooting InnoDB Data Dictionary Operations”.
- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through `mysqld_safe` or as a Windows service. You can then see what `mysqld` prints to the console, and so have a better grasp of what is going on. On Windows, start `mysqld` with the `--console` option to direct the output to the console window.
- Enable the `InnoDB` Monitors to obtain information about a problem (see Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”). If the problem is performance-related, or your server appears to be hung, you should enable the standard Monitor to print information about the internal state of `InnoDB`. If the problem is with locks, enable the Lock Monitor. If the problem is in creation of tables or other data dictionary operations, enable the Table Monitor to print the contents of the `InnoDB` internal data dictionary. To see tablespace information enable the Tablespace Monitor.

  `InnoDB` temporarily enables standard `InnoDB` Monitor output under the following conditions:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- A long semaphore wait
- `InnoDB` cannot find free blocks in the buffer pool
- Over 67% of the buffer pool is occupied by lock heaps or the adaptive hash index

- If you suspect that a table is corrupt, run `CHECK TABLE` on that table.

14.2.13.3 Troubleshooting InnoDB Data Dictionary Operations

A specific issue with tables is that the MySQL server keeps data dictionary information in `.frm` files it stores in the database directories, whereas `InnoDB` also stores the information into its own data dictionary inside the tablespace files. If you move `.frm` files around, or if the server crashes in the middle of a data dictionary operation, the locations of the `.frm` files may end up out of synchrony with the locations recorded in the `InnoDB` internal data dictionary.

CREATE TABLE Failure Due to Orphan Table

A symptom of an out-of-sync data dictionary is that a `CREATE TABLE` statement fails. If this occurs, you should look in the server's error log. If the log says that the table already exists inside the `InnoDB` internal data dictionary, you have an orphan table inside the `InnoDB` tablespace files that has no corresponding `.frm` file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphan table by following the instructions given in the error message. If you are still unable to use `DROP TABLE` successfully, the problem may be due to name completion in the `mysql` client. To work around this problem, start the `mysql` client with the `--skip-auto-rehash` option and try `DROP TABLE` again. (With name completion on, `mysql` tries to construct a list of table names, which fails when a problem such as just described exists.)

Cannot Open File Error

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open an `InnoDB` file:

```
ERROR 1016: Can't open file: 'child2.ibd'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphan `.frm` file without a corresponding table inside `InnoDB`. You can drop the orphan `.frm` file by deleting it manually.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Orphan Temporary Tables

If MySQL exits in the middle of an `ALTER TABLE` operation, you may be left with an orphan temporary table that takes up space on your system. This section describes how to identify and remove orphan temporary tables.

Orphan temporary table names begin with an `#sql-` prefix (e.g., `#sql-540_3`). The accompanying `.frm` file has the same base name as the orphan temporary table.

Note

If there is no `.frm` file, you can recreate it. The `.frm` file must have the same table schema as the orphan temporary table (it must have the same columns and indexes) and must be placed in the database directory of the orphan temporary table.

To identify orphan temporary tables on your system, you can view Table Monitor output. Look for table names that begin with `#sql`. If the original table resides in a file-per-table tablespace, the tablespace file (the `#sql-*.ibd` file) for the orphan temporary table should be visible in the database directory.

To remove an orphan temporary table, drop the table by issuing a `DROP TABLE` statement, enclosing the table name in backticks. For example:

```
mysql> DROP TABLE `#sql-540_3`;
```

Tablespace Does Not Exist

With `innodb_file_per_table` enabled, the following message might occur if the `.frm` or `.ibd` files (or both) are missing:

```
InnoDB: in InnoDB data dictionary has tablespace id N,
InnoDB: but tablespace with that id or name does not exist. Have
InnoDB: you deleted or moved .ibd files?
InnoDB: This may also be a table created with CREATE TEMPORARY TABLE
InnoDB: whose .ibd and .frm files MySQL automatically removed, but the
InnoDB: table still exists in the InnoDB internal data dictionary.
```

If this occurs, try the following procedure to resolve the problem:

1. Create a matching `.frm` file in some other database directory and copy it to the database directory where the orphan table is located.
2. Issue `DROP TABLE` for the original table. That should successfully drop the table and `InnoDB` should print a warning to the error log that the `.ibd` file was missing.

14.2.14 Limits on InnoDB Tables

Warning

Do *not* convert MySQL system tables in the `mysql` database from `MyISAM` to `InnoDB` tables. This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or regenerate them by reinitializing the data directory (see Section 2.18.1, “Initializing the Data Directory”).

Warning

It is not a good idea to configure `InnoDB` to use data files or log files on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Maximums and Minimums

- A table can contain a maximum of 1000 columns.
- The InnoDB internal maximum key length is 3500 bytes, but MySQL itself restricts this to 3072 bytes. (1024 bytes for non-64-bit builds before MySQL 5.0.17, and for all builds before 5.0.15.)
- An index key for a single-column index can be up to 767 bytes. The same length limit applies to any index key prefix. See Section 13.1.8, “CREATE INDEX Syntax”.
- The maximum row length, except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.

  If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in Section 14.2.11.2, “File Space Management”.
- Although InnoDB supports row sizes larger than 65,535 bytes internally, MySQL itself imposes a row-size limit of 65,535 for the combined size of all columns:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
    -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
    -> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

  See Section C.7.4, “Limits on Table Column Count and Row Size”.
- On some older operating systems, files must be less than 2GB. This is not a limitation of InnoDB itself, but if you require a large tablespace, you will need to configure it using several smaller data files rather than one large data file.
- The combined size of the InnoDB log files must be less than 4GB.
- The minimum tablespace size is 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- The default database page size in InnoDB is 16KB. By recompiling the code, you can set it to values ranging from 8KB to 64KB. You must update the values of UNIV_PAGE_SIZE and UNIV_PAGE_SIZE_SHIFT in the univ.i source file.

Note

Changing the page size is not a supported operation and there is no guarantee that InnoDB will function normally with a page size other than 16KB. Problems compiling or running InnoDB may occur.

A version of InnoDB built for one page size cannot use data files or log files from a version built for a different page size.

Index Types

- InnoDB tables do not support FULLTEXT indexes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `InnoDB` tables do not support spatial data types before MySQL 5.0.16. As of 5.0.16, `InnoDB` supports spatial data types, but not indexes on them.

Restrictions on InnoDB Tables

- `ANALYZE TABLE` determines index cardinality (as displayed in the `Cardinality` column of `SHOW INDEX` output) by doing eight random dives to each of the index trees and updating index cardinality estimates accordingly. Because these are only estimates, repeated runs of `ANALYZE TABLE` may produce different numbers. This makes `ANALYZE TABLE` fast on `InnoDB` tables but not 100% accurate because it does not take all rows into account.

  MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you can try using `ANALYZE TABLE`. In the few cases that `ANALYZE TABLE` does not produce values good enough for your particular tables, you can use `FORCE INDEX` with your queries to force the use of a particular index, or set the `max_seeks_for_key` system variable to ensure that MySQL prefers index lookups over table scans. See Section 5.1.4, “Server System Variables”, and Section B.5.5, “Optimizer-Related Issues”.

- If statements or transactions are running on a table and `ANALYZE TABLE` is run on the same table followed by a second `ANALYZE TABLE` operation, the second `ANALYZE TABLE` operation is blocked until the statements or transactions are completed. This behavior occurs because `ANALYZE TABLE` marks the currently loaded table definition as obsolete when `ANALYZE TABLE` is finished running. New statements or transactions (including a second `ANALYZE TABLE` statement) must load the new table definition into the table cache, which cannot occur until currently running statements or transactions are completed and the old table definition is purged. Loading multiple concurrent table definitions is not supported.

- `SHOW TABLE STATUS` does not give accurate statistics on `InnoDB` tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.

- `InnoDB` does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. To process a `SELECT COUNT(*) FROM t` statement, `InnoDB` scans an index of the table, which takes some time if the index is not entirely in the buffer pool. If your table does not change often, using the MySQL query cache is a good solution. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. If an approximate row count is sufficient, `SHOW TABLE STATUS` can be used.

- On Windows, `InnoDB` always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names.

- An `AUTO_INCREMENT` column *`ai_col`* must be defined as part of an index such that it is possible to perform the equivalent of an indexed `SELECT MAX(`*`ai_col`*`)` lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

- In MySQL 5.0 before MySQL 5.0.3, `InnoDB` does not support the `AUTO_INCREMENT` table option for setting the initial sequence value in a `CREATE TABLE` or `ALTER TABLE` statement. To set the value with `InnoDB`, insert a dummy row with a value one less and delete that dummy row, or insert the first row with an explicit value specified.

- While initializing a previously specified `AUTO_INCREMENT` column on a table, `InnoDB` sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. While accessing the auto-increment counter, `InnoDB` uses a specific `AUTO-INC` table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other clients cannot insert into

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the table while the AUTO-INC table lock is held. See Section 14.2.3.3, “AUTO_INCREMENT Handling in InnoDB”.

- When you restart the MySQL server, InnoDB may reuse an old value that was generated for an AUTO_INCREMENT column but never stored (that is, a value that was generated during an old transaction that was rolled back).
- When an AUTO_INCREMENT column runs out of values, InnoDB wraps a BIGINT to -9223372036854775808 and BIGINT UNSIGNED to 1. However, BIGINT values have 64 bits, so if you were to insert one million rows per second, it would still take nearly three hundred thousand years before BIGINT reached its upper bound. With all other integer type columns, a duplicate-key error results. This is general MySQL behavior, similar to how MyISAM works.
- DELETE FROM *tbl_name* does not regenerate the table but instead deletes all rows, one by one.
- Under some conditions, TRUNCATE *tbl_name* for an InnoDB table is mapped to DELETE FROM *tbl_name* and does not reset the AUTO_INCREMENT counter. See Section 13.1.21, “TRUNCATE TABLE Syntax”.
- The LOAD TABLE FROM MASTER statement for setting up replication slave servers does not work for InnoDB tables. A workaround is to alter the table to MyISAM on the master, then do the load, and after that alter the master table back to InnoDB. Do not do this if the tables use InnoDB-specific features such as foreign keys.
- Cascaded foreign key actions do not activate triggers.
- You cannot create a table with a column name that matches the name of an internal InnoDB column (including DB_ROW_ID, DB_TRX_ID, DB_ROLL_PTR, and DB_MIX_ID). In versions of MySQL before 5.0.21 this would cause a crash, since 5.0.21 the server will report error 1005 and refers to error −1 in the error message. This restriction applies only to use of the names in uppercase.
- As of MySQL 5.0.19, InnoDB does not ignore trailing spaces when comparing BINARY or VARBINARY column values. See Section 11.4.2, “The BINARY and VARBINARY Types”.

Locking and Transactions

- LOCK TABLES acquires two locks on each table if innodb_table_locks=1 (the default). In addition to a table lock on the MySQL layer, it also acquires an InnoDB table lock. Versions of MySQL before 4.1.2 did not acquire InnoDB table locks; the old behavior can be selected by setting innodb_table_locks=0. If no InnoDB table lock is acquired, LOCK TABLES completes even if some records of the tables are being locked by other transactions.
- All InnoDB locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke LOCK TABLES on InnoDB tables in autocommit=1 mode because the acquired InnoDB table locks would be released immediately.
- You cannot lock additional tables in the middle of a transaction because LOCK TABLES performs an implicit COMMIT and UNLOCK TABLES.
- InnoDB has a limit of 1023 concurrent transactions that have created undo records by modifying data. Workarounds include keeping transactions as small and fast as possible, delaying changes until near the end of the transaction, and using stored routines to reduce client/server latency delays. Applications should commit transactions before doing time-consuming client-side operations.

14.3 The MERGE Storage Engine

The MERGE storage engine, also known as the MRG_MyISAM engine, is a collection of identical MyISAM tables that can be used as one. “Identical” means that all tables have identical column and index

This documentation is for an older version. If you're
This documentation is for an older version. If you're

information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with myisampack. See Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”. Differences in table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS do not matter.

When you create a MERGE table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and an .MRG file contains the names of the underlying MyISAM tables that should be used as one. The tables do not have to be in the same database as the MERGE table.

You can use SELECT, DELETE, UPDATE, and INSERT on MERGE tables. You must have SELECT, DELETE, and UPDATE privileges on the MyISAM tables that you map to a MERGE table.

Note

The use of MERGE tables entails the following security issue: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*. If this behavior is undesirable, you can start the server with the new --skip-merge option to disable the MERGE storage engine. This option is available as of MySQL 5.0.24.

Use of DROP TABLE with a MERGE table drops only the MERGE specification. The underlying tables are not affected.

To create a MERGE table, you must specify a UNION=(*list-of-tables*) option that indicates which MyISAM tables to use. You can optionally specify an INSERT_METHOD option to control how inserts into the MERGE table take place. Use a value of FIRST or LAST to cause inserts to be made in the first or last underlying table, respectively. If you specify no INSERT_METHOD option or if you specify it with a value of NO, inserts into the MERGE table are not permitted and attempts to do so result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.

Column a is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a UNIQUE index in the underlying tables should be indexed in the MERGE table but not as a UNIQUE index.)

After creating the MERGE table, you can use it to issue queries that operate on the group of tables as a whole:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

To remap a `MERGE` table to a different collection of `MyISAM` tables, you can use one of the following methods:

- `DROP` the `MERGE` table and re-create it.
- Use `ALTER TABLE tbl_name UNION=(...)` to change the list of underlying tables.

  Beginning with MySQL 5.0.60, it is also possible to use `ALTER TABLE ... UNION=()` (that is, with an empty `UNION` clause) to remove all of the underlying tables. However, in this case, the table is effectively empty and inserts fail because there is no underlying table to take new rows. Such a table might be useful as a template for creating new `MERGE` tables with `CREATE TABLE ... LIKE`.

As of MySQL 5.0.36, the underlying table definitions and indexes must conform more closely than previously to the definition of the `MERGE` table. Conformance is checked when a table that is part of a `MERGE` table is opened, not when the `MERGE` table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a `MERGE` may cause a failure when the `MERGE` table is accessed. The conformance checks applied to each table are:

- The underlying table and the `MERGE` table must have the same number of columns.
- The column order in the underlying table and the `MERGE` table must match.
- Additionally, the specification for each corresponding column in the parent `MERGE` table and the underlying tables are compared and must satisfy these checks:
  - The column type in the underlying table and the `MERGE` table must be equal.
  - The column length in the underlying table and the `MERGE` table must be equal.
  - The column of the underlying table and the `MERGE` table can be `NULL`.
- The underlying table must have at least as many indexes as the `MERGE` table. The underlying table may have more indexes than the `MERGE` table, but cannot have fewer.

  Note

  A known issue exists where indexes on the same columns must be in identical order, in both the `MERGE` table and the underlying `MyISAM` table. See Bug #33653.

  Each index must satisfy these checks:

  - The index type of the underlying table and the `MERGE` table must be the same.
  - The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the `MERGE` table must be the same.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- For each index part:
  - Index part lengths must be equal.
  - Index part types must be equal.
  - Index part languages must be equal.
  - Check whether index parts can be `NULL`.

For information about the table checks applied prior to MySQL 5.0.36, see Section 14.3.2, “MERGE Table Problems”.

As of MySQL 5.0.44, if a `MERGE` table cannot be opened or used because of a problem with an underlying table, `CHECK TABLE` displays information about which table caused the problem.

Additional Resources

- A forum dedicated to the `MERGE` storage engine is available at http://forums.mysql.com/list.php?93.

14.3.1 MERGE Table Advantages and Disadvantages

`MERGE` tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with `myisampack`, and then create a `MERGE` table to use them as one.
- Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A `MERGE` table structured this way could be much faster than using a single large table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a `MERGE` table for others. You can even have many different `MERGE` tables that use overlapping sets of tables.
- Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a `MERGE` table than to repair a single large table.
- Instantly map many tables as one. A `MERGE` table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, `MERGE` table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a `MERGE` table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you can instead create a `MERGE` table from them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each `MyISAM` table is bound by this limit, but a collection of `MyISAM` tables is not.
- You can create an alias or synonym for a `MyISAM` table by defining a `MERGE` table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages of `MERGE` tables are:

- You can use only identical `MyISAM` tables for a `MERGE` table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Some MyISAM features are unavailable in MERGE tables. For example, you cannot create FULLTEXT indexes on MERGE tables. (You can create FULLTEXT indexes on the underlying MyISAM tables, but you cannot search the MERGE table with a full-text search.)
- If the MERGE table is nontemporary, all underlying MyISAM tables must be nontemporary. If the MERGE table is temporary, the MyISAM tables can be any mix of temporary and nontemporary.
- MERGE tables use more file descriptors than MyISAM tables. If 10 clients are using a MERGE table that maps to 10 tables, the server uses (10 × 10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Index reads are slower. When you read an index, the MERGE storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the MERGE storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes MERGE indexes much slower on eq_ref searches, but not much slower on ref searches. For more information about eq_ref and ref, see Section 13.8.2, “EXPLAIN Syntax”.

14.3.2 MERGE Table Problems

The following are known problems with MERGE tables:

- If you use ALTER TABLE to change a MERGE table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then uses the specified storage engine.
- The INSERT_METHOD table option for a MERGE table indicates which underlying MyISAM table to use for inserts into the MERGE table. However, use of the AUTO_INCREMENT table option for that MyISAM table has no effect for inserts into the MERGE table until at least one row has been inserted directly into the MyISAM table.
- A MERGE table cannot maintain uniqueness constraints over the entire table. When you perform an INSERT, the data goes into the first or last MyISAM table (as determined by the INSERT_METHOD option). MySQL ensures that unique key values remain unique within that MyISAM table, but not over all the underlying tables in the collection.
- Because the MERGE engine cannot enforce uniqueness over the set of underlying tables, REPLACE does not work as expected. The two key facts are:
  - REPLACE can detect unique key violations only in the underlying table to which it is going to write (which is determined by the INSERT_METHOD option). This differs from violations in the MERGE table itself.
  - If REPLACE detects a unique key violation, it will change only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the INSERT_METHOD option.

  Similar considerations apply for INSERT ... ON DUPLICATE KEY UPDATE.
- You should not use ANALYZE TABLE, REPAIR TABLE, OPTIMIZE TABLE, ALTER TABLE, DROP TABLE, DELETE without a WHERE clause, or TRUNCATE TABLE on any of the tables that are mapped into an open MERGE table. If you do so, the MERGE table may still refer to the original table and yield unexpected results. To work around this problem, ensure that no MERGE tables remain open by issuing a FLUSH TABLES statement prior to performing any of the named operations.

  The unexpected results include the possibility that the operation on the MERGE table will report table corruption. If this occurs after one of the named operations on the underlying MyISAM tables, the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

corruption message is spurious. To deal with this, issue a `FLUSH TABLES` statement after modifying the `MyISAM` tables.

- `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table.

- As of MySQL 5.0.36, the definition of the `MyISAM` tables and the `MERGE` table are checked when the tables are accessed (for example, as part of a `SELECT` or `INSERT` statement). The checks ensure that the definitions of the tables and the parent `MERGE` table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations will cause the statement to fail.

  Prior to MySQL 5.0.36, table checks are applied as follows:

  - When you create or alter `MERGE` table, there is no check to ensure that the underlying tables are existing `MyISAM` tables and have identical structures. When the `MERGE` table is used, MySQL checks that the row length for all mapped tables is equal, but this is not foolproof. If you create a `MERGE` table from dissimilar `MyISAM` tables, you are very likely to run into strange problems.

  - Similarly, if you create a `MERGE` table from non-`MyISAM` tables, or if you drop an underlying table or alter it to be a non-`MyISAM` table, no error for the `MERGE` table occurs until later when you attempt to use it.

  - Because the underlying `MyISAM` tables need not exist when the `MERGE` table is created, you can create the tables in any order, as long as you do not use the `MERGE` table until all of its underlying tables are in place. Also, if you can ensure that a `MERGE` table will not be used during a given period, you can perform maintenance operations on the underlying tables, such as backing up or restoring them, altering them, or dropping and recreating them. It is not necessary to redefine the `MERGE` table temporarily to exclude the underlying tables while you are operating on them.

- The order of indexes in the `MERGE` table and its underlying tables should be the same. If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table, and then use `ALTER TABLE` to add a nonunique index on the `MERGE` table, the index ordering is different for the tables if there was already a nonunique index in the underlying table. (This happens because `ALTER TABLE` puts `UNIQUE` indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.

- If you encounter an error message similar to `ERROR 1017 (HY000): Can't find file: 'tbl_name.MRG' (errno: 2)`, it generally indicates that some of the underlying tables do not use the `MyISAM` storage engine. Confirm that all of these tables are `MyISAM`.

- The maximum number of rows in a `MERGE` table is 2^{32} (~4.295E+09; the same as for a `MyISAM` table). It is not possible to merge multiple `MyISAM` tables into a single `MERGE` table that would have more than this number of rows. However, if you build MySQL using the `--with-big-tables` option, then the maximum number of rows is increased to 2^{64} (1.844E+19); for more information, see Section 2.17.3, “MySQL Source-Configuration Options”.

  Note

  As of MySQL 5.0.4, all standard binaries are built with this option.

- The `MERGE` storage engine does not support `INSERT DELAYED` statements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Using MERGE on underlying MyISAM tables that have different row formats is possible.
- In some cases, differing PACK_KEYS table option values among the MERGE and underlying tables cause unexpected results if the underlying tables contain CHAR or BINARY columns. As a workaround, use ALTER TABLE to ensure that all involved tables have the same PACK_KEYS value. (Bug #50646)

14.4 The MEMORY (HEAP) Storage Engine

The MEMORY storage engine creates tables with contents that are stored in memory. Formerly, these were known as HEAP tables. MEMORY is the preferred term, although HEAP remains supported for backward compatibility.

The MEMORY storage engine associates each table with one disk file. The file name begins with the table name and has an extension of .frm to indicate that it stores the table definition.

To specify that you want to create a MEMORY table, indicate that with an ENGINE table option:

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in .frm files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
    ->     SELECT ip,SUM(downloads) AS down
    ->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.
- MEMORY tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.
- The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

  ```
  CREATE TABLE lookup
      (id INT, INDEX USING HASH (id))
      ENGINE = MEMORY;
  CREATE TABLE lookup
      (id INT, INDEX USING BTREE (id))
      ENGINE = MEMORY;
  ```

  For general characteristics of B-tree and hash indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If a MEMORY table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.
- MEMORY tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)
- Columns that are indexed can contain NULL values.
- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.
- MEMORY supports INSERT DELAYED. See Section 13.2.5.2, “INSERT DELAYED Syntax”.
- Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.
- MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal temporary tables are:
  - MEMORY tables are never converted to disk tables. If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in Section 8.4.4, “Internal Temporary Table Use in MySQL”.
  - The maximum size of MEMORY tables is limited by the max_heap_table_size system variable, which has a default value of 16MB. To have larger (or smaller) MEMORY tables, you must change the value of this variable. The value in effect for CREATE TABLE is the value used for the life of the table. (If you use ALTER TABLE or TRUNCATE TABLE, the value in effect at that time becomes the new maximum size for the table. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size value.) You can set the size for individual tables as described later in this section.
- The server needs sufficient memory to maintain all MEMORY tables that are in use at the same time.
- Memory is not reclaimed if you delete individual rows from a MEMORY table. Memory is reclaimed only when the entire table is deleted. Memory that was previously used for rows that have been deleted will be re-used for new rows only within the same table. To free up the memory used by rows that have been deleted, use ALTER TABLE ENGINE=MEMORY to force a table rebuild.

  To free all the memory used by a MEMORY table when you no longer require its contents, you should execute DELETE or TRUNCATE TABLE to remove all rows, or remove the table altogether using DROP TABLE.
- If you want to populate a MEMORY table when the MySQL server starts, you can use the --init-file option. For example, you can put statements such as INSERT INTO ... SELECT or LOAD DATA INFILE into this file to load the table from a persistent data source. See Section 5.1.3, “Server Command Options”, and Section 13.2.6, “LOAD DATA INFILE Syntax”.
- A server's MEMORY tables become empty when it is shut down and restarted. However, if the server is a replication master, its slave are not aware that these tables have become empty, so they returns out-of-date content if you select data from these tables. To handle this, when a MEMORY table is used on a master for the first time since it was started, a DELETE statement is written to the master's binary log

This documentation is for an older version. If you're
This documentation is for an older version. If you're

automatically, thus synchronizing the slave to the master again. Note that even with this strategy, the slave still has outdated data in the table during the interval between the master's restart and its first use of the table. However, if you use the `--init-file` option to populate the `MEMORY` table on the master at startup, it ensures that this time interval is zero.

- The memory needed for one row in a `MEMORY` table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) × 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) × 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

  `ALIGN()` represents a round-up factor to cause the row length to be an exact multiple of the `char` pointer size. `sizeof(char*)` is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the `max_heap_table_size` system variable sets the limit on the maximum size of `MEMORY` tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global `max_heap_table_size` value unless you intend the value to be used for `MEMORY` tables created by all clients.) The following example creates two `MEMORY` tables, with a maximum size of 1MB and 2MB, respectively:

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

Both tables will revert to the server's global `max_heap_table_size` value if the server restarts.

You can also specify a `MAX_ROWS` table option in `CREATE TABLE` statements for `MEMORY` tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the `max_heap_table_size` value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use `MAX_ROWS`, set `max_heap_table_size` at least as high as the value to which you want each `MEMORY` table to be able to grow.

Additional Resources

- A forum dedicated to the `MEMORY` storage engine is available at http://forums.mysql.com/list.php?92.

14.5 The BDB (BerkeleyDB) Storage Engine

Sleepycat Software has provided MySQL with the Berkeley DB transactional storage engine. This storage engine typically is called `BDB` for short. `BDB` tables may have a greater chance of surviving crashes and are also capable of `COMMIT` and `ROLLBACK` operations on transactions.

Support for the `BDB` storage engine is included in MySQL source distributions, which come with a `BDB` distribution that is patched to make it work with MySQL. You cannot use an unpatched version of `BDB` with MySQL.

BDB support will be removed

As of MySQL 5.1, `BDB` is not supported.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For general information about Berkeley DB, please visit the Sleepycat Web site, http://www.sleepycat.com/.

14.5.1 Operating Systems Supported by BDB

We know that the BDB storage engine works with the following operating systems:

- Linux 2.x Intel
- Sun Solaris (SPARC and x86)
- FreeBSD 4.x/5.x (x86, sparc64)
- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x
- Windows

The BDB storage engine does *not* work with the following operating systems:

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA-64
- Linux 2.x s390
- OS X

Note

The preceding lists are not complete. We update them as we receive more information.

If you build MySQL from source with support for BDB tables, but the following error occurs when you start `mysqld`, it means that the BDB storage engine is not supported for your architecture:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded
Can't init databases
```

In this case, you must rebuild MySQL without BDB support or start the server with the `--skip-bdb` option.

14.5.2 Installing BDB

If you have downloaded a binary version of MySQL that includes support for Berkeley DB, simply follow the usual binary distribution installation instructions.

If you build MySQL from source, you can enable BDB support by invoking `configure` with the `--with-berkeley-db` option in addition to any other options that you normally use. Download a MySQL 5.0 distribution, change location into its top-level directory, and run this command:

```
shell> ./configure --with-berkeley-db [other-options]
```

For more information, Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”, and Section 2.17, “Installing MySQL from Source”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

14.5.3 BDB Startup Options

The following options to `mysqld` can be used to change the behavior of the `BDB` storage engine. For more information, see Section 5.1.3, “Server Command Options”.

Table 14.3 BDB Option/Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
bdb_cache_size			Yes		Global	No
bdb-home	Yes	Yes			Global	No
- *Variable*: bdb_home			Yes		Global	No
bdb-lock-detect	Yes	Yes	Yes		Global	No
bdb_log_buffer_size			Yes		Global	No
bdb-logdir	Yes	Yes			Global	No
- *Variable*: bdb_logdir			Yes		Global	No
bdb_max_lock			Yes		Global	No
bdb-no-recover	Yes	Yes				
bdb-no-sync	Yes	Yes				
bdb-shared-data	Yes	Yes			Global	No
- *Variable*: bdb_shared_data			Yes		Global	No
bdb-tmpdir	Yes	Yes			Global	No
- *Variable*: bdb_tmpdir			Yes		Global	No
have_bdb			Yes		Global	No
skip-bdb	Yes	Yes				
skip-sync-bdb-logs	Yes	Yes	Yes		Global	No
sync-bdb-logs	Yes	Yes	Yes		Global	No

- `--bdb-home=`*`dir_name`*

  The base directory for `BDB` tables. This should be the same directory that you use for `--datadir`.

- `--bdb-lock-detect=`*`method`*

  The `BDB` lock detection method. The option value should be `DEFAULT`, `OLDEST`, `RANDOM`, or `YOUNGEST`.

- `--bdb-logdir=`*`file_name`*

  The `BDB` log file directory.

- `--bdb-no-recover`

  Do not start Berkeley DB in recover mode.

- `--bdb-no-sync`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Don't synchronously flush the BDB logs. This option is deprecated; use --skip-sync-bdb-logs instead (see the description for --sync-bdb-logs).

- --bdb-shared-data

  Start Berkeley DB in multi-process mode. (Do not use DB_PRIVATE when initializing Berkeley DB.)

- --bdb-tmpdir=*dir_name*

  The BDB temporary file directory.

- --skip-bdb

  Disable the BDB storage engine.

- --sync-bdb-logs

  Synchronously flush the BDB logs. This option is enabled by default. Use --skip-sync-bdb-logs to disable it.

If you use the --skip-bdb option, MySQL does not initialize the Berkeley DB library and this saves a lot of memory. However, if you use this option, you cannot use BDB tables. If you try to create a BDB table, MySQL uses the default storage engine instead.

Normally, you should start mysqld without the --bdb-no-recover option if you intend to use BDB tables. However, this may cause problems when you try to start mysqld if the BDB log files are corrupted. See Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”.

With the bdb_max_lock variable, you can specify the maximum number of locks that can be active on a BDB table. The default is 10,000. You should increase this if errors such as the following occur when you perform long transactions or when mysqld has to examine many rows to execute a query:

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

You may also want to change the binlog_cache_size and max_binlog_cache_size variables if you are using large multiple-statement transactions. See Section 5.4.3, “The Binary Log”.

See also Section 5.1.4, “Server System Variables”.

14.5.4 Characteristics of BDB Tables

Each BDB table is stored on disk in two files. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and a .db file contains the table data and indexes.

To specify explicitly that you want a BDB table, indicate that with an ENGINE table option:

```
CREATE TABLE t (i INT) ENGINE = BDB;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.

BerkeleyDB is a synonym for BDB in the ENGINE table option.

The BDB storage engine provides transactional tables. The way you use these tables depends on the autocommit mode:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If you are running with autocommit enabled (which is the default), changes to `BDB` tables are committed immediately and cannot be rolled back.
- If you are running with autocommit disabled, changes do not become permanent until you execute a `COMMIT` statement. Instead of committing, you can execute `ROLLBACK` to forget the changes.

  You can start a transaction with the `START TRANSACTION` or `BEGIN` statement to suspend autocommit, or with `SET autocommit = 0` to disable autocommit explicitly.

For more information about transactions, see Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

The `BDB` storage engine has the following characteristics:

- `BDB` tables can have up to 31 indexes per table, 16 columns per index, and a maximum key size of 1024 bytes.
- MySQL requires a primary key in each `BDB` table so that each row can be uniquely identified. If you don't create one explicitly by declaring a `PRIMARY KEY`, MySQL creates and maintains a hidden primary key for you. The hidden key has a length of five bytes and is incremented for each insert attempt. This key does not appear in the output of `SHOW CREATE TABLE` or `DESCRIBE`.
- The primary key is faster than any other index, because it is stored together with the row data. The other indexes are stored as the key data plus the primary key, so it is important to keep the primary key as short as possible to save disk space and get better speed.

  This behavior is similar to that of `InnoDB`, where shorter primary keys save space not only in the primary index but in secondary indexes as well.
- If all columns that you access in a `BDB` table are part of the same index or part of the primary key, MySQL can execute the query without having to access the actual row. In a `MyISAM` table, this can be done only if the columns are part of the same index.
- Sequential scanning is slower for `BDB` tables than for `MyISAM` tables because the data in `BDB` tables is stored in B-trees and not in a separate data file.
- Key values are not prefix- or suffix-compressed like key values in `MyISAM` tables. In other words, key information takes a little more space in `BDB` tables compared to `MyISAM` tables.
- There are often holes in the `BDB` table to permit you to insert new rows in the middle of the index tree. This makes `BDB` tables somewhat larger than `MyISAM` tables.
- `SELECT COUNT(*) FROM` *`tbl_name`* is slow for `BDB` tables, because no row count is maintained in the table.
- The optimizer needs to know the approximate number of rows in the table. MySQL solves this by counting inserts and maintaining this in a separate segment in each `BDB` table. If you don't issue a lot of `DELETE` or `ROLLBACK` statements, this number should be accurate enough for the MySQL optimizer. However, MySQL stores the number only on close, so it may be incorrect if the server terminates unexpectedly. It should not be fatal even if this number is not 100% correct. You can update the row count by using `ANALYZE TABLE` or `OPTIMIZE TABLE`. See Section 13.7.2.1, “ANALYZE TABLE Syntax”, and Section 13.7.2.5, “OPTIMIZE TABLE Syntax”.
- Internal locking in `BDB` tables is done at the page level.
- `LOCK TABLES` works on `BDB` tables as with other tables. If you do not use `LOCK TABLES`, MySQL issues an internal multiple-write lock on the table (a lock that does not block other writers) to ensure that the table is properly locked if another thread issues a table lock.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- To support transaction rollback, the BDB storage engine maintains log files. For maximum performance, you can use the `--bdb-logdir` option to place the BDB logs on a different disk than the one where your databases are located.
- MySQL performs a checkpoint each time a new BDB log file is started, and removes any BDB log files that are not needed for current transactions. You can also use `FLUSH LOGS` at any time to checkpoint the Berkeley DB tables.

  For disaster recovery, you should use table backups plus MySQL's binary log. See Section 7.2, "Database Backup Methods".

  Warning

  If you delete old log files that are still in use, BDB is not able to do recovery at all and you may lose data if something goes wrong.
- Applications must always be prepared to handle cases where any change of a BDB table may cause an automatic rollback and any read may fail with a deadlock error.
- If you get a full disk with a BDB table, you get an error (probably error 28) and the transaction should roll back. This contrasts with MyISAM tables, for which `mysqld` waits for sufficient free disk space before continuing.

14.5.5 Restrictions on BDB Tables

The following list indicates restrictions that you must observe when using BDB tables:

- Each BDB table stores in its `.db` file the path to the file as it was created. This is done to enable detection of locks in a multi-user environment that supports symlinks. As a consequence of this, it is not possible to move BDB table files from one database directory to another.
- When making backups of BDB tables, you must either use `mysqldump` or else make a backup that includes the files for each BDB table (the `.frm` and `.db` files) as well as the BDB log files. The BDB storage engine stores unfinished transactions in its log files and requires them to be present when `mysqld` starts. The BDB logs are the files in the data directory with names of the form `log.NNNNNNNNN` (ten digits).
- If a column that permits NULL values has a unique index, only a single NULL value is permitted. This differs from other storage engines, which permit multiple NULL values in unique indexes.

14.5.6 Errors That May Occur When Using BDB Tables

- If the following error occurs when you start `mysqld` after upgrading, it means that the current version of BDB doesn't support the old log file format:

  ```
  bdb:  Ignoring log file: .../log.NNNNNNNNN:
  unsupported log version #
  ```

  In this case, you must delete all BDB logs from your data directory (the files that have names of the form `log.NNNNNNNNN`) and restart `mysqld`. We also recommend that you then use `mysqldump --opt` to dump your BDB tables, drop the tables, and restore them from the dump file.
- If autocommit mode is disabled and you drop a BDB table that is referenced in another transaction, you may get error messages of the following form in your MySQL error log:

  ```
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
001119 23:43:56  bdb:  Missing log fileid entry
001119 23:43:56  bdb:  txn_abort: Log undo failed for LSN:
                      1 3644744: Invalid
```

This is not fatal, but the fix is not trivial. Avoid dropping `BDB` tables except while autocommit mode is enabled.

14.6 The EXAMPLE Storage Engine

The `EXAMPLE` storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

The `EXAMPLE` storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `configure` with the `--with-example-storage-engine` option.

To examine the source for the `EXAMPLE` engine, look in the `sql/examples` directory of a MySQL source distribution.

When you create an `EXAMPLE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The `EXAMPLE` storage engine does not support indexing.

14.7 The FEDERATED Storage Engine

The `FEDERATED` storage engine is available beginning with MySQL 5.0.3. It is a storage engine that accesses data in tables of remote databases rather than in local tables.

The `FEDERATED` storage engine is available beginning with MySQL 5.0.3. This storage engine enables data to be accessed from a remote MySQL database on a local server without using replication or cluster technology. When using a `FEDERATED` table, queries on the local server are automatically executed on the remote (federated) tables. No data is stored on the local tables.

To include the `FEDERATED` storage engine if you build MySQL from source, invoke `configure` with the `--with-federated-storage-engine` option.

Beginning with MySQL 5.0.64, the `FEDERATED` storage engine is not enabled by default in the running server; to enable `FEDERATED`, you must start the MySQL server binary using the `--federated` option.

To examine the source for the `FEDERATED` engine, look in the `sql` directory of a source distribution for MySQL 5.0.3 or newer.

Additional Resources

- A forum dedicated to the `FEDERATED` storage engine is available at http://forums.mysql.com/list.php?105.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

14.7.1 Description of the FEDERATED Storage Engine

When you create a `FEDERATED` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. No other files are created, because the actual data is in a remote table. This differs from the way that storage engines for local tables work.

For local database tables, data files are local. For example, if you create a `MyISAM` table named `users`, the `MyISAM` handler creates a data file named `users.MYD`. A handler for local tables reads, inserts, deletes, and updates data in local data files, and rows are stored in a format particular to the handler. To read rows, the handler must parse data into columns. To write rows, column values must be converted to the row format used by the handler and written to the local data file.

With the MySQL `FEDERATED` storage engine, there are no local data files for a table (for example, there is no `.MYD` file). Instead, a remote database stores the data that normally would be in the table. The local server connects to a remote server, and uses the MySQL client API to read, delete, update, and insert data in the remote table. For example, data retrieval is initiated using a `SELECT * FROM tbl_name` SQL statement.

When a client issues an SQL statement that refers to a `FEDERATED` table, the flow of information between the local server (where the SQL statement is executed) and the remote server (where the data is physically stored) is as follows:

1. The storage engine looks through each column that the `FEDERATED` table has and constructs an appropriate SQL statement that refers to the remote table.
2. The statement is sent to the remote server using the MySQL client API.
3. The remote server processes the statement and the local server retrieves any result that the statement produces (an affected-rows count or a result set).
4. If the statement produces a result set, each column is converted to internal storage engine format that the `FEDERATED` engine expects and can use to display the result to the client that issued the original statement.

The local server communicates with the remote server using MySQL client C API functions. It invokes `mysql_real_query()` to send the statement. To read a result set, it uses `mysql_store_result()` and fetches rows one at a time using `mysql_fetch_row()`.

14.7.2 How to Use FEDERATED Tables

The procedure for using `FEDERATED` tables is very simple. Normally, you have two servers running, either both on the same host or on different hosts. (It is possible for a `FEDERATED` table to use another table that is managed by the same server, although there is little point in doing so.)

First, you must have a table on the remote server that you want to access by using a `FEDERATED` table. Suppose that the remote table is in the `federated` database and is defined like this:

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=MyISAM
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
DEFAULT CHARSET=latin1;
```

The example uses a MyISAM table, but the table could use any storage engine.

Next, create a FEDERATED table on the local server for accessing the remote table:

```
CREATE TABLE federated_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

(Before MySQL 5.0.13, use COMMENT rather than CONNECTION.)

The basic structure of this table should match that of the remote table, except that the ENGINE table option should be FEDERATED and the CONNECTION table option is a connection string that indicates to the FEDERATED engine how to connect to the remote server.

Note

You can improve the performance of a FEDERATED table by adding indexes to the table on the host. The optimization will occur because the query sent to the remote server will include the contents of the WHERE clause and will be sent to the remote server and subsequently executed locally. This reduces the network traffic that would otherwise request the entire table from the server for local processing.

The FEDERATED engine creates only the test_table.frm file in the federated database.

The remote host information indicates the remote server to which your local server connects, and the database and table information indicates which remote table to use as the data source. In this example, the remote server is indicated to be running as remote_host on port 9306, so there must be a MySQL server running on the remote host and listening to port 9306.

The general format of the connection string in the CONNECTION option is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Only mysql is supported as the *scheme* value at this point; the password and port number are optional.

Sample connection strings:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

The use of CONNECTION for specifying the connection string is nonoptimal and is likely to change in future. Keep this in mind for applications that use FEDERATED tables. Such applications are likely to need modification if the format for specifying connection information changes.

Because any password given in the connection string is stored as plain text, it can be seen by any user who can use SHOW CREATE TABLE or SHOW TABLE STATUS for the FEDERATED table, or query the TABLES table in the INFORMATION_SCHEMA database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

14.7.3 Limitations of the FEDERATED Storage Engine

The following items indicate features that the `FEDERATED` storage engine does and does not support:

- The remote server must be a MySQL server.
- The remote table that a `FEDERATED` table points to *must* exist before you try to access the table through the `FEDERATED` table.
- It is possible for one `FEDERATED` table to point to another, but you must be careful not to create a loop.
- There is no support for transactions.
- A `FEDERATED` table does not support indexes in the usual sense; because access to the table data is handled remotely, it is actually the remote table that makes use of indexes. This means that, for a query that cannot use any indexes and so requires a full table scan, the server fetches all rows from the remote table and filters them locally. This occurs regardless of any `WHERE` or `LIMIT` used with this `SELECT` statement; these clauses are applied locally to the returned rows.

  Queries that fail to use indexes can thus cause poor performance and network overload. In addition, since returned rows must be stored in memory, such a query can also lead to the local server swapping, or even hanging.
- Care should be taken when creating a `FEDERATED` table since the index definition from an equivalent `MyISAM` or other table may not be supported. For example, creating a `FEDERATED` table with an index prefix on `VARCHAR`, `TEXT` or `BLOB` columns will fail. The following definition in `MyISAM` is valid:

  ```
  CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=MYISAM;
  ```

  The key prefix in this example is incompatible with the `FEDERATED` engine, and the equivalent statement will fail:

  ```
  CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=FEDERATED
    CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
  ```

  If possible, you should try to separate the column and index definition when creating tables on both the remote server and the local server to avoid these index issues.
- Internally, the implementation uses `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `HANDLER`.
- The `FEDERATED` storage engine supports `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and indexes. It does not support `ALTER TABLE`, or any Data Definition Language statements that directly affect the structure of the table, other than `DROP TABLE`. The current implementation does not use prepared statements.
- `FEDERATED` accepts `INSERT ... ON DUPLICATE KEY UPDATE` statements, but if a duplicate-key violation occurs, the statement fails with an error.
- Performance on a `FEDERATED` table when performing bulk inserts (for example, on a `INSERT INTO ... SELECT ...` statement) is slower than with other table types because each selected row is treated as an individual `INSERT` statement on the federated table.
- Before MySQL 5.0.46, for a multiple-row insert into a `FEDERATED` table that refers to a remote transactional table, if the insert failed for a row due to constraint failure, the remote table would contain a partial commit (the rows preceding the failed one) instead of rolling back the statement completely. This occurred because the rows were treated as individual inserts.

  As of MySQL 5.0.46, `FEDERATED` performs bulk-insert handling such that multiple rows are sent to the remote table in a batch. This provides a performance improvement. Also, if the remote table is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

transactional, it enables the remote storage engine to perform statement rollback properly should an error occur. This capability has the following limitations:

- The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
- Bulk-insert handling does not occur for `INSERT ... ON DUPLICATE KEY UPDATE`.

- There is no way for the `FEDERATED` engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database system. The integrity of the data in the local table could be breached if there was any change to the remote database.
- Any `DROP TABLE` statement issued against a `FEDERATED` table drops only the local table, not the remote table.
- `FEDERATED` tables do not work with the query cache.

14.8 The ARCHIVE Storage Engine

The `ARCHIVE` storage engine is used for storing large amounts of data without indexes in a very small footprint.

The `ARCHIVE` storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `configure` with the `--with-archive-storage-engine` option.

To examine the source for the `ARCHIVE` engine, look in the `sql` directory of a MySQL source distribution.

You can check whether the `ARCHIVE` storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_archive';
```

When you create an `ARCHIVE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine creates other files, all having names beginning with the table name. The data and metadata files have extensions of `.ARZ` and `.ARM`, respectively. An `.ARN` file may appear during optimization operations.

The `ARCHIVE` engine supports `INSERT`, `REPLACE`, and `SELECT`, but not `DELETE` or `UPDATE`. It does support `ORDER BY` operations, `BLOB` columns, and basically all but spatial data types (see Section 11.5.1, “Spatial Data Types”). The `ARCHIVE` engine uses row-level locking.

Storage: Rows are compressed as they are inserted. The `ARCHIVE` engine uses `zlib` lossless data compression (see http://www.zlib.net/). You can use `OPTIMIZE TABLE` to analyze the table and pack it into a smaller format (for a reason to use `OPTIMIZE TABLE`, see later in this section). Beginning with MySQL 5.0.15, the engine also supports `CHECK TABLE`. There are several types of insertions that are used:

- An `INSERT` statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A `SELECT` forces a flush to occur, unless the only insertions that have come in were `INSERT DELAYED` (those flush as necessary). See Section 13.2.5.2, “INSERT DELAYED Syntax”.
- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A `SELECT` never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A `SELECT` operation performs a complete table scan: When a `SELECT` occurs, it finds out how many rows are currently

This documentation is for an older version. If you're
This documentation is for an older version. If you're

available and reads that number of rows. `SELECT` is performed as a consistent read. Note that lots of `SELECT` statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use `OPTIMIZE TABLE` or `REPAIR TABLE`. The number of rows in `ARCHIVE` tables reported by `SHOW TABLE STATUS` is always accurate. See Section 13.7.2.5, “OPTIMIZE TABLE Syntax”, Section 13.7.2.6, “REPAIR TABLE Syntax”, and Section 13.7.5.33, “SHOW TABLE STATUS Syntax”.

Additional Resources

- A forum dedicated to the `ARCHIVE` storage engine is available at http://forums.mysql.com/list.php?112.

14.9 The CSV Storage Engine

The `CSV` storage engine stores data in text files using comma-separated values format. It is unavailable on Windows until MySQL 5.1.

The `CSV` storage engine is included in MySQL binary distributions (except on Windows). To enable this storage engine if you build MySQL from source, invoke `configure` with the `--with-csv-storage-engine` option.

To examine the source for the `CSV` engine, look in the `sql/examples` directory of a MySQL source distribution.

When you create a `CSV` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine also creates a data file. Its name begins with the table name and has a `.CSV` extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
    -> ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+------+------------+
| i    | c          |
+------+------------+
|    1 | record one |
|    2 | record two |
+------+------------+
2 rows in set (0.00 sec)
```

If you examine the `test.CSV` file in the database directory created by executing the preceding statements, its contents should look like this:

```
"1","record one"
"2","record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

The `CSV` storage engine does not support indexing.

14.10 The BLACKHOLE Storage Engine

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The BLACKHOLE storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

The BLACKHOLE storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `configure` with the `--with-blackhole-storage-engine` option.

To examine the source for the BLACKHOLE engine, look in the `sql` directory of a MySQL source distribution.

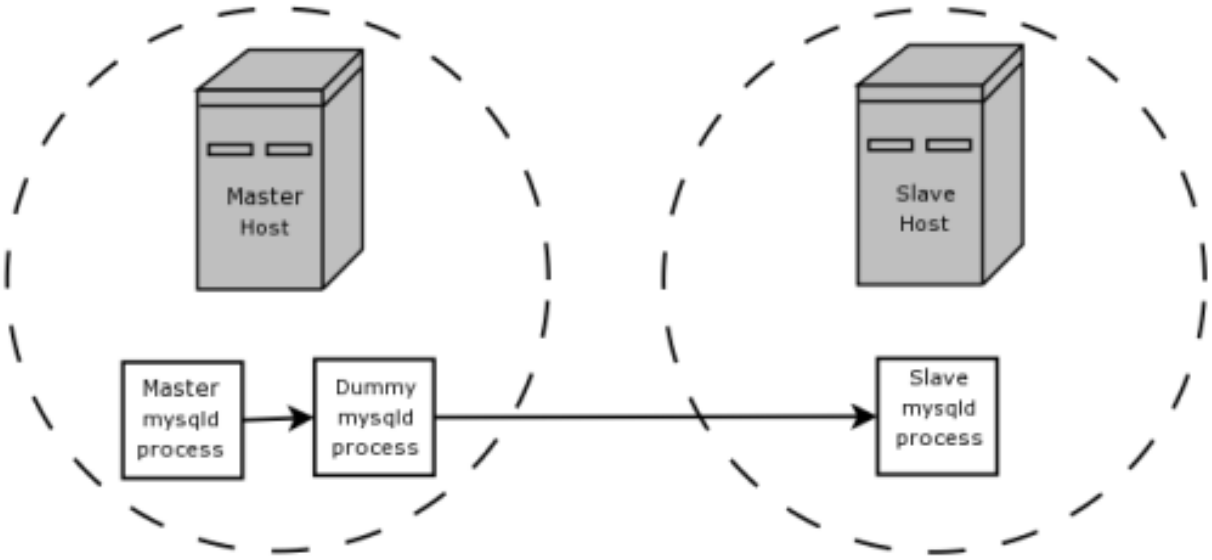
When you create a BLACKHOLE table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. There are no other files associated with the table.

The BLACKHOLE storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

You can check whether the BLACKHOLE storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_blackhole_engine';
```

Inserts into a BLACKHOLE table do not store any data, but if the binary log is enabled, the SQL statements are logged (and replicated to slave servers). This can be useful as a repeater or filter mechanism. Suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is BLACKHOLE, depicted as follows:



This documentation is for an older version. If you're
This documentation is for an older version. If you're

The master writes to its binary log. The “dummy” `mysqld` process acts as a slave, applying the desired combination of `replicate-do-*` and `replicate-ignore-*` rules, and writes a new, filtered binary log of its own. (See Section 16.1.2, “Replication and Binary Logging Options and Variables”.) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional `mysqld` process on the replication master host. This type of setup can be repeated with additional replication slaves.

`INSERT` triggers for `BLACKHOLE` tables work as expected. However, because the `BLACKHOLE` table does not actually store any data, `UPDATE` and `DELETE` triggers are not activated: The `FOR EACH ROW` clause in the trigger definition does not apply because there are no rows.

Other possible uses for the `BLACKHOLE` storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using `BLACKHOLE` with and without binary logging enabled.
- `BLACKHOLE` is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

Blackhole Engine and Auto Increment Columns

The Blackhole engine is a no-op engine. Any operations performed on a table using Blackhole will have no effect. This should be born in mind when considering the behavior of primary key columns that auto increment. The engine will not automatically increment field values, and does not retain auto increment field state. This has important implications in replication.

Consider the following replication scenario where all three of the following conditions apply:

1. On a master server there is a blackhole table with an auto increment field that is a primary key.
2. On a slave the same table exists but using the MyISAM engine.
3. Inserts are performed into the master's table without explicitly setting the auto increment value in the `INSERT` statement itself or through using a `SET INSERT_ID` statement.

In this scenario replication will fail with a duplicate entry error on the primary key column.

In statement based replication, the value of `INSERT_ID` in the context event will always be the same. Replication will therefore fail due to trying insert a row with a duplicate value for a primary key column.

In row based replication, the value that the engine returns for the row always be the same for each insert. This will result in the slave attempting to replay two insert log entries using the same value for the primary key column, and so replication will fail.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 15 High Availability and Scalability

Table of Contents

Data is the currency of today's web, mobile, social, enterprise and cloud applications. Ensuring data is always available is a top priority for any organization. Minutes of downtime can result in significant loss of revenue and reputation.

There is no “one size fits all” approach to delivering High Availability (HA). Unique application attributes, business requirements, operational capabilities and legacy infrastructure can all influence HA technology selection. And technology is only one element in delivering HA: people and processes are just as critical as the technology itself.

MySQL is deployed into many applications demanding availability and scalability. **Availability** refers to the ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Scalability refers to the ability to spread both the database and the load of your application queries across multiple MySQL servers.

Because each application has different operational and availability requirements, MySQL offers a range of certified and supported solutions, delivering the appropriate levels of High Availability (HA) and scalability to meet service level requirements. Such solutions extend from replication, through virtualization and geographically redundant, multi-data center solutions delivering 99.999% uptime.

Selecting the right high availability solution for an application largely depends on:

- The level of availability required.
- The type of application being deployed.
- Accepted best practices within your own environment.

The primary solutions supported by MySQL include:

- MySQL Replication. Learn more: Chapter 16, *Replication*.
- MySQL Fabric. Learn more: MySQL Fabric.
- MySQL Cluster. Learn more: Chapter 17, *MySQL Cluster*.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- Oracle Clusterware Agent for MySQL. Learn more about Oracle Clusterware.
- MySQL with Solaris Cluster. Learn more about Solaris Cluster.

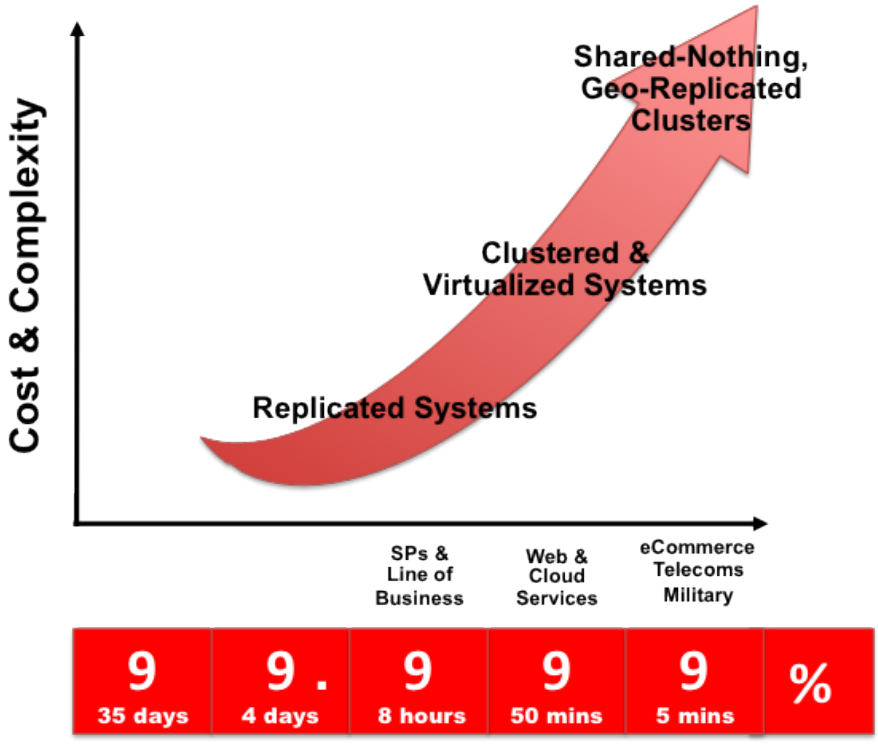
Further options are available using third-party solutions.

Each architecture used to achieve highly available database services is differentiated by the levels of uptime it offers. These architectures can be grouped into three main categories:

- Data Replication.
- Clustered & Virtualized Systems.
- Shared-Nothing, Geographically-Replicated Clusters.

As illustrated in the following figure, each of these architectures offers progressively higher levels of uptime, which must be balanced against potentially greater levels of cost and complexity that each can incur. Simply deploying a high availability architecture is not a guarantee of actually delivering HA. In fact, a poorly implemented and maintained shared-nothing cluster could easily deliver lower levels of availability than a simple data replication solution.

Figure 15.1 Tradeoffs: Cost and Complexity versus Availability



The following table compares the HA and Scalability capabilities of the various MySQL solutions:

Table 15.1 Feature Comparison of MySQL HA Solutions

Requirement	MySQL Replication	MySQL Cluster
Availability		
Platform Support	All Supported by MySQL Server (http://www.mysql.com/	All Supported by MySQL Cluster (http://www.mysql.com/support/supportedplatforms/cluster.html)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Requirement	MySQL Replication	MySQL Cluster
	support/supportedplatforms/database.html)	
Automated IP Failover	No	Depends on Connector and Configuration
Automated Database Failover	No	Yes
Automatic Data Resynchronization	No	Yes
Typical Failover Time	User / Script Dependent	1 Second and Less
Synchronous Replication	No, Asynchronous and Semisynchronous	Yes
Shared Storage	No, Distributed	No, Distributed
Geographic redundancy support	Yes	Yes, via MySQL Replication
Update Schema On-Line	No	Yes
Scalability		
Number of Nodes	One Master, Multiple Slaves	255
Built-in Load Balancing	Reads, via MySQL Replication	Yes, Reads and Writes
Supports Read-Intensive Workloads	Yes	Yes
Supports Write-Intensive Workloads	Yes, via Application-Level Sharding	Yes, via Auto-Sharding
Scale On-Line (add nodes, repartition, etc.)	No	Yes

15.1 Using MySQL within an Amazon EC2 Instance

The Amazon Elastic Compute Cloud (EC2) service provides virtual servers that you can build and deploy to run a variety of different applications and services, including MySQL. The EC2 service is based around the Xen framework, supporting x86, Linux based, platforms with individual instances of a virtual machine referred to as an Amazon Machine Image (AMI). You have complete (root) access to the AMI instance that you create, enabling you to configure and install your AMI in any way you choose.

To use EC2, you create an AMI based on the configuration and applications that you intend to use, and upload the AMI to the Amazon Simple Storage Service (S3). From the S3 resource, you can deploy one or more copies of the AMI to run as an instance within the EC2 environment. The EC2 environment provides management and control of the instance and contextual information about the instance while it is running.

Because you can create and control the AMI, the configuration, and the applications, you can deploy and create any environment you choose. This includes a basic MySQL server in addition to more extensive replication, HA and scalability scenarios that enable you to take advantage of the EC2 environment, and the ability to deploy additional instances as the demand for your MySQL services and applications grow.

To aid the deployment and distribution of work, three different Amazon EC2 instances are available, small (identified as `m1.small`), large (`m1.large`) and extra large (`m1.xlarge`). The different types provide different levels of computing power measured in EC2 computer units (ECU). A summary of the different instance configurations is shown in the following table.

EC2 Attribute	Small	Large	Extra Large
Platform	32-bit	64-bit	64-bit

This documentation is for an older version. If you're
This documentation is for an older version. If you're

EC2 Attribute	Small	Large	Extra Large
CPU cores	1	2	4
ECUs	1	4	8
RAM	1.7GB	7.5GB	15GB
Storage	150GB	840GB	1680GB
I/O Performance	Medium	High	High

The typical model for deploying and using MySQL within the EC2 environment is to create a basic AMI that you can use to hold your database data and application. Once the basic environment for your database and application has been created you can then choose to deploy the AMI to a suitable instance. Here the flexibility of having an AMI that can be re-deployed from the small to the large or extra large EC2 instance makes it easy to upgrade the hardware environment without rebuilding your application or database stack.

To get started with MySQL on EC2, including information on how to set up and install MySQL within an EC2 installation and how to port and migrate your data to the running instance, see Section 15.1.1, “Setting Up MySQL on an EC2 AMI”.

For tips and advice on how to create a scalable EC2 environment using MySQL, including guides on setting up replication, see Section 15.1.3, “Deploying a MySQL Database Using EC2”.

15.1.1 Setting Up MySQL on an EC2 AMI

There are many different ways of setting up an EC2 AMI with MySQL, including using any of the pre-configured AMIs supplied by Amazon.

The default *Getting Started* AMI provided by Amazon uses Fedora Core 4, and you can install MySQL by using `yum`:

```
shell> yum install mysql
```

This installs both the MySQL server and the Perl DBD::mysql driver for the Perl DBI API.

Alternatively, you can use one of the AMIs that include MySQL within the standard installation.

Finally, you can also install a standard version of MySQL downloaded from the MySQL Web site. The installation process and instructions are identical to any other installation of MySQL on Linux. See Chapter 2, *Installing and Upgrading MySQL*.

The standard configuration for MySQL places the data files in the default location, `/var/lib/mysql`. The default data directory on an EC2 instance is `/mnt` (although on the large and extra large instance you can alter this configuration). You must edit `/etc/my.cnf` to set the `datadir` option to point to the larger storage area.

Important

The first time you use the main storage location within an EC2 instance it needs to be initialized. The initialization process starts automatically the first time you write to the device. You can start using the device right away, but the write performance of the new device is significantly lower on the initial writes until the initialization process has finished.

To avoid this problem when setting up a new instance, you should start the initialization process before populating your MySQL database. One way to do this is to use `dd` to write to the file system:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
root-shell> dd if=/dev/zero of=initialize bs=1024M count=50
```

The preceding creates a 50GB on the file system and starts the initialization process. Delete the file once the process has finished.

The initialization process can be time-consuming. On the small instance, initialization takes between two and three hours. For the large and extra large drives, the initialization can be 10 or 20 hours, respectively.

In addition to configuring the correct storage location for your MySQL data files, also consider setting the following other settings in your instance before you save the instance configuration for deployment:

- Set the MySQL server ID, so that when you use it for replication, the ID information is set correctly.
- Enabling binary logging, so that replication can be initialized without starting and stopping the server.
- Set the caching and memory parameters for your storage engines. There are no limitations or restrictions on what storage engines you use in your EC2 environment. Choose a configuration, possibly using one of the standard configurations provided with MySQL appropriate for the instance on which you expect to deploy. The large and extra large instances have RAM that can be dedicated to caching. Be aware that if you choose to install `memcached` on the servers as part of your application stack you must ensure there is enough memory for both MySQL and `memcached`.

Once you have configured your AMI with MySQL and the rest of your application stack, save the AMI so that you can deploy and reuse the instance.

Once you have your application stack configured in an AMI, populating your MySQL database with data should be performed by creating a dump of your database using `mysqldump`, transferring the dump to the EC2 instance, and then reloading the information into the EC2 instance database.

Before using your instance with your application in a production situation, be aware of the limitations of the EC2 instance environment. See Section 15.1.2, “EC2 Instance Limitations”. To begin using your MySQL AMI, consult the notes on deployment. See Section 15.1.3, “Deploying a MySQL Database Using EC2”.

15.1.2 EC2 Instance Limitations

Be aware of the following limitations of the EC2 instances before deploying your applications. Although these shouldn't affect your ability to deploy within the Amazon EC2 environment, they may alter the way you setup and configure your environment to support your application.

- Data stored within instances is not persistent. If you create an instance and populate the instance with data, then the data only remains in place while the machine is running, and does not survive a reboot. If you shut down the instance, any data it contained is lost.

  To ensure that you do not lose information, take regular backups using `mysqldump`. If the data being stored is critical, consider using replication to keep a “live” backup of your data in the event of a failure. When creating a backup, write the data to the Amazon S3 service to avoid the transfer charges applied when copying data offsite.

- EC2 instances are not persistent. If the hardware on which an instance is running fails, the instance is shut down. This can lead to loss of data or service.

  However, if you use EBS, you can attach an EBS storage volume to an EC2 instance, and that EBS volume is persistent. Like a disk, an EBS volume can fail, but it is possible to create point-in-time snapshots of the volume. Snapshots are persisted to Amazon S3 and can be used to restore data in the event of volume failure.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- To replicate your EC2 instances to a non-EC2 environment, be aware of the transfer costs to and from the EC2 service. Data transfer between different EC2 instances is free, so using replication within the EC2 environment does not incur additional charges.
- Certain HA features are either not directly supported, or have limiting factors or problems that could reduce their utility. For example, using DRBD or MySQL Cluster might not work. The default storage configuration is also not redundant. You can use software-based RAID to improve redundancy, but this implies a further performance hit.

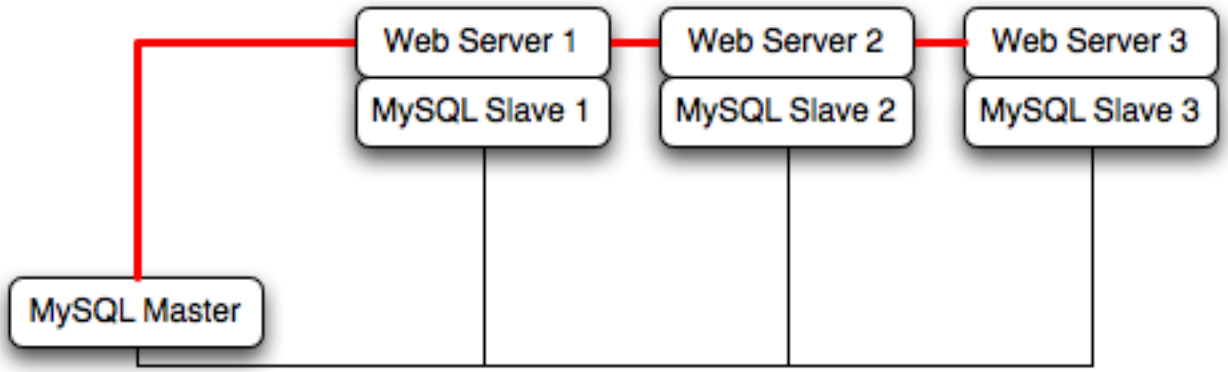
15.1.3 Deploying a MySQL Database Using EC2

Because you cannot guarantee the uptime and availability of your EC2 instances, when deploying MySQL within the EC2 environment, use an approach that enables you to easily distribute work among your EC2 instances. There are a number of ways of doing this. Using sharding techniques, where you split the application across multiple servers dedicating specific blocks of your dataset and users to different servers is an effective way of doing this. As a general rule, it is easier to create more EC2 instances to support more users than to upgrade the instance to a larger machine.

The EC2 architecture works best when you treat the EC2 instances as temporary, cache-based solutions, rather than as a long-term, high availability solution. In addition to using multiple machines, take advantage of other services, such as `memcached` to provide additional caching for your application to help reduce the load on the MySQL server so that it can concentrate on writes. On the large and extra large instances within EC2, the RAM available can provide a large memory cache for data.

Most types of scale-out topology that you would use with your own hardware can be used and applied within the EC2 environment. However, use the limitations and advice already given to ensure that any potential failures do not lose you any data. Also, because the relative power of each EC2 instance is so low, be prepared to alter your application to use sharding and add further EC2 instances to improve the performance of your application.

For example, take the typical scale-out environment shown following, where a single master replicates to one or more slaves (three in this example), with a web server running on each replication slave.



You can reproduce this structure completely within the EC2 environment, using an EC2 instance for the master, and one instance for each of the web and MySQL slave servers.

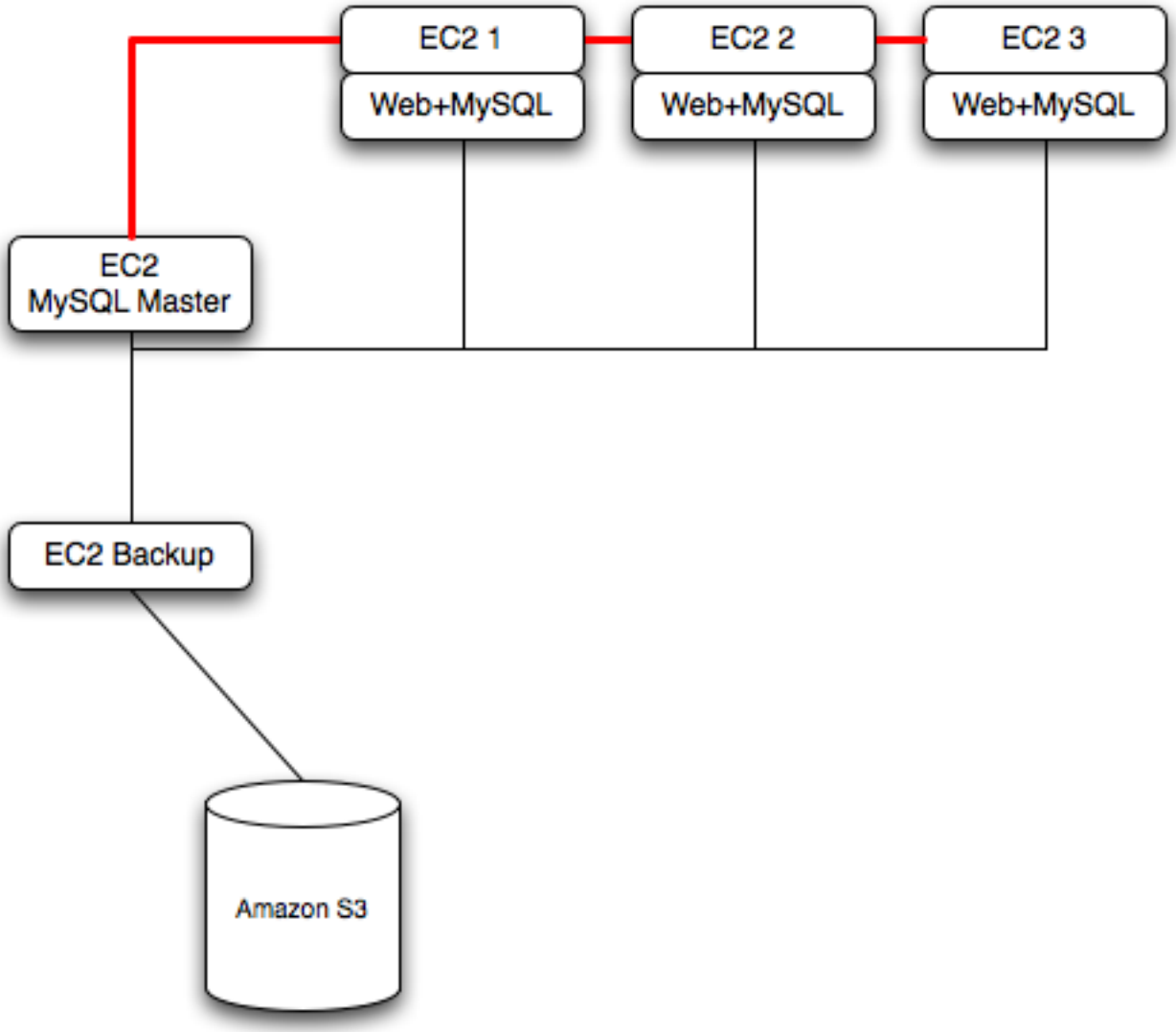
Note

Within the EC2 environment, internal (private) IP addresses used by the EC2 instances are constant. Always use these internal addresses and names when communicating between instances. Only use public IP addresses when

This documentation is for an older version. If you're

communicating with the outside world - for example, when publicizing your application.

To ensure reliability of your database, add at least one replication slave dedicated to providing an active backup and storage to the Amazon S3 facility. You can see an example of this in the following topology.



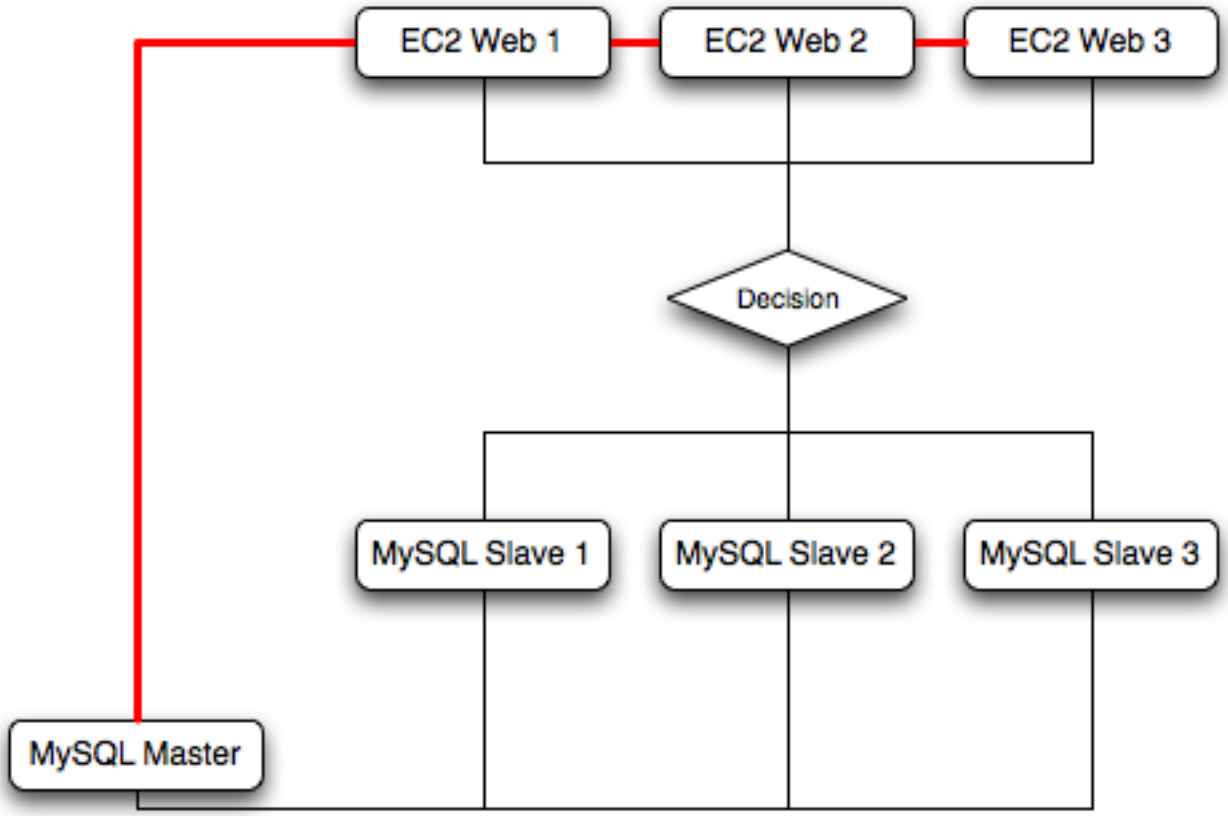
Using `memcached` within your EC2 instances should provide better performance. The large and extra large instances have a significant amount of RAM. To use `memcached` in your application, when loading information from the database, first check whether the item exists in the cache. If the data you are looking for exists in the cache, use it. If not, reload the data from the database and populate the cache.

Sharding divides up data in your entire database by allocating individual machines or machine groups to provide a unique set of data according to an appropriate group. For example, you might put all users with a surname ending in the letters A-D onto a single server. When a user connects to the application and their surname is known, queries can be redirected to the appropriate MySQL server.

When using sharding with EC2, separate the web server and MySQL server into separate EC2 instances, and then apply the sharding decision logic into your application. Once you know which MySQL server you

This documentation is for an older version. If you're
This documentation is for an older version. If you're

should be using for accessing the data you then distribute queries to the appropriate server. You can see a sample of this in the following illustration.



Warning

With sharding and EC2, be careful that the potential for failure of an instance does not affect your application. If the EC2 instance that provides the MySQL server for a particular shard fails, then all of the data on that shard becomes unavailable.

15.2 Using ZFS Replication

To support high availability environments, providing an instant copy of the information on both the currently active machine and the hot backup is a critical part of the HA solution. There are many solutions to this problem, such as Chapter 16, *Replication*.

The ZFS file system provides functionality to create a snapshot of the file system contents, transfer the snapshot to another machine, and extract the snapshot to recreate the file system. You can create a snapshot at any time, and you can create as many snapshots as you like. By continually creating, transferring, and restoring snapshots, you can provide synchronization between one or more machines in a fashion similar to DRBD.

The following example shows a simple Solaris system running with a single ZFS pool, mounted at `/scratchpool`:

```
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0d0s0        4.6G   3.7G   886M    82%    /
/devices                 0K     0K     0K     0%    /devices
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ctfs                     0K     0K     0K     0%    /system/contract
proc                     0K     0K     0K     0%    /proc
mnttab                   0K     0K     0K     0%    /etc/mnttab
swap                   1.4G   892K   1.4G     1%    /etc/svc/volatile
objfs                    0K     0K     0K     0%    /system/object
/usr/lib/libc/libc_hwcap1.so.1
                       4.6G   3.7G   886M    82%    /lib/libc.so.1
fd                       0K     0K     0K     0%    /dev/fd
swap                   1.4G    40K   1.4G     1%    /tmp
swap                   1.4G    28K   1.4G     1%    /var/run
/dev/dsk/c0d0s7         26G   913M    25G     4%    /export/home
scratchpool             16G    24K    16G     1%    /scratchpool
```

The MySQL data is stored in a directory on `/scratchpool`. To help demonstrate some of the basic replication functionality, there are also other items stored in `/scratchpool` as well:

```
total 17
drwxr-xr-x  31 root     bin           50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root     bin            5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root     sys           16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000     1000          40 Nov  6 19:16 emacs-22.1/
```

To create a snapshot of the file system, you use `zfs snapshot`, specifying the pool and the snapshot name:

```
root-shell> zfs snapshot scratchpool@snap1
```

To list the snapshots already taken:

```
root-shell> zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
scratchpool@snap1      0      -  24.5K  -
scratchpool@snap2      0      -  24.5K  -
```

The snapshots themselves are stored within the file system metadata, and the space required to keep them varies as time goes on because of the way the snapshots are created. The initial creation of a snapshot is very quick, because instead of taking an entire copy of the data and metadata required to hold the entire snapshot, ZFS records only the point in time and metadata of when the snapshot was created.

As more changes to the original file system are made, the size of the snapshot increases because more space is required to keep the record of the old blocks. If you create lots of snapshots, say one per day, and then delete the snapshots from earlier in the week, the size of the newer snapshots might also increase, as the changes that make up the newer state have to be included in the more recent snapshots, rather than being spread over the seven snapshots that make up the week.

You cannot directly back up the snapshots because they exist within the file system metadata rather than as regular files. To get the snapshot into a format that you can copy to another file system, tape, and so on, you use the `zfs send` command to create a stream version of the snapshot.

For example, to write the snapshot out to a file:

```
root-shell> zfs send scratchpool@snap1 >/backup/scratchpool-snap1
```

Or tape:

```
root-shell> zfs send scratchpool@snap1 >/dev/rmt/0
```

You can also write out the incremental changes between two snapshots using `zfs send`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
root-shell> zfs send scratchpool@snap1 scratchpool@snap2 >/backup/scratchpool-changes
```

To recover a snapshot, you use `zfs recv`, which applies the snapshot information either to a new file system, or to an existing one.

15.2.1 Using ZFS for File System Replication

Because `zfs send` and `zfs recv` use streams to exchange data, you can use them to replicate information from one system to another by combining `zfs send`, `ssh`, and `zfs recv`.

For example, to copy a snapshot of the `scratchpool` file system to a new file system called `slavepool` on a new server, you would use the following command. This sequence combines the snapshot of `scratchpool`, the transmission to the slave machine (using `ssh` with login credentials), and the recovery of the snapshot on the slave using `zfs recv`:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

The first part of the pipeline, `zfs send scratchpool@snap1`, streams the snapshot. The `ssh` command, and the command that it executes on the other server, `pfexec zfs recv -F slavepool`, receives the streamed snapshot data and writes it to slavepool. In this instance, I've specified the `-F` option which forces the snapshot data to be applied, and is therefore destructive. This is fine, as I'm creating the first version of my replicated file system.

On the slave machine, the replicated file system contains the exact same content:

```
root-shell> ls -al /slavepool/
total 23
drwxr-xr-x   6 root     root           7 Nov  8 09:13 ./
drwxr-xr-x  29 root     root          34 Nov  9 07:06 ../
drwxr-xr-x  31 root     bin           50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root     bin            5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root     sys           16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000     1000          40 Nov  6 19:16 emacs-22.1/
```

Once a snapshot has been created, to synchronize the file system again, you create a new snapshot and then use the incremental snapshot feature of `zfs send` to send the changes between the two snapshots to the slave machine again:

```
root-shell> zfs send -i scratchpool@snapshot1 scratchpool@snapshot2 |ssh id@host pfexec zfs recv slavepool
```

This operation only succeeds if the file system on the slave machine has not been modified at all. You cannot apply the incremental changes to a destination file system that has changed. In the example above, the `ls` command would cause problems by changing the metadata, such as the last access time for files or directories.

To prevent changes on the slave file system, set the file system on the slave to be read-only:

```
root-shell> zfs set readonly=on slavepool
```

Setting `readonly` means that you cannot change the file system on the slave by normal means, including the file system metadata. Operations that would normally update metadata (like our `ls`) silently perform their function without attempting to update the file system state.

In essence, the slave file system is nothing but a static copy of the original file system. However, even when configured to be read-only, a file system can have snapshots applied to it. With the file system set to read only, re-run the initial copy:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

Now you can make changes to the original file system and replicate them to the slave.

15.2.2 Configuring MySQL for ZFS Replication

Configuring MySQL on the source file system is a case of creating the data on the file system that you intend to replicate. The configuration file in the example below has been updated to use `/scratchpool/mysql-data` as the data directory, and now you can initialize the tables:

```
root-shell> mysql_install_db --defaults-file=/etc/mysql/5.5/my.cnf --user=mysql
```

To synchronize the initial information, perform a new snapshot and then send an incremental snapshot to the slave using `zfs send`:

```
root-shell> zfs snapshot scratchpool@snap2
root-shell> zfs send -i scratchpool@snap1 scratchpool@snap2|ssh id@host pfexec zfs recv slavepool
```

Doublecheck that the slave has the data by looking at the MySQL data directory on the `slavepool`:

```
root-shell> ls -al /slavepool/mysql-data/
```

Now you can start up MySQL, create some data, and then replicate the changes using `zfs send`/ `zfs recv` to the slave to synchronize the changes.

The rate at which you perform the synchronization depends on your application and environment. The limitation is the speed required to perform the snapshot and then to send the changes over the network.

To automate the process, create a script that performs the snapshot, send, and receive operation, and use `cron` to synchronize the changes at set times or intervals.

15.2.3 Handling MySQL Recovery with ZFS

When using ZFS replication to provide a constant copy of your data, ensure that you can recover your tables, either manually or automatically, in the event of a failure of the original system.

In the event of a failure, follow this sequence:

1. Stop the script on the master, if it is still up and running.
2. Set the slave file system to be read/write:

   ```
   root-shell> zfs set readonly=off slavepool
   ```

3. Start up `mysqld` on the slave. If you are using `InnoDB`, you get auto-recovery, if it is needed, to make sure the table data is correct, as shown here when I started up from our mid-INSERT snapshot:

   ```
   InnoDB: The log sequence number in ibdata files does not match
   InnoDB: the log sequence number in the ib_logfiles!
   081109 15:59:59  InnoDB: Database was not shut down normally!
   InnoDB: Starting crash recovery.
   InnoDB: Reading tablespace information from the .ibd files...
   InnoDB: Restoring possible half-written data pages from the doublewrite
   InnoDB: buffer...
   081109 16:00:03  InnoDB: Started; log sequence number 0 1142807951
   ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
081109 16:00:03 [Note] /slavepool/mysql-5.0.67-solaris10-i386/bin/mysqld: ready for connections.
Version: '5.0.67'  socket: '/tmp/mysql.sock'  port: 3306  MySQL Community Server (GPL)
```

Use InnoDB tables and a regular synchronization schedule to reduce the risk for significant data loss. On MyISAM tables, you might need to run REPAIR TABLE, and you might even have lost some information.

15.3 Using MySQL with memcached

memcached is a simple, highly scalable key-based cache that stores data and objects wherever dedicated or spare RAM is available for quick access by applications, without going through layers of parsing or disk I/O. To use, you run the memcached command on one or more hosts and then use the shared cache to store objects. For more usage instructions, see Section 15.3.2, “Using memcached”

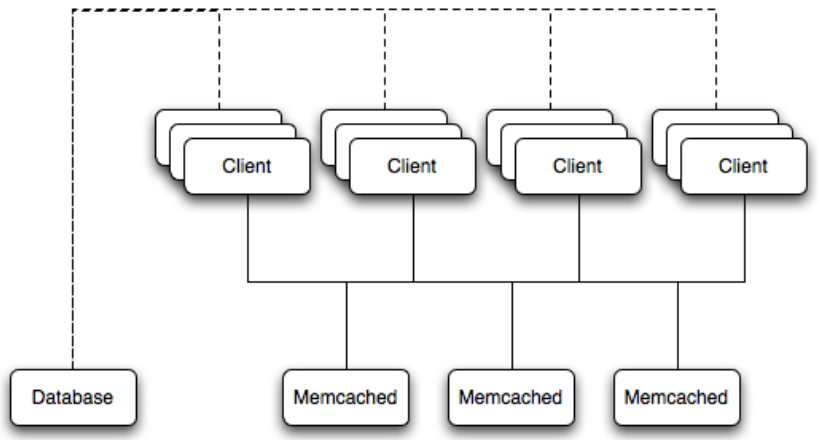
Benefits of using memcached include:

- Because all information is stored in RAM, the access speed is faster than loading the information each time from disk.
- Because the “value” portion of the key-value pair does not have any data type restrictions, you can cache data such as complex structures, documents, images, or a mixture of such things.
- If you use the in-memory cache to hold transient information, or as a read-only cache for information also stored in a database, the failure of any memcached server is not critical. For persistent data, you can fall back to an alternative lookup method using database queries, and reload the data into RAM on a different server.

The typical usage environment is to modify your application so that information is read from the cache provided by memcached. If the information is not in memcached, then the data is loaded from the MySQL database and written into the cache so that future requests for the same object benefit from the cached data.

For a typical deployment layout, see Figure 15.2, “memcached Architecture Overview”.

Figure 15.2 memcached Architecture Overview



In the example structure, any of the clients can contact one of the memcached servers to request a given key. Each client is configured to talk to all of the servers shown in the illustration. Within the client, when the request is made to store the information, the key used to reference the data is hashed and this hash is then used to select one of the memcached servers. The selection of the memcached server takes place on the client before the server is contacted, keeping the process lightweight.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The same algorithm is used again when a client requests the same key. The same key generates the same hash, and the same memcached server is selected as the source for the data. Using this method, the cached data is spread among all of the memcached servers, and the cached information is accessible from any client. The result is a distributed, memory-based, cache that can return information, particularly complex data and structures, much faster than natively reading the information from the database.

The data held within a traditional memcached server is never stored on disk (only in RAM, which means there is no persistence of data), and the RAM cache is always populated from the backing store (a MySQL database). If a memcached server fails, the data can always be recovered from the MySQL database.

15.3.1 Installing memcached

You can build and install memcached from the source code directly, or you can use an existing operating system package or installation.

Installing memcached from a Binary Distribution

To install memcached on a Red Hat, or Fedora host, use yum:

```
root-shell> yum install memcached
```

Note

On CentOS, you may be able to obtain a suitable RPM from another source, or use the source tarball.

To install memcached on a Debian or Ubuntu host, use apt-get:

```
root-shell> apt-get install memcached
```

To install memcached on a Gentoo host, use emerge:

```
root-shell> emerge install memcached
```

Building memcached from Source

On other Unix-based platforms, including Solaris, AIX, HP-UX and OS X, and Linux distributions not mentioned already, you must install from source. For Linux, make sure you have a 2.6-based kernel, which includes the improved epoll interface. For all platforms, ensure that you have libevent 1.1 or higher installed. You can obtain libevent from libevent web page.

You can obtain the source for memcached from memcached Web site.

To build memcached, follow these steps:

1. Extract the memcached source package:

   ```
   shell> gunzip -c memcached-1.2.5.tar.gz | tar xf -
   ```

2. Change to the memcached-*1.2.5* directory:

   ```
   shell> cd memcached-1.2.5
   ```

3. Run configure

   ```
   ```


This documentation is for an older version. If you're

```
shell> ./configure
```

Some additional options you might specify to the configure:

- --prefix

  To specify a different installation directory, use the --prefix option:

  ```
  shell> ./configure --prefix=/opt
  ```

  The default is to use the /usr/local directory.

- --with-libevent

  If you have installed libevent and configure cannot find the library, use the --with-libevent option to specify the location of the installed library.

- --enable-64bit

  To build a 64-bit version of memcached (which enables you to use a single instance with a large RAM allocation), use --enable-64bit.

- --enable-threads

  To enable multi-threading support in memcached, which improves the response times on servers with a heavy load, use --enable-threads. You must have support for the POSIX threads within your operating system to enable thread support. For more information on the threading support, see Section 15.3.2.7, “memcached Thread Support”.

- --enable-dtrace

  memcached includes a range of DTrace threads that can be used to monitor and benchmark a memcached instance. For more information, see Section 15.3.2.5, “Using memcached and DTrace”.

4. Run make to build memcached:

   ```
   shell> make
   ```

5. Run make install to install memcached:

   ```
   shell> make install
   ```

15.3.2 Using memcached

To start using memcached, start the memcached service on one or more servers. Running memcached sets up the server, allocates the memory and starts listening for connections from clients.

Note

You do not need to be a privileged user (root) to run memcached except to listen on one of the privileged TCP/IP ports (below 1024). You must, however, use a user that has not had their memory limits restricted using setrlimit or similar.

To start the server, run memcached as a nonprivileged (that is, non-root) user:

```
shell> memcached
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

By default, memcached uses the following settings:

- Memory allocation of 64MB
- Listens for connections on all network interfaces, using port 11211
- Supports a maximum of 1024 simultaneous connections

Typically, you would specify the full combination of options that you want when starting memcached, and normally provide a startup script to handle the initialization of memcached. For example, the following line starts memcached with a maximum of 1024MB RAM for the cache, listening on port 11211 on the IP address 192.168.0.110, running as a background daemon:

```
shell> memcached -d -m 1024 -p 11211 -l 192.168.0.110
```

To ensure that memcached is started up on boot, check the init script and configuration parameters.

memcached supports the following options:

- `-u user`

  If you start memcached as root, use the -u option to specify the user for executing memcached:

  ```
  shell> memcached -u memcache
  ```

- `-m memory`

  Set the amount of memory allocated to memcached for object storage. Default is 64MB.

  To increase the amount of memory allocated for the cache, use the -m option to specify the amount of RAM to be allocated (in megabytes). The more RAM you allocate, the more data you can store and therefore the more effective your cache is.

  Warning

  Do not specify a memory allocation larger than your available RAM. If you specify too large a value, then some RAM allocated for memcached uses swap space, and not physical RAM. This may lead to delays when storing and retrieving values, because data is swapped to disk, instead of storing the data directly in RAM.

  You can use the output of the vmstat command to get the free memory, as shown in free column:

  ```
  shell> vmstat
  kthr      memory            page            disk          faults      cpu
  r b w   swap  free  re  mf pi po fr de sr s1 s2 -- --   in   sy   cs us sy id
  0 0 0 5170504 3450392 2  7  2  0  0  0  4  0  0  0  0  296   54  199  0  0 100
  ```

  For example, to allocate 3GB of RAM:

  ```
  shell> memcached -m 3072
  ```

  On 32-bit x86 systems where you are using PAE to access memory above the 4GB limit, you cannot allocate RAM beyond the maximum process size. You can get around this by running multiple instances of memcached, each listening on a different port:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> memcached -m 1024 -p11211
shell> memcached -m 1024 -p11212
shell> memcached -m 1024 -p11213
```

Note

On all systems, particularly 32-bit, ensure that you leave enough room for both memcached application in addition to the memory setting. For example, if you have a dedicated memcached host with 4GB of RAM, do not set the memory size above 3500MB. Failure to do this may cause either a crash or severe performance issues.

- `-l interface`

  Specify a network interface/address to listen for connections. The default is to listen on all available address (INADDR_ANY).

  ```
  shell> memcached -l 192.168.0.110
  ```

  Support for IPv6 address support was added in memcached 1.2.5.

- `-p port`

  Specify the TCP port to use for connections. Default is 18080.

  ```
  shell> memcached -p 18080
  ```

- `-U port`

  Specify the UDP port to use for connections. Default is 11211, 0 switches UDP off.

  ```
  shell> memcached -U 18080
  ```

- `-s socket`

  Specify a Unix socket to listen on.

  If you are running memcached on the same server as the clients, you can disable the network interface and use a local Unix socket using the `-s` option:

  ```
  shell> memcached -s /tmp/memcached
  ```

  Using a Unix socket automatically disables network support, and saves network ports (allowing more ports to be used by your web server or other process).

- `-a mask`

  Specify the access mask to be used for the Unix socket, in octal. Default is 0700.

- `-c connections`

  Specify the maximum number of simultaneous connections to the memcached service. The default is 1024.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> memcached -c 2048
```

Use this option, either to reduce the number of connections (to prevent overloading memcached service) or to increase the number to make more effective use of the server running memcached server.

- -t threads

  Specify the number of threads to use when processing incoming requests.

  By default, memcached is configured to use 4 concurrent threads. The threading improves the performance of storing and retrieving data in the cache, using a locking system to prevent different threads overwriting or updating the same values. To increase or decrease the number of threads, use the -t option:

  ```
  shell> memcached -t 8
  ```

- -d

  Run memcached as a daemon (background) process:

  ```
  shell> memcached -d
  ```

- -r

  Maximize the size of the core file limit. In the event of a failure, this attempts to dump the entire memory space to disk as a core file, up to any limits imposed by setrlimit.

- -M

  Return an error to the client when the memory has been exhausted. This replaces the normal behavior of removing older items from the cache to make way for new items.

- -k

  Lock down all paged memory. This reserves the memory before use, instead of allocating new slabs of memory as new items are stored in the cache.

  Note

  There is a user-level limit on how much memory you can lock. Trying to allocate more than the available memory fails. You can set the limit for the user you started the daemon with (not for the -u user user) within the shell by using ulimit -S -l NUM_KB

- -v

  Verbose mode. Prints errors and warnings while executing the main event loop.

- -vv

  Very verbose mode. In addition to information printed by -v, also prints each client command and the response.

- -vvv

  Extremely verbose mode. In addition to information printed by -vv, also show the internal state transitions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- -h

  Print the help message and exit.

- -i

  Print the memcached and libevent license.

- -I mem

  Specify the maximum size permitted for storing an object within the memcached instance. The size supports a unit postfix (k for kilobytes, m for megabytes). For example, to increase the maximum supported object size to 32MB:

  ```
  shell> memcached -I 32m
  ```

  The maximum object size you can specify is 128MB, the default remains at 1MB.

  This option was added in 1.4.2.

- -b

  Set the backlog queue limit. The backlog queue configures how many network connections can be waiting to be processed by memcached. Increasing this limit may reduce errors received by the client that it is not able to connect to the memcached instance, but does not improve the performance of the server. The default is 1024.

- -P pidfile

  Save the process ID of the memcached instance into file.

- -f

  Set the chunk size growth factor. When allocating new memory chunks, the allocated size of new chunks is determined by multiplying the default slab size by this factor.

  To see the effects of this option without extensive testing, use the -vv command-line option to show the calculated slab sizes. For more information, see Section 15.3.2.8, “memcached Logs”.

- -n bytes

  The minimum space allocated for the key+value+flags information. The default is 48 bytes.

- -L

  On systems that support large memory pages, enables large memory page use. Using large memory pages enables memcached to allocate the item cache in one large chunk, which can improve the performance by reducing the number misses when accessing memory.

- -C

  Disable the use of compare and swap (CAS) operations.

  This option was added in memcached 1.3.x.

- -D char

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Set the default character to be used as a delimiter between the key prefixes and IDs. This is used for the per-prefix statistics reporting (see Section 15.3.4, “Getting memcached Statistics”). The default is the colon (:). If this option is used, statistics collection is turned on automatically. If not used, you can enable stats collection by sending the stats detail on command to the server.

This option was added in memcached 1.3.x.

- -R num

  Sets the maximum number of requests per event process. The default is 20.

- -B protocol

  Set the binding protocol, that is, the default memcached protocol support for client connections. Options are ascii, binary or auto. Automatic (auto) is the default.

  This option was added in memcached 1.4.0.

15.3.2.1 memcached Deployment

When using memcached you can use a number of different potential deployment strategies and topologies. The exact strategy to use depends on your application and environment. When developing a system for deploying memcached within your system, keep in mind the following points:

- memcached is only a caching mechanism. It shouldn't be used to store information that you cannot otherwise afford to lose and then load from a different location.
- There is no security built into the memcached protocol. At a minimum, make sure that the servers running memcached are only accessible from inside your network, and that the network ports being used are blocked (using a firewall or similar). If the information on the memcached servers that is being stored is any sensitive, then encrypt the information before storing it in memcached.
- memcached does not provide any sort of failover. Because there is no communication between different memcached instances. If an instance fails, your application must capable of removing it from the list, reloading the data and then writing data to another memcached instance.
- Latency between the clients and the memcached can be a problem if you are using different physical machines for these tasks. If you find that the latency is a problem, move the memcached instances to be on the clients.
- Key length is determined by the memcached server. The default maximum key size is 250 bytes.
- Try to use at least two memcached instances, especially for multiple clients, to avoid having a single point of failure. Ideally, create as many memcached nodes as possible. When adding and removing memcached instances from a pool, the hashing and distribution of key/value pairs may be affected. For information on how to avoid problems, see Section 15.3.2.4, “memcached Hashing/Distribution Types”.

15.3.2.2 Using Namespaces

The memcached cache is a very simple massive key/value storage system, and as such there is no way of compartmentalizing data automatically into different sections. For example, if you are storing information by the unique ID returned from a MySQL database, then storing the data from two different tables could run into issues because the same ID might be valid in both tables.

Some interfaces provide an automated mechanism for creating *namespaces* when storing information into the cache. In practice, these namespaces are merely a prefix before a given ID that is applied every time a value is stored or retrieve from the cache.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can implement the same basic principle by using keys that describe the object and the unique identifier within the key that you supply when the object is stored. For example, when storing user data, prefix the ID of the user with `user:` or `user-`.

Note

Using namespaces or prefixes only controls the keys stored/retrieved. There is no security within memcached, and therefore no way to enforce that a particular client only accesses keys with a particular namespace. Namespaces are only useful as a method of identifying data and preventing corruption of key/value pairs.

15.3.2.3 Data Expiry

There are two types of data expiry within a memcached instance. The first type is applied at the point when you store a new key/value pair into the memcached instance. If there is not enough space within a suitable slab to store the value, then an existing least recently used (LRU) object is removed (evicted) from the cache to make room for the new item.

The LRU algorithm ensures that the object that is removed is one that is either no longer in active use or that was used so long ago that its data is potentially out of date or of little value. However, in a system where the memory allocated to memcached is smaller than the number of regularly used objects required in the cache, a lot of expired items could be removed from the cache even though they are in active use. You use the statistics mechanism to get a better idea of the level of evictions (expired objects). For more information, see Section 15.3.4, “Getting memcached Statistics”.

You can change this eviction behavior by setting the `-M` command-line option when starting memcached. This option forces an error to be returned when the memory has been exhausted, instead of automatically evicting older data.

The second type of expiry system is an explicit mechanism that you can set when a key/value pair is inserted into the cache, or when deleting an item from the cache. Using an expiration time can be a useful way of ensuring that the data in the cache is up to date and in line with your application needs and requirements.

A typical scenario for explicitly setting the expiry time might include caching session data for a user when accessing a Web site. memcached uses a lazy expiry mechanism where the explicit expiry time that has been set is compared with the current time when the object is requested. Only objects that have not expired are returned.

You can also set the expiry time when explicitly deleting an object from the cache. In this case, the expiry time is really a timeout and indicates the period when any attempts to set the value for a given key are rejected.

15.3.2.4 memcached Hashing/Distribution Types

The memcached client interface supports a number of different distribution algorithms that are used in multi-server configurations to determine which host should be used when setting or getting data from a given memcached instance. When you get or set a value, a hash is constructed from the supplied key and then used to select a host from the list of configured servers. Because the hashing mechanism uses the supplied key as the basis for the hash, the same server is selected during both set and get operations.

You can think of this process as follows. Given an array of servers (a, b, and c), the client uses a hashing algorithm that returns an integer based on the key being stored or retrieved. The resulting value is then used to select a server from the list of servers configured in the client. Most standard client hashing within memcache clients uses a simple modulus calculation on the value against the number of configured memcached servers. You can summarize the process in pseudocode as:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
@memcservers = ['a.memc','b.memc','c.memc'];
$value = hash($key);
$chosen = $value % length(@memcservers);
```

Replacing the above with values:

```
@memcservers = ['a.memc','b.memc','c.memc'];
$value = hash('myid');
$chosen = 7009 % 3;
```

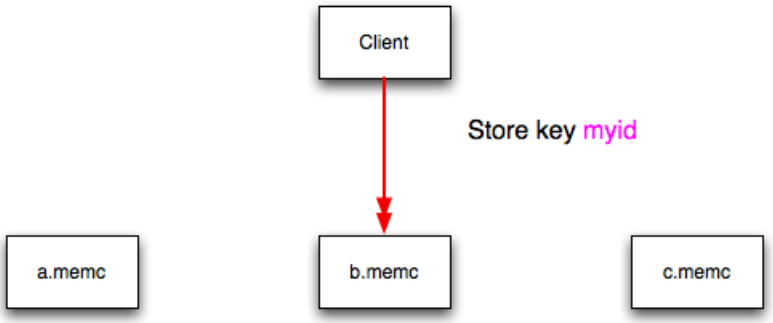
In the above example, the client hashing algorithm chooses the server at index 1 (`7009 % 3 = 1`), and store or retrieve the key and value with that server.

Note

This selection and hashing process is handled automatically by the memcached client you are using; you need only provide the list of memcached servers to use.

You can see a graphical representation of this below in Figure 15.3, “memcached Hash Selection”.

Figure 15.3 memcached Hash Selection



The same hashing and selection process takes place during any operation on the specified key within the memcached client.

Using this method provides a number of advantages:

- The hashing and selection of the server to contact is handled entirely within the client. This eliminates the need to perform network communication to determine the right machine to contact.
- Because the determination of the memcached server occurs entirely within the client, the server can be selected automatically regardless of the operation being executed (set, get, increment, etc.).
- Because the determination is handled within the client, the hashing algorithm returns the same value for a given key; values are not affected or reset by differences in the server environment.
- Selection is very fast. The hashing algorithm on the key value is quick and the resulting selection of the server is from a simple array of available machines.
- Using client-side hashing simplifies the distribution of data over each memcached server. Natural distribution of the values returned by the hashing algorithm means that keys are automatically spread over the available servers.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Providing that the list of servers configured within the client remains the same, the same stored key returns the same value, and therefore selects the same server.

However, if you do not use the same hashing mechanism then the same data may be recorded on different servers by different interfaces, both wasting space on your memcached and leading to potential differences in the information.

Note

One way to use a multi-interface compatible hashing mechanism is to use the libmemcached library and the associated interfaces. Because the interfaces for the different languages (including C, Ruby, Perl and Python) use the same client library interface, they always generate the same hash code from the ID.

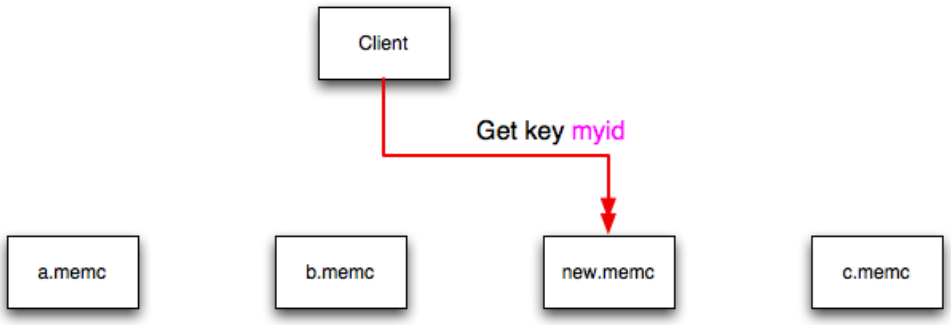
The problem with client-side selection of the server is that the list of the servers (including their sequential order) *must* remain consistent on each client using the memcached servers, and the servers must be available. If you try to perform an operation on a key when:

- A new memcached instance has been added to the list of available instances
- A memcached instance has been removed from the list of available instances
- The order of the memcached instances has changed

When the hashing algorithm is used on the given key, but with a different list of servers, the hash calculation may choose a different server from the list.

If a new memcached instance is added into the list of servers, as new.memc is in the example below, then a GET operation using the same key, myid, can result in a cache-miss. This is because the same value is computed from the key, which selects the same index from the array of servers, but index 2 now points to the new server, not the server c.memc where the data was originally stored. This would result in a cache miss, even though the key exists within the cache on another memcached instance.

Figure 15.4 memcached Hash Selection with New memcached instance



This means that servers c.memc and new.memc both contain the information for key myid, but the information stored against the key in eachs server may be different in each instance. A more significant problem is a much higher number of cache-misses when retrieving data, as the addition of a new server changes the distribution of keys, and this in turn requires rebuilding the cached data on the memcached instances, causing an increase in database reads.

The same effect can occur if you actively manage the list of servers configured in your clients, adding and removing the configured memcached instances as each instance is identified as being available. For example, removing a memcached instance when the client notices that the instance can no longer be contacted can cause the server selection to fail as described here.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To prevent this causing significant problems and invalidating your cache, you can select the hashing algorithm used to select the server. There are two common types of hashing algorithm, *consistent* and *modula*.

With *consistent* hashing algorithms, the same key when applied to a list of servers always uses the same server to store or retrieve the keys, even if the list of configured servers changes. This means that you can add and remove servers from the configure list and always use the same server for a given key. There are two types of consistent hashing algorithms available, Ketama and Wheel. Both types are supported by libmemcached, and implementations are available for PHP and Java.

Any consistent hashing algorithm has some limitations. When you add servers to an existing list of configured servers, keys are distributed to the new servers as part of the normal distribution. When you remove servers from the list, the keys are re-allocated to another server within the list, meaning that the cache needs to be re-populated with the information. Also, a consistent hashing algorithm does not resolve the issue where you want consistent selection of a server across multiple clients, but where each client contains a different list of servers. The consistency is enforced only within a single client.

With a *modula* hashing algorithm, the client selects a server by first computing the hash and then choosing a server from the list of configured servers. As the list of servers changes, so the server selected when using a modula hashing algorithm also changes. The result is the behavior described above; changes to the list of servers mean that different servers are selected when retrieving data, leading to cache misses and increase in database load as the cache is re-seeded with information.

If you use only a single memcached instance for each client, or your list of memcached servers configured for a client never changes, then the selection of a hashing algorithm is irrelevant, as it has no noticeable effect.

If you change your servers regularly, or you use a common set of servers that are shared among a large number of clients, then using a consistent hashing algorithm should help to ensure that your cache data is not duplicated and the data is evenly distributed.

15.3.2.5 Using memcached and DTrace

memcached includes a number of different DTrace probes that can be used to monitor the operation of the server. The probes included can monitor individual connections, slab allocations, and modifications to the hash table when a key/value pair is added, updated, or removed.

For more information on DTrace and writing DTrace scripts, read the DTrace User Guide.

Support for DTrace probes was added to memcached 1.2.6 includes a number of DTrace probes that can be used to help monitor your application. DTrace is supported on Solaris 10, OpenSolaris, OS X 10.5 and FreeBSD. To enable the DTrace probes in memcached, build from source and use the --enable-dtrace option. For more information, see Section 15.3.1, “Installing memcached”.

The probes supported by memcached are:

- conn-allocate(connid)

  Fired when a connection object is allocated from the connection pool.

  - connid: The connection ID.

- conn-release(connid)

  Fired when a connection object is released back to the connection pool.

  Arguments:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - connid: The connection ID.

- conn-create(ptr)

  Fired when a new connection object is being created (that is, there are no free connection objects in the connection pool).

  Arguments:

  - ptr: A pointer to the connection. object

- conn-destroy(ptr)

  Fired when a connection object is being destroyed.

  Arguments:

  - ptr: A pointer to the connection object.

- conn-dispatch(connid, threadid)

  Fired when a connection is dispatched from the main or connection-management thread to a worker thread.

  Arguments:

  - connid: The connection ID.
  - threadid: The thread ID.

- slabs-allocate(size, slabclass, slabsize, ptr)

  Allocate memory from the slab allocator.

  Arguments:

  - size: The requested size.
  - slabclass: The allocation is fulfilled in this class.
  - slabsize: The size of each item in this class.
  - ptr: A pointer to allocated memory.

- slabs-allocate-failed(size, slabclass)

  Failed to allocate memory (out of memory).

  Arguments:

  - size: The requested size.
  - slabclass: The class that failed to fulfill the request.

- slabs-slabclass-allocate(slabclass)

  Fired when a slab class needs more space.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Arguments:

- slabclass: The class that needs more memory.

- slabs-slabclass-allocate-failed(slabclass)

  Failed to allocate memory (out of memory).

  Arguments:

  - slabclass: The class that failed to grab more memory.

- slabs-free(size, slabclass, ptr)

  Release memory.

  Arguments:

  - size: The amount of memory to release, in bytes.
  - slabclass: The class the memory belongs to.
  - ptr: A pointer to the memory to release.

- assoc-find(key, depth)

  Fired when we have searched the hash table for a named key. These two elements provide an insight into how well the hash function operates. Traversals are a sign of a less optimal function, wasting CPU capacity.

  Arguments:

  - key: The key searched for.
  - depth: The depth in the list of hash table.

- assoc-insert(key, nokeys)

  Fired when a new item has been inserted.

  Arguments:

  - key: The key just inserted.
  - nokeys: The total number of keys currently being stored, including the key for which insert was called.

- assoc-delete(key, nokeys)

  Fired when a new item has been removed.

  Arguments:

  - key: The key just deleted.
  - nokeys: The total number of keys currently being stored, excluding the key for which delete was called.

- item-link(key, size)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Fired when an item is being linked in the cache.

Arguments:

- `key`: The items key.
- `size`: The size of the data.

- `item-unlink(key, size)`

  Fired when an item is being deleted.

  Arguments:

  - `key`: The items key.
  - `size`: The size of the data.

- `item-remove(key, size)`

  Fired when the refcount for an item is reduced.

  Arguments:

  - `key`: The item's key.
  - `size`: The size of the data.

- `item-update(key, size)`

  Fired when the "last referenced" time is updated.

  Arguments:

  - `key`: The item's key.
  - `size`: The size of the data.

- `item-replace(oldkey, oldsize, newkey, newsize)`

  Fired when an item is being replaced with another item.

  Arguments:

  - `oldkey`: The key of the item to replace.
  - `oldsize`: The size of the old item.
  - `newkey`: The key of the new item.
  - `newsize`: The size of the new item.

- `process-command-start(connid, request, size)`

  Fired when the processing of a command starts.

  Arguments:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- connid: The connection ID.
- request: The incoming request.
- size: The size of the request.

- process-command-end(connid, response, size)

  Fired when the processing of a command is done.

  Arguments:

  - connid: The connection ID.
  - response: The response to send back to the client.
  - size: The size of the response.

- command-get(connid, key, size)

  Fired for a get command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The size of the key's data (or -1 if not found).

- command-gets(connid, key, size, casid)

  Fired for a gets command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The size of the key's data (or -1 if not found).
  - casid: The casid for the item.

- command-add(connid, key, size)

  Fired for a add command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The new size of the key's data (or -1 if not found).

- command-set(connid, key, size)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Fired for a set command.

Arguments:

- connid: The connection ID.
- key: The requested key.
- size: The new size of the key's data (or -1 if not found).

- command-replace(connid, key, size)

  Fired for a replace command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The new size of the key's data (or -1 if not found).

- command-prepend(connid, key, size)

  Fired for a prepend command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The new size of the key's data (or -1 if not found).

- command-append(connid, key, size)

  Fired for a append command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The new size of the key's data (or -1 if not found).

- command-cas(connid, key, size, casid)

  Fired for a cas command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - size: The size of the key's data (or -1 if not found).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

    - casid: The cas ID requested.
- command-incr(connid, key, val)

  Fired for incr command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - val: The new value.
- command-decr(connid, key, val)

  Fired for decr command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - val: The new value.
- command-delete(connid, key, exptime)

  Fired for a delete command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - exptime: The expiry time.

15.3.2.6 Memory Allocation within memcached

When you first start memcached, the memory that you have configured is not automatically allocated. Instead, memcached only starts allocating and reserving physical memory once you start saving information into the cache.

When you start to store data into the cache, memcached does not allocate the memory for the data on an item by item basis. Instead, a slab allocation is used to optimize memory usage and prevent memory fragmentation when information expires from the cache.

With slab allocation, memory is reserved in blocks of 1MB. The slab is divided up into a number of blocks of equal size. When you try to store a value into the cache, memcached checks the size of the value that you are adding to the cache and determines which slab contains the right size allocation for the item. If a slab with the item size already exists, the item is written to the block within the slab.

If the new item is bigger than the size of any existing blocks, then a new slab is created, divided up into blocks of a suitable size. If an existing slab with the right block size already exists, but there are no free blocks, a new slab is created. If you update an existing item with data that is larger than the existing block allocation for that key, then the key is re-allocated into a suitable slab.

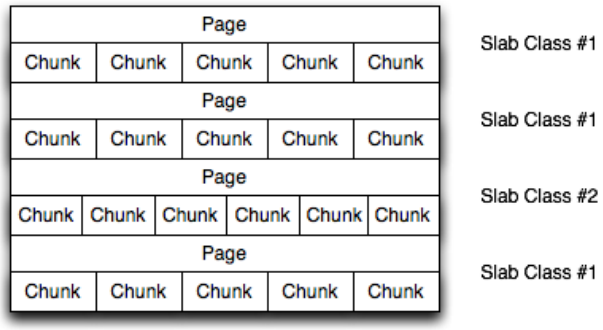
This documentation is for an older version. If you're
This documentation is for an older version. If you're

For example, the default size for the smallest block is 88 bytes (40 bytes of value, and the default 48 bytes for the key and flag data). If the size of the first item you store into the cache is less than 40 bytes, then a slab with a block size of 88 bytes is created and the value stored.

If the size of the data that you intend to store is larger than this value, then the block size is increased by the chunk size factor until a block size large enough to hold the value is determined. The block size is always a function of the scale factor, rounded up to a block size which is exactly divisible into the chunk size.

For a sample of the structure, see Figure 15.5, "Memory Allocation in memcached".

Figure 15.5 Memory Allocation in memcached



The result is that you have multiple pages allocated within the range of memory allocated to memcached. Each page is 1MB in size (by default), and is split into a different number of chunks, according to the chunk size required to store the key/value pairs. Each instance has multiple pages allocated, and a page is always created when a new item needs to be created requiring a chunk of a particular size. A slab may consist of multiple pages, and each page within a slab contains an equal number of chunks.

The chunk size of a new slab is determined by the base chunk size combined with the chunk size growth factor. For example, if the initial chunks are 104 bytes in size, and the default chunk size growth factor is used (1.25), then the next chunk size allocated would be the best power of 2 fit for 104*1.25, or 136 bytes.

Allocating the pages in this way ensures that memory does not get fragmented. However, depending on the distribution of the objects that you store, it may lead to an inefficient distribution of the slabs and chunks if you have significantly different sized items. For example, having a relatively small number of items within each chunk size may waste a lot of memory with just few chunks in each allocated page.

You can tune the growth factor to reduce this effect by using the `-f` command line option, which adapts the growth factor applied to make more effective use of the chunks and slabs allocated. For information on how to determine the current slab allocation statistics, see Section 15.3.4.2, "memcached Slabs Statistics".

If your operating system supports it, you can also start memcached with the `-L` command line option. This option preallocates all the memory during startup using large memory pages. This can improve performance by reducing the number of misses in the CPU memory cache.

15.3.2.7 memcached Thread Support

If you enable the thread implementation within when building memcached from source, then memcached uses multiple threads in addition to the libevent system to handle requests.

When enabled, the threading implementation operates as follows:

- Threading is handled by wrapping functions within the code to provide basic protection from updating the same global structures at the same time.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Each thread uses its own instance of the libevent to help improve performance.
- TCP/IP connections are handled with a single thread listening on the TCP/IP socket. Each connection is then distributed to one of the active threads on a simple round-robin basis. Each connection then operates solely within this thread while the connection remains open.
- For UDP connections, all the threads listen to a single UDP socket for incoming requests. Threads that are not currently dealing with another request ignore the incoming packet. One of the remaining, nonbusy, threads reads the request and sends the response. This implementation can lead to increased CPU load as threads wake from sleep to potentially process the request.

Using threads can increase the performance on servers that have multiple CPU cores available, as the requests to update the hash table can be spread between the individual threads. To minimize overhead from the locking mechanism employed, experiment with different thread values to achieve the best performance based on the number and type of requests within your given workload.

15.3.2.8 memcached Logs

If you enable verbose mode, using the -v, -vv, or -vvv options, then the information output by memcached includes details of the operations being performed.

Without the verbose options, memcached normally produces no output during normal operating.

- **Output when using -v**

  The lowest verbosity level shows you:

  - Errors and warnings
  - Transient errors
  - Protocol and socket errors, including exhausting available connections
  - Each registered client connection, including the socket descriptor number and the protocol used.

    For example:

    ```
    32: Client using the ascii protocol
    33: Client using the ascii protocol
    ```

    The socket descriptor is only valid while the client remains connected. Non-persistent connections may not be effectively represented.

  Examples of the error messages output at this level include:

  ```
  <%d send buffer was %d, now %d
  Can't listen for events on fd %d
  Can't read from libevent pipe
  Catastrophic: event fd doesn't match conn fd!
  Couldn't build response
  Couldn't realloc input buffer
  Couldn't update event
  Failed to build UDP headers
  Failed to read, and not due to blocking
  Too many open connections
  Unexpected state %d
  ```

- **Output when using -vv**

This documentation is for an older version. If you're

When using the second level of verbosity, you get more detailed information about protocol operations, keys updated, chunk and network operatings and details.

During the initial start-up of memcached with this level of verbosity, you are shown the sizes of the individual slab classes, the chunk sizes, and the number of entries per slab. These do not show the allocation of the slabs, just the slabs that would be created when data is added. You are also given information about the listen queues and buffers used to send information. A sample of the output generated for a TCP/IP based system with the default memory and growth factors is given below:

```
shell> memcached -vv
slab class   1: chunk size     80 perslab 13107
slab class   2: chunk size    104 perslab 10082
slab class   3: chunk size    136 perslab  7710
slab class   4: chunk size    176 perslab  5957
slab class   5: chunk size    224 perslab  4681
slab class   6: chunk size    280 perslab  3744
slab class   7: chunk size    352 perslab  2978
slab class   8: chunk size    440 perslab  2383
slab class   9: chunk size    552 perslab  1899
slab class  10: chunk size    696 perslab  1506
slab class  11: chunk size    872 perslab  1202
slab class  12: chunk size   1096 perslab   956
slab class  13: chunk size   1376 perslab   762
slab class  14: chunk size   1720 perslab   609
slab class  15: chunk size   2152 perslab   487
slab class  16: chunk size   2696 perslab   388
slab class  17: chunk size   3376 perslab   310
slab class  18: chunk size   4224 perslab   248
slab class  19: chunk size   5280 perslab   198
slab class  20: chunk size   6600 perslab   158
slab class  21: chunk size   8256 perslab   127
slab class  22: chunk size  10320 perslab   101
slab class  23: chunk size  12904 perslab    81
slab class  24: chunk size  16136 perslab    64
slab class  25: chunk size  20176 perslab    51
slab class  26: chunk size  25224 perslab    41
slab class  27: chunk size  31536 perslab    33
slab class  28: chunk size  39424 perslab    26
slab class  29: chunk size  49280 perslab    21
slab class  30: chunk size  61600 perslab    17
slab class  31: chunk size  77000 perslab    13
slab class  32: chunk size  96256 perslab    10
slab class  33: chunk size 120320 perslab     8
slab class  34: chunk size 150400 perslab     6
slab class  35: chunk size 188000 perslab     5
slab class  36: chunk size 235000 perslab     4
slab class  37: chunk size 293752 perslab     3
slab class  38: chunk size 367192 perslab     2
slab class  39: chunk size 458992 perslab     2
<26 server listening (auto-negotiate)
<29 server listening (auto-negotiate)
<30 send buffer was 57344, now 2097152
<31 send buffer was 57344, now 2097152
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Using this verbosity level can be a useful way to check the effects of the growth factor used on slabs with different memory allocations, which in turn can be used to better tune the growth factor to suit the data you are storing in the cache. For example, if you set the growth factor to 4 (quadrupling the size of each slab):

```
shell> memcached -f 4 -m 1g -vv
slab class   1: chunk size     80 perslab 13107
slab class   2: chunk size    320 perslab  3276
slab class   3: chunk size   1280 perslab   819
slab class   4: chunk size   5120 perslab   204
slab class   5: chunk size  20480 perslab    51
slab class   6: chunk size  81920 perslab    12
slab class   7: chunk size 327680 perslab     3
...
```

During use of the cache, this verbosity level also prints out detailed information on the storage and recovery of keys and other information. An example of the output during a typical set/get and increment/decrement operation is shown below.

```
32: Client using the ascii protocol
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key
>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key1 1 36

>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
```

During client communication, for each line, the initial character shows the direction of flow of the information. The < for communication from the client to the memcached server and > for communication back to the client. The number is the numeric socket descriptor for the connection.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- **Output when using `-vvv`**

  This level of verbosity includes the transitions of connections between different states in the event library while reading and writing content to/from the clients. It should be used to diagnose and identify issues in client communication. For example, you can use this information to determine if `memcached` is taking a long time to return information to the client, during the read of the client operation or before returning and completing the operation. An example of the typical sequence for a set operation is provided below:

```
<32 new auto-negotiating client connection
32: going from conn_new_cmd to conn_waiting
32: going from conn_waiting to conn_read
32: going from conn_read to conn_parse_cmd
32: Client using the ascii protocol
<32 set my_key 0 0 10
32: going from conn_parse_cmd to conn_nread
> NOT FOUND my_key
>32 STORED
32: going from conn_nread to conn_write
32: going from conn_write to conn_new_cmd
32: going from conn_new_cmd to conn_waiting
32: going from conn_waiting to conn_read
32: going from conn_read to conn_closing
<32 connection closed.
```

All of the verbosity levels in `memcached` are designed to be used during debugging or examination of issues. The quantity of information generated, particularly when using `-vvv`, is significant, particularly on a busy server. Also be aware that writing the error information out, especially to disk, may negate some of the performance gains you achieve by using `memcached`. Therefore, use in production or deployment environments is not recommended.

15.3.3 Developing a `memcached` Application

A number of language interfaces let applications store and retrieve information with `memcached` servers. You can write `memcached` applications in popular languages such as Perl, PHP, Python, Ruby, C, and Java.

Data stored into a `memcached` server is referred to by a single string (the key), with storage into the cache and retrieval from the cache using the key as the reference. The cache therefore operates like a large associative array or hash table. It is not possible to structure or otherwise organize the information stored in the cache. To emulate database notions such as multiple tables or composite key values, you must encode the extra information into the strings used as keys. For example, to store or look up the address corresponding to a specific latitude and longitude, you might turn those two numeric values into a single comma-separated string to use as a key.

15.3.3.1 Basic `memcached` Operations

The interface to `memcached` supports the following methods for storing and retrieving information in the cache, and these are consistent across all the different APIs, although the language specific mechanics might be different:

- `get(key)`: Retrieves information from the cache. Returns the value associated with the key if the specified key exists. Returns `NULL`, `nil`, `undefined`, or the closest equivalent in the corresponding language, if the specified key does not exist.

- `set(key, value [, expiry])`: Sets the item associated with a key in the cache to the specified value. This either updates an existing item if the key already exists, or adds a new key/value pair if the key doesn't exist. If the expiry time is specified, then the item expires (and is deleted) when the expiry

This documentation is for an older version. If you're
This documentation is for an older version. If you're

time is reached. The time is specified in seconds, and is taken as a relative time if the value is less than 30 days (30*24*60*60), or an absolute time (epoch) if larger than this value.

- `add(key, value [, expiry])`: Adds the key and associated value to the cache, if the specified key does not already exist.
- `replace(key, value [, expiry])`: Replaces the item associated with the specified `key`, only if the key already exists. The new value is given by the `value` parameter.
- `delete(key [, time])`: Deletes the `key` and its associated item from the cache. If you supply a `time`, then adding another item with the specified `key` is blocked for the specified period.
- `incr(key , value)`: Increments the item associated with the `key` by the specified `value`.
- `decr(key , value)`: Decrements the item associated with the `key` by the specified `value`.
- `flush_all`: Invalidates (or expires) all the current items in the cache. Technically they still exist (they are not deleted), but they are silently destroyed the next time you try to access them.

In all implementations, most or all of these functions are duplicated through the corresponding native language interface.

When practical, use `memcached` to store full items, rather than caching a single column value from the database. For example, when displaying a record about an object (invoice, user history, or blog post), load all the data for the associated entry from the database, and compile it into the internal structure that would normally be required by the application. Save the complete object in the cache.

Complex data structures cannot be stored directly. Most interfaces serialize the data for you, that is, put it in a textual form that can reconstruct the original pointers and nesting. Perl uses `Storable`, PHP uses `serialize`, Python uses `cPickle` (or `Pickle`) and Java uses the `Serializable` interface. In most cases, the serialization interface used is customizable. To share data stored in `memcached` instances between different language interfaces, consider using a common serialization solution such as JSON (Javascript Object Notation).

15.3.3.2 Using `memcached` as a MySQL Caching Layer

When using `memcached` to cache MySQL data, your application must retrieve data from the database and load the appropriate key-value pairs into the cache. Then, subsequent lookups can be done directly from the cache.

Because MySQL has its own in-memory caching mechanisms for queried data, such as the `InnoDB` buffer pool and the MySQL query cache, look for opportunities beyond loading individual column values or rows into the cache. Prefer to cache composite values, such as those retrieved from multiple tables through a join query, or result sets assembled from multiple rows.

Caution

Limit the information in the cache to non-sensitive data, because there is no security required to access or update the information within a `memcached` instance. Anybody with access to the machine has the ability to read, view and potentially update the information. To keep the data secure, encrypt the information before caching it. To restrict the users capable of connecting to the server, either disable network access, or use IPTables or similar techniques to restrict access to the `memcached` ports to a select set of hosts.

You can introduce `memcached` to an existing application, even if caching was not part of the original design. In many languages and environments the changes to the application will be just a few lines, first

This documentation is for an older version. If you're
This documentation is for an older version. If you're

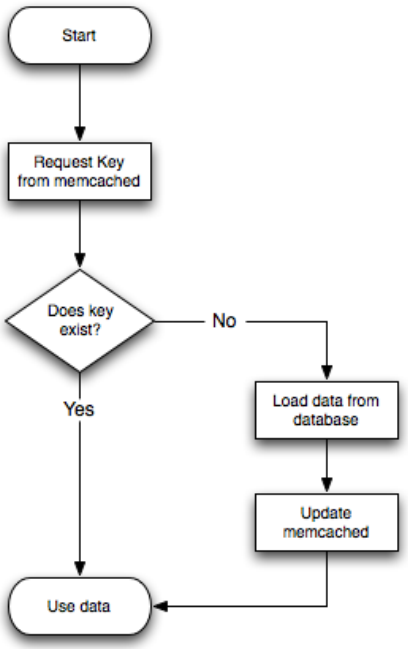
to attempt to read from the cache when loading data, fall back to the old method if the information is not cached, and to update the cache with information once the data has been read.

The general sequence for using memcached in any language as a caching solution for MySQL is as follows:

1. Request the item from the cache.
2. If the item exists, use the item data.
3. If the item does not exist, load the data from MySQL, and store the value into the cache. This means the value is available to the next client that requests it from the cache.

For a flow diagram of this sequence, see Figure 15.6, “Typical memcached Application Flowchart”.

Figure 15.6 Typical memcached Application Flowchart



Adapting Database Best Practices to memcached Applications

The most direct way to cache MySQL data is to use a 2-column table, where the first column is a primary key. Because of the uniqueness requirements for memcached keys, make sure your database schema makes appropriate use of primary keys and unique constraints.

If you combine multiple column values into a single memcached item value, choose data types to make it easy to parse the value back into its components, for example by using a separator character between numeric values.

The queries that map most easily to memcached lookups are those with a single WHERE clause, using an = or IN operator. For complicated WHERE clauses, or those using operators such as <, >, BETWEEN, or LIKE, memcached does not provide a simple or efficient way to scan through or filter the keys or associated values, so typically you perform those operations as SQL queries on the underlying database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

15.3.3.3 Using libmemcached with C and C++

The libmemcached library provides both C and C++ interfaces to memcached and is also the basis for a number of different additional API implementations, including Perl, Python and Ruby. Understanding the core libmemcached functions can help when using these other interfaces.

The C library is the most comprehensive interface library for memcached and provides functions and operational systems not always exposed in interfaces not based on the libmemcached library.

The different functions can be divided up according to their basic operation. In addition to functions that interface to the core API, a number of utility functions provide extended functionality, such as appending and prepending data.

To build and install libmemcached, download the libmemcached package, run configure, and then build and install:

```
shell> tar xjf libmemcached-0.21.tar.gz
shell> cd libmemcached-0.21
shell> ./configure
shell> make
shell> make install
```

On many Linux operating systems, you can install the corresponding libmemcached package through the usual yum, apt-get, or similar commands.

To build an application that uses the library, first set the list of servers. Either directly manipulate the servers configured within the main memcached_st structure, or separately populate a list of servers, and then add this list to the memcached_st structure. The latter method is used in the following example. Once the server list has been set, you can call the functions to store or retrieve data. A simple application for setting a preset value to localhost is provided here:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *key= "keystring";
  char *value= "keyvalue";

  memcached_server_st *memcached_servers_parse (char *server_strings);
  memc= memcached_create(NULL);

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));

  rc= memcached_set(memc, key, strlen(key), value, strlen(value), (time_t)0, (uint32_t)0);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Key stored successfully\n");
  else
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));

  return 0;
}
```

To test the success of an operation, use the return value, or populated result code, for a given function. The value is always set to MEMCACHED_SUCCESS if the operation succeeded. In the event of a failure, use the memcached_strerror() function to translate the result code into a printable string.

To build the application, specify the memcached library:

```
shell> gcc -o memc_basic memc_basic.c -lmemcached
```

Running the above sample application, after starting a memcached server, should return a success message:

```
shell> memc_basic
Added server successfully
Key stored successfully
```

libmemcached Base Functions

The base libmemcached functions let you create, destroy and clone the main memcached_st structure that is used to interface with the memcached servers. The main functions are defined below:

```
memcached_st *memcached_create (memcached_st *ptr);
```

Creates a new memcached_st structure for use with the other libmemcached API functions. You can supply an existing, static, memcached_st structure, or NULL to have a new structured allocated. Returns a pointer to the created structure, or NULL on failure.

```
void memcached_free (memcached_st *ptr);
```

Frees the structure and memory allocated to a previously created memcached_st structure.

```
memcached_st *memcached_clone(memcached_st *clone, memcached_st *source);
```

Clones an existing memcached structure from the specified source, copying the defaults and list of servers defined in the structure.

libmemcached Server Functions

The libmemcached API uses a list of servers, stored within the memcached_server_st structure, to act as the list of servers used by the rest of the functions. To use memcached, you first create the server list, and then apply the list of servers to a valid libmemcached object.

Because the list of servers, and the list of servers within an active libmemcached object can be manipulated separately, you can update and manage server lists while an active libmemcached interface is running.

The functions for manipulating the list of servers within a memcached_st structure are:

```
memcached_return
   memcached_server_add (memcached_st *ptr,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
                          char *hostname,
                          unsigned int port);
```

Adds a server, using the given `hostname` and `port` into the `memcached_st` structure given in `ptr`.

```
memcached_return
   memcached_server_add_unix_socket (memcached_st *ptr,
                                     char *socket);
```

Adds a Unix socket to the list of servers configured in the `memcached_st` structure.

```
unsigned int memcached_server_count (memcached_st *ptr);
```

Returns a count of the number of configured servers within the `memcached_st` structure.

```
memcached_server_st *
   memcached_server_list (memcached_st *ptr);
```

Returns an array of all the defined hosts within a `memcached_st` structure.

```
memcached_return
   memcached_server_push (memcached_st *ptr,
                          memcached_server_st *list);
```

Pushes an existing list of servers onto list of servers configured for a current `memcached_st` structure. This adds servers to the end of the existing list, and duplicates are not checked.

The `memcached_server_st` structure can be used to create a list of `memcached` servers which can then be applied individually to `memcached_st` structures.

```
memcached_server_st *
   memcached_server_list_append (memcached_server_st *ptr,
                                 char *hostname,
                                 unsigned int port,
                                 memcached_return *error);
```

Adds a server, with `hostname` and `port`, to the server list in `ptr`. The result code is handled by the `error` argument, which should point to an existing `memcached_return` variable. The function returns a pointer to the returned list.

```
unsigned int memcached_server_list_count (memcached_server_st *ptr);
```

Returns the number of the servers in the server list.

```
void memcached_server_list_free (memcached_server_st *ptr);
```

Frees the memory associated with a server list.

```
memcached_server_st *memcached_servers_parse (char *server_strings);
```

Parses a string containing a list of servers, where individual servers are separated by a comma, space, or both, and where individual servers are of the form *`server[:port]`*. The return value is a server list structure.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

libmemcached Set Functions

The set-related functions within libmemcached provide the same functionality as the core functions supported by the memcached protocol. The full definition for the different functions is the same for all the base functions (add, replace, prepend, append). For example, the function definition for memcached_set() is:

```
memcached_return
   memcached_set (memcached_st *ptr,
                  const char *key,
                  size_t key_length,
                  const char *value,
                  size_t value_length,
                  time_t expiration,
                  uint32_t flags);
```

The ptr is the memcached_st structure. The key and key_length define the key name and length, and value and value_length the corresponding value and length. You can also set the expiration and optional flags. For more information, see Controlling libmemcached Behaviors.

This table outlines the remainder of the set-related libmemcached functions and the equivalent core functions supported by the memcached protocol.

libmemcached Function	Equivalent Core Function
memcached_set(memc, key, key_length, value, value_length, expiration, flags)	Generic set() operation.
memcached_add(memc, key, key_length, value, value_length, expiration, flags)	Generic add() function.
memcached_replace(memc, key, key_length, value, value_length, expiration, flags)	Generic replace().
memcached_prepend(memc, key, key_length, value, value_length, expiration, flags)	Prepends the specified value before the current value of the specified key.
memcached_append(memc, key, key_length, value, value_length, expiration, flags)	Appends the specified value after the current value of the specified key.
memcached_cas(memc, key, key_length, value, value_length, expiration, flags, cas)	Overwrites the data for a given key as long as the corresponding cas value is still the same within the server.
memcached_set_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic set(), but has the option of an additional master key that can be used to identify an individual server.
memcached_add_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic add(), but has the option of an additional master key that can be used to identify an individual server.
memcached_replace_by_key(memc, master_key, master_key_length, key,	Similar to the generic replace(), but has the option of an additional master key that can be used to identify an individual server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

libmemcached Function	Equivalent Core Function
key_length, value, value_length, expiration, flags)	
memcached_prepend_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_prepend(), but has the option of an additional master key that can be used to identify an individual server.
memcached_append_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_append(), but has the option of an additional master key that can be used to identify an individual server.
memcached_cas_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_cas(), but has the option of an additional master key that can be used to identify an individual server.

The by_key methods add two further arguments that define the master key, to be used and applied during the hashing stage for selecting the servers. You can see this in the following definition:

```
memcached_return
   memcached_set_by_key(memcached_st *ptr,
                        const char *master_key,
                        size_t master_key_length,
                        const char *key,
                        size_t key_length,
                        const char *value,
                        size_t value_length,
                        time_t expiration,
                        uint32_t flags);
```

All the functions return a value of type memcached_return, which you can compare against the MEMCACHED_SUCCESS constant.

libmemcached Get Functions

The libmemcached functions provide both direct access to a single item, and a multiple-key request mechanism that provides much faster responses when fetching a large number of keys simultaneously.

The main get-style function, which is equivalent to the generic get() is memcached_get(). This function returns a string pointer, pointing to the value associated with the specified key.

```
char *memcached_get (memcached_st *ptr,
                     const char *key, size_t key_length,
                     size_t *value_length,
                     uint32_t *flags,
                     memcached_return *error);
```

A multi-key get, memcached_mget(), is also available. Using a multiple key get operation is much quicker to do in one block than retrieving the key values with individual calls to memcached_get(). To start the multi-key get, call memcached_mget():

```
memcached_return
    memcached_mget (memcached_st *ptr,
                    char **keys, size_t *key_length,
                    unsigned int number_of_keys);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The return value is the success of the operation. The `keys` parameter should be an array of strings containing the keys, and `key_length` an array containing the length of each corresponding key. `number_of_keys` is the number of keys supplied in the array.

To fetch the individual values, use `memcached_fetch()` to get each corresponding value.

```
char *memcached_fetch (memcached_st *ptr,
                       const char *key, size_t *key_length,
                       size_t *value_length,
                       uint32_t *flags,
                       memcached_return *error);
```

The function returns the key value, with the `key`, `key_length` and `value_length` parameters being populated with the corresponding key and length information. The function returns `NULL` when there are no more values to be returned. A full example, including the populating of the key data and the return of the information is provided here.

```
#include <stdio.h>
#include <sstring.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *keys[]= {"huey", "dewey", "louie"};
  size_t key_length[3];
  char *values[]= {"red", "blue", "green"};
  size_t value_length[3];
  unsigned int x;
  uint32_t flags;

  char return_key[MEMCACHED_MAX_KEY];
  size_t return_key_length;
  char *return_value;
  size_t return_value_length;

  memc= memcached_create(NULL);

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));

  for(x= 0; x < 3; x++)
    {
      key_length[x] = strlen(keys[x]);
      value_length[x] = strlen(values[x]);

      rc= memcached_set(memc, keys[x], key_length[x], values[x],
                        value_length[x], (time_t)0, (uint32_t)0);
      if (rc == MEMCACHED_SUCCESS)
        fprintf(stderr,"Key %s stored successfully\n",keys[x]);
      else
        fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));
    }

  rc= memcached_mget(memc, keys, key_length, 3);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  if (rc == MEMCACHED_SUCCESS)
    {
      while ((return_value= memcached_fetch(memc, return_key, &return_key_length,
                                            &return_value_length, &flags, &rc)) != NULL)
        {
          if (rc == MEMCACHED_SUCCESS)
            {
              fprintf(stderr,"Key %s returned %s\n",return_key, return_value);
            }
        }
    }

  return 0;
}
```

Running the above application produces the following output:

```
shell> memc_multi_fetch
Added server successfully
Key huey stored successfully
Key dewey stored successfully
Key louie stored successfully
Key huey returned red
Key dewey returned blue
Key louie returned green
```

Controlling libmemcached Behaviors

The behavior of libmemcached can be modified by setting one or more behavior flags. These can either be set globally, or they can be applied during the call to individual functions. Some behaviors also accept an additional setting, such as the hashing mechanism used when selecting servers.

To set global behaviors:

```
memcached_return
   memcached_behavior_set (memcached_st *ptr,
                           memcached_behavior flag,
                           uint64_t data);
```

To get the current behavior setting:

```
uint64_t
   memcached_behavior_get (memcached_st *ptr,
                           memcached_behavior flag);
```

The following table describes libmemcached behavior flags.

Behavior	Description
MEMCACHED_BEHAVIOR_NO_BLOCK	Caused libmemcached to use asynchronous I/O.
MEMCACHED_BEHAVIOR_TCP_NODELAY	Turns on no-delay for network sockets.
MEMCACHED_BEHAVIOR_HASH	Without a value, sets the default hashing algorithm for keys to use MD5. Other valid values include MEMCACHED_HASH_DEFAULT, MEMCACHED_HASH_MD5, MEMCACHED_HASH_CRC, MEMCACHED_HASH_FNV1_64, MEMCACHED_HASH_FNV1A_64, MEMCACHED_HASH_FNV1_32, and MEMCACHED_HASH_FNV1A_32.
MEMCACHED_BEHAVIOR_DISTRIBUTION	Changes the method of selecting the server used to store a given value. The default method

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Behavior	Description
	is MEMCACHED_DISTRIBUTION_MODULA. You can enable consistent hashing by setting MEMCACHED_DISTRIBUTION_CONSISTENT. MEMCACHED_DISTRIBUTION_CONSISTENT is an alias for the value MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA.
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS	Cache the lookups made to the DNS service. This can improve the performance if you are using names instead of IP addresses for individual hosts.
MEMCACHED_BEHAVIOR_SUPPORT_CAS	Support CAS operations. By default, this is disabled because it imposes a performance penalty.
MEMCACHED_BEHAVIOR_KETAMA	Sets the default distribution to MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA and the hash to MEMCACHED_HASH_MD5.
MEMCACHED_BEHAVIOR_POLL_TIMEOUT	Modify the timeout value used by poll(). Supply a signed int pointer for the timeout value.
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS	Buffers IO requests instead of them being sent. A get operation, or closing the connection causes the data to be flushed.
MEMCACHED_BEHAVIOR_VERIFY_KEY	Forces libmemcached to verify that a specified key is valid.
MEMCACHED_BEHAVIOR_SORT_HOSTS	If set, hosts added to the list of configured hosts for a memcached_st structure are placed into the host list in sorted order. This breaks consistent hashing if that behavior has been enabled.
MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT	In nonblocking mode this changes the value of the timeout during socket connection.

libmemcached Command-Line Utilities

In addition to the main C library interface, libmemcached also includes a number of command-line utilities that can be useful when working with and debugging memcached applications.

All of the command-line tools accept a number of arguments, the most critical of which is servers, which specifies the list of servers to connect to when returning information.

The main tools are:

- memcat: Display the value for each ID given on the command line:

```
shell> memcat --servers=localhost hwkey
Hello world
```

- memcp: Copy the contents of a file into the cache, using the file name as the key:

```
shell> echo "Hello World" > hwkey
shell> memcp --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
Hello world
```

- memrm: Remove an item from the cache:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> memcat --servers=localhost hwkey
Hello world
shell> memrm --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
```

- memslap: Test the load on one or more memcached servers, simulating get/set and multiple client operations. For example, you can simulate the load of 100 clients performing get operations:

```
shell> memslap --servers=localhost --concurrency=100 --flush --test=get
memslap --servers=localhost --concurrency=100 --flush --test=get Threads connecting to servers 100
 Took 13.571 seconds to read data
```

- memflush: Flush (empty) the contents of the memcached cache.

```
shell> memflush --servers=localhost
```

15.3.3.4 Using MySQL and memcached with Perl

The Cache::Memcached module provides a native interface to the Memcache protocol, and provides support for the core functions offered by memcached. Install the module using your operating system's package management system, or using CPAN:

```
root-shell> perl -MCPAN -e 'install Cache::Memcached'
```

To use memcached from Perl through the Cache::Memcached module, first create a new Cache::Memcached object that defines the list of servers and other parameters for the connection. The only argument is a hash containing the options for the cache interface. For example, to create a new instance that uses three memcached servers:

```
use Cache::Memcached;

my $cache = new Cache::Memcached {
    'servers' => [
        '192.168.0.100:11211',
        '192.168.0.101:11211',
        '192.168.0.102:11211',
 ],
};
```

Note

When using the Cache::Memcached interface with multiple servers, the API automatically performs certain operations across all the servers in the group. For example, getting statistical information through Cache::Memcached returns a hash that contains data on a host-by-host basis, as well as generalized statistics for all the servers in the group.

You can set additional properties on the cache object instance when it is created by specifying the option as part of the option hash. Alternatively, you can use a corresponding method on the instance:

- servers or method set_servers(): Specifies the list of the servers to be used. The servers list should be a reference to an array of servers, with each element as the address and port number combination (separated by a colon). You can also specify a local connection through a Unix socket (for example /tmp/sock/memcached). To specify the server with a weight (indicating how much more frequently the server should be used during hashing), specify an array reference with the memcached server instance and a weight number. Higher numbers give higher priority.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- compress_threshold or method set_compress_threshold(): Specifies the threshold when values are compressed. Values larger than the specified number are automatically compressed (using zlib) during storage and retrieval.
- no_rehash or method set_norehash(): Disables finding a new server if the original choice is unavailable.
- readonly or method set_readonly(): Disables writes to the memcached servers.

Once the Cache::Memcached object instance has been configured, you can use the set() and get() methods to store and retrieve information from the memcached servers. Objects stored in the cache are automatically serialized and deserialized using the Storable module.

The Cache::Memcached interface supports the following methods for storing/retrieving data, and relate to the generic methods as shown in the table.

Cache::Memcached Function	Equivalent Generic Method
get()	Generic get().
get_multi(keys)	Gets multiple keys from memcache using just one query. Returns a hash reference of key/value pairs.
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

Below is a complete example for using memcached with Perl and the Cache::Memcached module:

```
#!/usr/bin/perl

use Cache::Memcached;
use DBI;
use Data::Dumper;

# Configure the memcached server

my $cache = new Cache::Memcached {
    'servers' => [
                  'localhost:11211',
                  ],
    };

# Get the film name from the command line
# memcached keys must not contain spaces, so create
# a key name by replacing spaces with underscores

my $filmname = shift or die "Must specify the film name\n";
my $filmkey = $filmname;
$filmkey =~ s/ /_/;

# Load the data from the cache

my $filmdata = $cache->get($filmkey);

# If the data wasn't in the cache, then we load it from the database
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
if (!defined($filmdata))
{
    $filmdata = load_filmdata($filmname);

    if (defined($filmdata))
    {

# Set the data into the cache, using the key

 if ($cache->set($filmkey,$filmdata))
        {
            print STDERR "Film data loaded from database and cached\n";
        }
        else
        {
            print STDERR "Couldn't store to cache\n";
 }
    }
    else
    {
      die "Couldn't find $filmname\n";
    }
}
else
{
    print STDERR "Film data loaded from Memcached\n";
}

sub load_filmdata
{
    my ($filmname) = @_;

    my $dsn = "DBI:mysql:database=sakila;host=localhost;port=3306";

    $dbh = DBI->connect($dsn, 'sakila','password');

    my ($filmbase) = $dbh->selectrow_hashref(sprintf('select * from film where title = %s',
                                                     $dbh->quote($filmname)));

    if (!defined($filmname))
    {
      return (undef);
    }

    $filmbase->{stars} =
 $dbh->selectall_arrayref(sprintf('select concat(first_name," ",last_name) ' .
                                          'from film_actor left join (actor) ' .
                                          'on (film_actor.actor_id = actor.actor_id) ' .
                                          ' where film_id=%s',
                                          $dbh->quote($filmbase->{film_id})));

    return($filmbase);
}
```

The example uses the Sakila database, obtaining film data from the database and writing a composite record of the film and actors to memcached. When calling it for a film does not exist, you get this result:

```
shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from database and cached
```

When accessing a film that has already been added to the cache:

```
shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from Memcached
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

15.3.3.5 Using MySQL and memcached with Python

The Python memcache module interfaces to memcached servers, and is written in pure Python (that is, without using one of the C APIs). You can download and install a copy from Python Memcached.

To install, download the package and then run the Python installer:

```
python setup.py install
running install
running bdist_egg
running egg_info
creating python_memcached.egg-info
...
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing python_memcached-1.43-py2.4.egg
creating /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Extracting python_memcached-1.43-py2.4.egg to /usr/lib64/python2.4/site-packages
Adding python-memcached 1.43 to easy-install.pth file

Installed /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Processing dependencies for python-memcached==1.43
Finished processing dependencies for python-memcached==1.43
```

Once installed, the memcache module provides a class-based interface to your memcached servers. When you store Python data structures as memcached items, they are automatically serialized (turned into string values) using the Python cPickle or pickle modules.

To create a new memcache interface, import the memcache module and create a new instance of the memcache.Client class. For example, if the memcached daemon is running on localhost using the default port:

```
import memcache
memc = memcache.Client(['127.0.0.1:11211'])
```

The first argument is an array of strings containing the server and port number for each memcached instance to use. To enable debugging, set the optional debug parameter to 1.

By default, the hashing mechanism used to divide the items among multiple servers is crc32. To change the function used, set the value of memcache.serverHashFunction to the alternate function to use. For example:

```
from zlib import adler32
memcache.serverHashFunction = adler32
```

Once you have defined the servers to use within the memcache instance, the core functions provide the same functionality as in the generic interface specification. The following table provides a summary of the supported functions:

Python memcache Function	Equivalent Generic Function
get()	Generic get().
get_multi(keys)	Gets multiple values from the supplied array of keys. Returns a hash reference of key/value pairs.
set()	Generic set().
set_multi(dict [, expiry [, key_prefix]])	Sets multiple key/value pairs from the supplied dict.
add()	Generic add().

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Python memcache Function	Equivalent Generic Function
replace()	Generic replace().
prepend(key, value [, expiry])	Prepends the supplied value to the value of the existing key.
append(key, value [, expiry[)	Appends the supplied value to the value of the existing key.
delete()	Generic delete().
delete_multi(keys [, expiry [, key_prefix]])	Deletes all the keys from the hash matching each string in the array keys.
incr()	Generic incr().
decr()	Generic decr().

Note

Within the Python memcache module, all the *_multi()functions support an optional key_prefix parameter. If supplied, then the string is used as a prefix to all key lookups. For example, if you call:

```
memc.get_multi(['a','b'], key_prefix='users:')
```

The function retrieves the keys users:a and users:b from the servers.

Here is an example showing the storage and retrieval of information to a memcache instance, loading the raw data from MySQL:

```
import sys
import MySQLdb
import memcache

memc = memcache.Client(['127.0.0.1:11211'], debug=1);

try:
    conn = MySQLdb.connect (host = "localhost",
                            user = "sakila",
                            passwd = "password",
                            db = "sakila")
except MySQLdb.Error, e:
     print "Error %d: %s" % (e.args[0], e.args[1])
     sys.exit (1)

popularfilms = memc.get('top5films')

if not popularfilms:
    cursor = conn.cursor()
    cursor.execute('select film_id,title from film order by rental_rate desc limit 5')
    rows = cursor.fetchall()
    memc.set('top5films',rows,60)
    print "Updated memcached with MySQL data"
else:
    print "Loaded data from memcached"
    for row in popularfilms:
        print "%s, %s" % (row[0], row[1])
```

When executed for the first time, the data is loaded from the MySQL database and stored to the memcached server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> python memc_python.py
Updated memcached with MySQL data
```

Because the data is automatically serialized using cPickle/pickle, when you load the data back from memcached, you can use the object directly. In the example above, the information stored to memcached is in the form of rows from a Python DB cursor. When accessing the information (within the 60 second expiry time), the data is loaded from memcached and dumped:

```
shell> python memc_python.py
Loaded data from memcached
2, ACE GOLDFINGER
7, AIRPLANE SIERRA
8, AIRPORT POLLOCK
10, ALADDIN CALENDAR
13, ALI FOREVER
```

The serialization and deserialization happens automatically. Because serialization of Python data may be incompatible with other interfaces and languages, you can change the serialization module used during initialization. For example, you might use JSON format when you store complex data structures using a script written in one language, and access them in a script written in a different language.

15.3.3.6 Using MySQL and memcached with PHP

PHP provides support for the Memcache functions through a PECL extension. To enable the PHP memcache extensions, build PHP using the --enable-memcache option to configure when building from source.

If you are installing on a Red Hat-based server, you can install the php-pecl-memcache RPM:

```
root-shell> yum --install php-pecl-memcache
```

On Debian-based distributions, use the php-memcache package.

To set global runtime configuration options, specify the configuration option values within your php.ini file. The following table provides the name, default value, and a description for each global runtime configuration option.

Configuration option	Default	Description
memcache.allow_failover	1	Specifies whether another server in the list should be queried if the first server selected fails.
memcache.max_failover_attempts	20	Specifies the number of servers to try before returning a failure.
memcache.chunk_size	8192	Defines the size of network chunks used to exchange data with the memcached server.
memcache.default_port	11211	Defines the default port to use when communicating with the memcached servers.
memcache.hash_strategy	standard	Specifies which hash strategy to use. Set to consistent to enable servers to be added or removed from the pool without causing the keys to be remapped to other servers. When set to standard, an older (modula) strategy is used that potentially uses different servers for storage.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Configuration option	Default	Description
memcache.hash_function	crc32	Specifies which function to use when mapping keys to servers. crc32 uses the standard CRC32 hash. fnv uses the FNV-1a hashing algorithm.

To create a connection to a memcached server, create a new Memcache object and then specify the connection options. For example:

```
<?php

$cache = new Memcache;
$cache->connect('localhost',11211);
?>
```

This opens an immediate connection to the specified server.

To use multiple memcached servers, you need to add servers to the memcache object using addServer():

```
bool Memcache::addServer ( string $host [, int $port [, bool $persistent
                [, int $weight [, int $timeout [, int $retry_interval
                [, bool $status [, callback $failure_callback
                ]]]]]] )
```

The server management mechanism within the php-memcache module is a critical part of the interface as it controls the main interface to the memcached instances and how the different instances are selected through the hashing mechanism.

To create a simple connection to two memcached instances:

```
<?php

$cache = new Memcache;
$cache->addServer('192.168.0.100',11211);
$cache->addServer('192.168.0.101',11211);
?>
```

In this scenario, the instance connection is not explicitly opened, but only opened when you try to store or retrieve a value. To enable persistent connections to memcached instances, set the $persistent argument to true. This is the default setting, and causes the connections to remain open.

To help control the distribution of keys to different instances, use the global memcache.hash_strategy setting. This sets the hashing mechanism used to select. You can also add another weight to each server, which effectively increases the number of times the instance entry appears in the instance list, therefore increasing the likelihood of the instance being chosen over other instances. To set the weight, set the value of the $weight argument to more than one.

The functions for setting and retrieving information are identical to the generic functional interface offered by memcached, as shown in this table:

PECL memcache Function	Generic Function
get()	Generic get().
set()	Generic set().
add()	Generic add().
replace()	Generic replace().

This documentation is for an older version. If you're
This documentation is for an older version. If you're

PECL memcache Function	Generic Function
delete()	Generic delete().
increment()	Generic incr().
decrement()	Generic decr().

A full example of the PECL memcache interface is provided below. The code loads film data from the Sakila database when the user provides a film name. The data stored into the memcached instance is recorded as a mysqli result row, and the API automatically serializes the information for you.

```
<?php

$memc = new Memcache;
$memc->addServer('localhost','11211');

if(empty($_POST['film'])) {
?>
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Simple Memcache Lookup</title>
    </head>
    <body>
      <form method="post">
        <p><b>Film</b>: <input type="text" size="20" name="film"></p>
        <input type="submit">
      </form>
      <hr/>
<?php

} else {

    echo "Loading data...\n";

    $film   = htmlspecialchars($_POST['film'], ENT_QUOTES, 'UTF-8');
    $mfilms = $memc->get($film);

    if ($mfilms) {

        printf("<p>Film data for %s loaded from memcache</p>", $mfilms['title']);

        foreach (array_keys($mfilms) as $key) {
            printf("<p><b>%s</b>: %s</p>", $key, $mfilms[$key]);
        }

    } else {

        $mysqli = mysqli('localhost','sakila','password','sakila');

        if (mysqli_connect_error()) {
            sprintf("Database error: (%d) %s", mysqli_connect_errno(), mysqli_connect_error());
            exit;
        }

        $sql = sprintf('SELECT * FROM film WHERE title="%s"', $mysqli->real_escape_string($film));

        $result = $mysqli->query($sql);

        if (!$result) {
            sprintf("Database error: (%d) %s", $mysqli->errno, $mysqli->error);
            exit;
        }
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        $row = $result->fetch_assoc();

        $memc->set($row['title'], $row);

        printf("<p>Loaded (%s) from MySQL</p>", htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8');
    }
}
?>
  </body>
</html>
```

With PHP, the connections to the memcached instances are kept open as long as the PHP and associated Apache instance remain running. When adding or removing servers from the list in a running instance (for example, when starting another script that mentions additional servers), the connections are shared, but the script only selects among the instances explicitly configured within the script.

To ensure that changes to the server list within a script do not cause problems, make sure to use the consistent hashing mechanism.

15.3.3.7 Using MySQL and memcached with Ruby

There are a number of different modules for interfacing to memcached within Ruby. The Ruby-MemCache client library provides a native interface to memcached that does not require any external libraries, such as libmemcached. You can obtain the installer package from http://www.deveiate.org/projects/RMemCache.

To install, extract the package and then run install.rb:

```
shell> install.rb
```

If you have RubyGems, you can install the Ruby-MemCache gem:

```
shell> gem install Ruby-MemCache
Bulk updating Gem source index for: http://gems.rubyforge.org
Install required dependency io-reactor? [Yn]  y
Successfully installed Ruby-MemCache-0.0.1
Successfully installed io-reactor-0.05
Installing ri documentation for io-reactor-0.05...
Installing RDoc documentation for io-reactor-0.05...
```

To use a memcached instance from within Ruby, create a new instance of the MemCache object.

```
require 'memcache'
memc = MemCache::new '192.168.0.100:11211'
```

You can add a weight to each server to increase the likelihood of the server being selected during hashing by appending the weight count to the server host name/port string:

```
require 'memcache'
memc = MemCache::new '192.168.0.100:11211:3'
```

To add servers to an existing list, you can append them directly to the MemCache object:

```
memc += ["192.168.0.101:11211"]
```

To set data into the cache, you can just assign a value to a key within the new cache object, which works just like a standard Ruby hash object:

```
memc["key"] = "value"
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Or to retrieve the value:

```
print memc["key"]
```

For more explicit actions, you can use the method interface, which mimics the main memcached API functions, as summarized in the following table:

Ruby MemCache Method	Equivalent memcached API Functions
get()	Generic get().
get_hash(keys)	Get the values of multiple keys, returning the information as a hash of the keys and their values.
set()	Generic set().
set_many(pairs)	Set the values of the keys and values in the hash pairs.
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

15.3.3.8 Using MySQL and memcached with Java

The com.danga.MemCached class within Java provides a native interface to memcached instances. You can obtain the client from https://github.com/gwhalin/Memcached-Java-Client/downloads. The Java class uses hashes that are compatible with libmemcached, so you can mix and match Java and libmemcached applications accessing the same memcached instances. The serialization between Java and other interfaces are not compatible. If this is a problem, use JSON or a similar nonbinary serialization format.

On most systems, you can download the package and use the jar directly.

To use the com.danga.MemCached interface, you create a MemCachedClient instance and then configure the list of servers by configuring the SockIOPool. Through the pool specification you set up the server list, weighting, and the connection parameters to optimized the connections between your client and the memcached instances that you configure.

Generally, you can configure the memcached interface once within a single class, then use this interface throughout the rest of your application.

For example, to create a basic interface, first configure the MemCachedClient and base SockIOPool settings:

```
public class MyClass {

    protected static MemCachedClient mcc = new MemCachedClient();

    static {

        String[] servers =
            {
                "localhost:11211",
            };

        Integer[] weights = { 1 };
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
        SockIOPool pool = SockIOPool.getInstance();

        pool.setServers( servers );
        pool.setWeights( weights );
```

In the above sample, the list of servers is configured by creating an array of the memcached instances to use. You can then configure individual weights for each server.

The remainder of the properties for the connection are optional, but you can set the connection numbers (initial connections, minimum connections, maximum connections, and the idle timeout) by setting the pool parameters:

```
pool.setInitConn( 5 );
pool.setMinConn( 5 );
pool.setMaxConn( 250 );
pool.setMaxIdle( 1000 * 60 * 60 * 6
```

Once the parameters have been configured, initialize the connection pool:

```
pool.initialize();
```

The pool, and the connection to your memcached instances should now be ready to use.

To set the hashing algorithm used to select the server used when storing a given key, use pool.setHashingAlg():

```
pool.setHashingAlg( SockIOPool.NEW_COMPAT_HASH );
```

Valid values are NEW_COMPAT_HASH, OLD_COMPAT_HASH and NATIVE_HASH are also basic modula hashing algorithms. For a consistent hashing algorithm, use CONSISTENT_HASH. These constants are equivalent to the corresponding hash settings within libmemcached.

The following table outlines the Java com.danga.MemCached methods and the equivalent generic methods in the memcached interface specification.

Java com.danga.MemCached Method	Equivalent Generic Method
get()	Generic get().
getMulti(keys)	Get the values of multiple keys, returning the information as Hash map using java.lang.String for the keys and java.lang.Object for the corresponding values.
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

15.3.3.9 Using the memcached TCP Text Protocol

Communicating with a memcached server can be achieved through either the TCP or UDP protocols. When using the TCP protocol, you can use a simple text based interface for the exchange of information.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When communicating with memcached, you can connect to the server using the port configured for the server. You can open a connection with the server without requiring authorization or login. As soon as you have connected, you can start to send commands to the server. When you have finished, you can terminate the connection without sending any specific disconnection command. Clients are encouraged to keep their connections open to decrease latency and improve performance.

Data is sent to the memcached server in two forms:

- Text lines, which are used to send commands to the server, and receive responses from the server.
- Unstructured data, which is used to receive or send the value information for a given key. Data is returned to the client in exactly the format it was provided.

Both text lines (commands and responses) and unstructured data are always terminated with the string \r \n. Because the data being stored may contain this sequence, the length of the data (returned by the client before the unstructured data is transmitted should be used to determine the end of the data.

Commands to the server are structured according to their operation:

- **Storage commands**: set, add, replace, append, prepend, cas

  Storage commands to the server take the form:

  ```
  command key [flags] [exptime] length [noreply]
  ```

  Or when using compare and swap (cas):

  ```
  cas key [flags] [exptime] length [casunique] [noreply]
  ```

  Where:

  - command: The command name.
    - set: Store value against key
    - add: Store this value against key if the key does not already exist
    - replace: Store this value against key if the key already exists
    - append: Append the supplied value to the end of the value for the specified key. The flags and exptime arguments should not be used.
    - prepend: Append value currently in the cache to the end of the supplied value for the specified key. The flags and exptime arguments should not be used.
    - cas: Set the specified key to the supplied value, only if the supplied casunique matches. This is effectively the equivalent of change the information if nobody has updated it since I last fetched it.
  - key: The key. All data is stored using a the specific key. The key cannot contain control characters or whitespace, and can be up to 250 characters in size.
  - flags: The flags for the operation (as an integer). Flags in memcached are transparent. The memcached server ignores the contents of the flags. They can be used by the client to indicate any type of information. In memcached 1.2.0 and lower the value is a 16-bit integer value. In memcached 1.2.1 and higher the value is a 32-bit integer.
  - exptime: The expiry time, or zero for no expiry.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- length: The length of the supplied value block in bytes, excluding the terminating \r\n characters.
- casunique: A unique 64-bit value of an existing entry. This is used to compare against the existing value. Use the value returned by the gets command when issuing cas updates.
- noreply: Tells the server not to reply to the command.

For example, to store the value abcdef into the key xyzkey, you would use:

```
set xyzkey 0 0 6\r\nabcdef\r\n
```

The return value from the server is one line, specifying the status or error information. For more information, see Table 15.3, "memcached Protocol Responses".

- **Retrieval commands**: get, gets

  Retrieval commands take the form:

  ```
  get key1 [key2 .... keyn]
  gets key1 [key2 ... keyn]
  ```

  You can supply multiple keys to the commands, with each requested key separated by whitespace.

  The server responds with an information line of the form:

  ```
  VALUE key flags bytes [casunique]
  ```

  Where:

  - key: The key name.
  - flags: The value of the flag integer supplied to the memcached server when the value was stored.
  - bytes: The size (excluding the terminating \r\n character sequence) of the stored value.
  - casunique: The unique 64-bit integer that identifies the item.

  The information line is immediately followed by the value data block. For example:

  ```
  get xyzkey\r\n
  VALUE xyzkey 0 6\r\n
  abcdef\r\n
  ```

  If you have requested multiple keys, an information line and data block is returned for each key found. If a requested key does not exist in the cache, no information is returned.

- **Delete commands**: delete

  Deletion commands take the form:

  ```
  delete key [time] [noreply]
  ```

  Where:

  - key: The key name.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- time: The time in seconds (or a specific Unix time) for which the client wishes the server to refuse add or replace commands on this key. All add, replace, get, and gets commands fail during this period. set operations succeed. After this period, the key is deleted permanently and all commands are accepted.

  If not supplied, the value is assumed to be zero (delete immediately).

- noreply: Tells the server not to reply to the command.

Responses to the command are either DELETED to indicate that the key was successfully removed, or NOT_FOUND to indicate that the specified key could not be found.

- **Increment/Decrement**: incr, decr

  The increment and decrement commands change the value of a key within the server without performing a separate get/set sequence. The operations assume that the currently stored value is a 64-bit integer. If the stored value is not a 64-bit integer, then the value is assumed to be zero before the increment or decrement operation is applied.

  Increment and decrement commands take the form:

```
incr key value [noreply]
decr key value [noreply]
```

  Where:

  - key: The key name.
  - value: An integer to be used as the increment or decrement value.
  - noreply: Tells the server not to reply to the command.

  The response is:

  - NOT_FOUND: The specified key could not be located.
  - value: The new value associated with the specified key.

  Values are assumed to be unsigned. For decr operations, the value is never decremented below 0. For incr operations, the value wraps around the 64-bit maximum.

- **Statistics commands**: stats

  The stats command provides detailed statistical information about the current status of the memcached instance and the data it is storing.

  Statistics commands take the form:

```
STAT [name] [value]
```

  Where:

  - name: The optional name of the statistics to return. If not specified, the general statistics are returned.
  - value: A specific value to be used when performing certain statistics operations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The return value is a list of statistics data, formatted as follows:

```
STAT name value
```

The statistics are terminated with a single line, END.

For more information, see Section 15.3.4, “Getting memcached Statistics”.

For reference, a list of the different commands supported and their formats is provided below.

Table 15.2 memcached Command Reference

Command	Command Formats
set	set key flags exptime length, set key flags exptime length noreply
add	add key flags exptime length, add key flags exptime length noreply
replace	replace key flags exptime length, replace key flags exptime length noreply
append	append key length, append key length noreply
prepend	prepend key length, prepend key length noreply
cas	cas key flags exptime length casunique, cas key flags exptime length casunique noreply
get	get key1 [key2 ... keyn]
gets	
delete	delete key, delete key noreply, delete key expiry, delete key expiry noreply
incr	incr key, incr key noreply, incr key value, incr key value noreply
decr	decr key, decr key noreply, decr key value, decr key value noreply
stat	stat, stat name, stat name value

When sending a command to the server, the response from the server is one of the settings in the following table. All response values from the server are terminated by \r\n:

Table 15.3 memcached Protocol Responses

String	Description
STORED	Value has successfully been stored.
NOT_STORED	The value was not stored, but not because of an error. For commands where you are adding a or updating a value if it exists (such as add and replace), or where the item has already been set to be deleted.
EXISTS	When using a cas command, the item you are trying to store already exists and has been modified since you last checked it.
NOT_FOUND	The item you are trying to store, update or delete does not exist or has already been deleted.
ERROR	You submitted a nonexistent command name.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

String	Description
CLIENT_ERROR errorstring	There was an error in the input line, the detail is contained in errorstring.
SERVER_ERROR errorstring	There was an error in the server that prevents it from returning the information. In extreme conditions, the server may disconnect the client after this error occurs.
VALUE keys flags length	The requested key has been found, and the stored key, flags and data block are returned, of the specified length.
DELETED	The requested key was deleted from the server.
STAT name value	A line of statistics data.
END	The end of the statistics data.

15.3.4 Getting memcached Statistics

The memcached system has a built-in statistics system that collects information about the data being stored into the cache, cache hit ratios, and detailed information on the memory usage and distribution of information through the slab allocation used to store individual items. Statistics are provided at both a basic level that provide the core statistics, and more specific statistics for specific areas of the memcached server.

This information can be useful to ensure that you are getting the correct level of cache and memory usage, and that your slab allocation and configuration properties are set at an optimal level.

The stats interface is available through the standard memcached protocol, so the reports can be accessed by using telnet to connect to the memcached. The supplied memcached-tool includes support for obtaining the Section 15.3.4.2, “memcached Slabs Statistics” and Section 15.3.4.1, “memcached General Statistics” information. For more information, see Section 15.3.4.6, “Using memcached-tool”.

Alternatively, most of the language API interfaces provide a function for obtaining the statistics from the server.

For example, to get the basic stats using telnet:

```
shell> telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats
STAT pid 23599
STAT uptime 675
STAT time 1211439587
STAT version 1.2.5
STAT pointer_size 32
STAT rusage_user 1.404992
STAT rusage_system 4.694685
STAT curr_items 32
STAT total_items 56361
STAT bytes 2642
STAT curr_connections 53
STAT total_connections 438
STAT connection_structures 55
STAT cmd_get 113482
STAT cmd_set 80519
STAT get_hits 78926
STAT get_misses 34556
STAT evictions 0
STAT bytes_read 6379783
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
STAT bytes_written 4860179
STAT limit_maxbytes 67108864
STAT threads 1
END
```

When using Perl and the Cache::Memcached module, the stats() function returns information about all the servers currently configured in the connection object, and total statistics for all the memcached servers as a whole.

For example, the following Perl script obtains the stats and dumps the hash reference that is returned:

```
use Cache::Memcached;
use Data::Dumper;

my $memc = new Cache::Memcached;
$memc->set_servers(\@ARGV);

print Dumper($memc->stats());
```

When executed on the same memcached as used in the Telnet example above we get a hash reference with the host by host and total statistics:

```
$VAR1 = {
    'hosts' => {
           'localhost:11211' => {
                      'misc' => {
                            'bytes' => '2421',
                            'curr_connections' => '3',
                            'connection_structures' => '56',
                            'pointer_size' => '32',
                            'time' => '1211440166',
                            'total_items' => '410956',
                            'cmd_set' => '588167',
                            'bytes_written' => '35715151',
                            'evictions' => '0',
                            'curr_items' => '31',
                            'pid' => '23599',
                            'limit_maxbytes' => '67108864',
                            'uptime' => '1254',
                            'rusage_user' => '9.857805',
                            'cmd_get' => '838451',
                            'rusage_system' => '34.096988',
                            'version' => '1.2.5',
                            'get_hits' => '581511',
                            'bytes_read' => '46665716',
                            'threads' => '1',
                            'total_connections' => '3104',
                            'get_misses' => '256940'
                          },
                      'sizes' => {
                             '128' => '16',
                             '64' => '15'
                           }
                    }
         },
    'self' => {},
    'total' => {
           'cmd_get' => 838451,
           'bytes' => 2421,
           'get_hits' => 581511,
           'connection_structures' => 56,
           'bytes_read' => 46665716,
           'total_items' => 410956,
           'total_connections' => 3104,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
          'cmd_set' => 588167,
          'bytes_written' => 35715151,
          'curr_items' => 31,
          'get_misses' => 256940
        }
      };
```

The statistics are divided up into a number of distinct sections, and then can be requested by adding the type to the stats command. Each statistics output is covered in more detail in the following sections.

- General statistics, see Section 15.3.4.1, “memcached General Statistics”.
- Slab statistics (slabs), see Section 15.3.4.2, “memcached Slabs Statistics”.
- Item statistics (items), see Section 15.3.4.3, “memcached Item Statistics”.
- Size statistics (sizes), see Section 15.3.4.4, “memcached Size Statistics”.
- Detailed status (detail), see Section 15.3.4.5, “memcached Detail Statistics”.

15.3.4.1 memcached General Statistics

The output of the general statistics provides an overview of the performance and use of the memcached instance. The statistics returned by the command and their meaning is shown in the following table.

The following terms are used to define the value type for each statistics value:

- 32u: 32-bit unsigned integer
- 64u: 64-bit unsigned integer
- 32u:32u: Two 32-bit unsigned integers separated by a colon
- String: Character string

Statistic	Data type	Description	Version
pid	32u	Process ID of the memcached instance.	
uptime	32u	Uptime (in seconds) for this memcached instance.	
time	32u	Current time (as epoch).	
version	string	Version string of this instance.	
pointer_size	string	Size of pointers for this host specified in bits (32 or 64).	
rusage_user	32u:32u	Total user time for this instance (seconds:microseconds).	
rusage_system	32u:32u	Total system time for this instance (seconds:microseconds).	
curr_items	32u	Current number of items stored by this instance.	
total_items	32u	Total number of items stored during the life of this instance.	
bytes	64u	Current number of bytes used by this server to store items.	
curr_connections	32u	Current number of open connections.	
total_connections	32u	Total number of connections opened since the server started running.	
connection_structures	32u	Number of connection structures allocated by the server.	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Statistic	Data type	Description	Version
cmd_get	64u	Total number of retrieval requests (get operations).	
cmd_set	64u	Total number of storage requests (set operations).	
get_hits	64u	Number of keys that have been requested and found present.	
get_misses	64u	Number of items that have been requested and not found.	
delete_hits	64u	Number of keys that have been deleted and found present.	1.3.x
delete_misses	64u	Number of items that have been delete and not found.	1.3.x
incr_hits	64u	Number of keys that have been incremented and found present.	1.3.x
incr_misses	64u	Number of items that have been incremented and not found.	1.3.x
decr_hits	64u	Number of keys that have been decremented and found present.	1.3.x
decr_misses	64u	Number of items that have been decremented and not found.	1.3.x
cas_hits	64u	Number of keys that have been compared and swapped and found present.	1.3.x
cas_misses	64u	Number of items that have been compared and swapped and not found.	1.3.x
cas_badvalue	64u	Number of keys that have been compared and swapped, but the comparison (original) value did not match the supplied value.	1.3.x
evictions	64u	Number of valid items removed from cache to free memory for new items.	
bytes_read	64u	Total number of bytes read by this server from network.	
bytes_written	64u	Total number of bytes sent by this server to network.	
limit_maxbytes	32u	Number of bytes this server is permitted to use for storage.	
threads	32u	Number of worker threads requested.	
conn_yields	64u	Number of yields for connections (related to the -R option).	1.4.0

The most useful statistics from those given here are the number of cache hits, misses, and evictions.

A large number of get_misses may just be an indication that the cache is still being populated with information. The number should, over time, decrease in comparison to the number of cache get_hits. If, however, you have a large number of cache misses compared to cache hits after an extended period of execution, it may be an indication that the size of the cache is too small and you either need to increase the total memory size, or increase the number of the memcached instances to improve the hit ratio.

A large number of evictions from the cache, particularly in comparison to the number of items stored is a sign that your cache is too small to hold the amount of information that you regularly want to keep cached. Instead of items being retained in the cache, items are being evicted to make way for new items keeping the turnover of items in the cache high, reducing the efficiency of the cache.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

15.3.4.2 memcached Slabs Statistics

To get the slabs statistics, use the stats slabs command, or the API equivalent.

The slab statistics provide you with information about the slabs that have created and allocated for storing information within the cache. You get information both on each individual slab-class and total statistics for the whole slab.

```
STAT 1:chunk_size 104
STAT 1:chunks_per_page 10082
STAT 1:total_pages 1
STAT 1:total_chunks 10082
STAT 1:used_chunks 10081
STAT 1:free_chunks 1
STAT 1:free_chunks_end 10079
STAT 9:chunk_size 696
STAT 9:chunks_per_page 1506
STAT 9:total_pages 63
STAT 9:total_chunks 94878
STAT 9:used_chunks 94878
STAT 9:free_chunks 0
STAT 9:free_chunks_end 0
STAT active_slabs 2
STAT total_malloced 67083616
END
```

Individual stats for each slab class are prefixed with the slab ID. A unique ID is given to each allocated slab from the smallest size up to the largest. The prefix number indicates the slab class number in relation to the calculated chunk from the specified growth factor. Hence in the example, 1 is the first chunk size and 9 is the 9th chunk allocated size.

The parameters returned for each chunk size and a description of each parameter are provided in the following table.

Statistic	Description	Version
chunk_size	Space allocated to each chunk within this slab class.	
chunks_per_page	Number of chunks within a single page for this slab class.	
total_pages	Number of pages allocated to this slab class.	
total_chunks	Number of chunks allocated to the slab class.	
used_chunks	Number of chunks allocated to an item..	
free_chunks	Number of chunks not yet allocated to items.	
free_chunks_end	Number of free chunks at the end of the last allocated page.	
get_hits	Number of get hits to this chunk	1.3.x
cmd_set	Number of set commands on this chunk	1.3.x
delete_hits	Number of delete hits to this chunk	1.3.x
incr_hits	Number of increment hits to this chunk	1.3.x
decr_hits	Number of decrement hits to this chunk	1.3.x
cas_hits	Number of CAS hits to this chunk	1.3.x
cas_badval	Number of CAS hits on this chunk where the existing value did not match	1.3.x
mem_requested	The true amount of memory of memory requested within this chunk	1.4.1

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following additional statistics cover the information for the entire server, rather than on a chunk by chunk basis:

Statistic	Description	Version
active_slabs	Total number of slab classes allocated.	
total_malloced	Total amount of memory allocated to slab pages.	

The key values in the slab statistics are the chunk_size, and the corresponding total_chunks and used_chunks parameters. These given an indication of the size usage of the chunks within the system. Remember that one key/value pair is placed into a chunk of a suitable size.

From these stats, you can get an idea of your size and chunk allocation and distribution. If you store many items with a number of largely different sizes, consider adjusting the chunk size growth factor to increase in larger steps to prevent chunk and memory wastage. A good indication of a bad growth factor is a high number of different slab classes, but with relatively few chunks actually in use within each slab. Increasing the growth factor creates fewer slab classes and therefore makes better use of the allocated pages.

15.3.4.3 memcached Item Statistics

To get the items statistics, use the stats items command, or the API equivalent.

The items statistics give information about the individual items allocated within a given slab class.

```
STAT items:2:number 1
STAT items:2:age 452
STAT items:2:evicted 0
STAT items:2:evicted_nonzero 0
STAT items:2:evicted_time 2
STAT items:2:outofmemory 0
STAT items:2:tailrepairs 0
...
STAT items:27:number 1
STAT items:27:age 452
STAT items:27:evicted 0
STAT items:27:evicted_nonzero 0
STAT items:27:evicted_time 2
STAT items:27:outofmemory 0
STAT items:27:tailrepairs 0
```

The prefix number against each statistics relates to the corresponding chunk size, as returned by the stats slabs statistics. The result is a display of the number of items stored within each chunk within each slab size, and specific statistics about their age, eviction counts, and out of memory counts. A summary of the statistics is given in the following table.

Statistic	Description	
number	The number of items currently stored in this slab class.	
age	The age of the oldest item within the slab class, in seconds.	
evicted	The number of items evicted to make way for new entries.	
evicted_time	The time of the last evicted entry	
evicted_nonzero	The time of the last evicted non-zero entry	1.4.0
outofmemory	The number of items for this slab class that have triggered an out of memory error (only value when the -M command line option is in effect).	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Statistic	Description	
tailrepairs	Number of times the entries for a particular ID need repairing	

Item level statistics can be used to determine how many items are stored within a given slab and their freshness and recycle rate. You can use this to help identify whether there are certain slab classes that are triggering a much larger number of evictions that others.

15.3.4.4 memcached Size Statistics

To get size statistics, use the `stats sizes` command, or the API equivalent.

The size statistics provide information about the sizes and number of items of each size within the cache. The information is returned as two columns, the first column is the size of the item (rounded up to the nearest 32 byte boundary), and the second column is the count of the number of items of that size within the cache:

```
96 35
128 38
160 807
192 804
224 410
256 222
288 83
320 39
352 53
384 33
416 64
448 51
480 30
512 54
544 39
576 10065
```

Caution

Running this statistic locks up your cache as each item is read from the cache and its size calculated. On a large cache, this may take some time and prevent any set or get operations until the process completes.

The item size statistics are useful only to determine the sizes of the objects you are storing. Since the actual memory allocation is relevant only in terms of the chunk size and page size, the information is only useful during a careful debugging or diagnostic session.

15.3.4.5 memcached Detail Statistics

For memcached 1.3.x and higher, you can enable and obtain detailed statistics about the get, set, and del operations on theindividual keys stored in the cache, and determine whether the attempts hit (found) a particular key. These operations are only recorded while the detailed stats analysis is turned on.

To enable detailed statistics, you must send the `stats detail on` command to the memcached server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.
stats detail on
OK
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Individual statistics are recorded for every get, set and del operation on a key, including keys that are not currently stored in the server. For example, if an attempt is made to obtain the value of key abckey and it does not exist, the get operating on the specified key are recorded while detailed statistics are in effect, even if the key is not currently stored. The hits, that is, the number of get or del operations for a key that exists in the server are also counted.

To turn detailed statistics off, send the stats detail off command to the memcached server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.
stats detail on
OK
```

To obtain the detailed statistics recorded during the process, send the stats detail dump command to the memcached server:

```
stats detail dump
PREFIX hykkey get 0 hit 0 set 1 del 0
PREFIX xyzkey get 0 hit 0 set 1 del 0
PREFIX yukkey get 1 hit 0 set 0 del 0
PREFIX abckey get 3 hit 3 set 1 del 0
END
```

You can use the detailed statistics information to determine whether your memcached clients are using a large number of keys that do not exist in the server by comparing the hit and get or del counts. Because the information is recorded by key, you can also determine whether the failures or operations are clustered around specific keys.

15.3.4.6 Using memcached-tool

The memcached-tool, located within the scripts directory within the memcached source directory. The tool provides convenient access to some reports and statistics from any memcached instance.

The basic format of the command is:

```
shell> ./memcached-tool hostname:port [command]
```

The default output produces a list of the slab allocations and usage. For example:

```
shell> memcached-tool localhost:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1      80B        93s       1      20      no        0        0    0
  2     104B        93s       1      16      no        0        0    0
  3     136B      1335s       1      28      no        0        0    0
  4     176B      1335s       1      24      no        0        0    0
  5     224B      1335s       1      32      no        0        0    0
  6     280B      1335s       1      34      no        0        0    0
  7     352B      1335s       1      36      no        0        0    0
  8     440B      1335s       1      46      no        0        0    0
  9     552B      1335s       1      58      no        0        0    0
 10     696B      1335s       1      66      no        0        0    0
 11     872B      1335s       1      89      no        0        0    0
 12     1.1K      1335s       1     112      no        0        0    0
 13     1.3K      1335s       1     145      no        0        0    0
 14     1.7K      1335s       1     123      no        0        0    0
 15     2.1K      1335s       1     198      no        0        0    0
 16     2.6K      1335s       1     199      no        0        0    0
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
 17     3.3K      1335s       1     229      no        0        0    0
 18     4.1K      1335s       1     248     yes       36        2    0
 19     5.2K      1335s       2     328      no        0        0    0
 20     6.4K      1335s       2     316     yes      387        1    0
 21     8.1K      1335s       3     381     yes      492        1    0
 22    10.1K      1335s       3     303     yes      598        2    0
 23    12.6K      1335s       5     405     yes      605        1    0
 24    15.8K      1335s       6     384     yes      766        2    0
 25    19.7K      1335s       7     357     yes      908      170    0
 26    24.6K      1336s       7     287     yes     1012        1    0
 27    30.8K      1336s       7     231     yes     1193      169    0
 28    38.5K      1336s       4     104     yes     1323      169    0
 29    48.1K      1336s       1      21     yes     1287        1    0
 30    60.2K      1336s       1      17     yes     1093      169    0
 31    75.2K      1337s       1      13     yes      713      168    0
 32    94.0K      1337s       1      10     yes      278      168    0
 33   117.5K      1336s       1       3      no        0        0    0
```

This output is the same if you specify the `command` as `display`:

```
shell> memcached-tool localhost:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1      80B        93s       1      20      no        0        0    0
  2     104B        93s       1      16      no        0        0    0
...
```

The output shows a summarized version of the output from the `slabs` statistics. The columns provided in the output are shown below:

- `#`: The slab number
- `Item_Size`: The size of the slab
- `Max_age`: The age of the oldest item in the slab
- `Pages`: The number of pages allocated to the slab
- `Count`: The number of items in this slab
- `Full?`: Whether the slab is fully populated
- `Evicted`: The number of objects evicted from this slab
- `Evict_Time`: The time (in seconds) since the last eviction
- `OOM`: The number of items that have triggered an out of memory error

You can also obtain a dump of the general statistics for the server using the `stats` command:

```
shell> memcached-tool localhost:11211 stats
#localhost:11211   Field       Value
         accepting_conns           1
                   bytes         162
              bytes_read         485
           bytes_written        6820
              cas_badval           0
                cas_hits           0
              cas_misses           0
               cmd_flush           0
                 cmd_get           4
                 cmd_set           2
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
           conn_yields           0
connection_structures          11
     curr_connections          10
           curr_items           2
            decr_hits           0
          decr_misses           1
          delete_hits           0
        delete_misses           0
            evictions           0
             get_hits           4
           get_misses           0
            incr_hits           0
          incr_misses           2
       limit_maxbytes    67108864
 listen_disabled_num           0
                  pid       12981
        pointer_size          32
       rusage_system    0.013911
         rusage_user    0.011876
             threads           4
                time  1255518565
   total_connections          20
         total_items           2
              uptime         880
             version       1.4.2
```

15.3.5 memcached FAQ

15.3.5.1 Can memcached be run on a Windows environment?

No. Currently memcached is available only on the Unix/Linux platform. There is an unofficial port available, see http://www.codeplex.com/memcachedproviders.

15.3.5.2 What is the maximum size of an object you can store in memcached? Is that configurable?

The default maximum object size is 1MB. In memcached 1.4.2 and later, you can change the maximum size of an object using the -I command line option.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For versions before this, to increase this size, you have to re-compile memcached. You can modify the value of the POWER_BLOCK within the slabs.c file within the source.

In memcached 1.4.2 and higher, you can configure the maximum supported object size by using the -I command-line option. For example, to increase the maximum object size to 5MB:

```
$ memcached -I 5m
```

If an object is larger than the maximum object size, you must manually split it. memcached is very simple: you give it a key and some data, it tries to cache it in RAM. If you try to store more than the default maximum size, the value is just truncated for speed reasons.

15.3.5.3. Is it true memcached will be much more effective with db-read-intensive applications than with db-write-intensive applications?

Yes. memcached plays no role in database writes, it is a method of caching data already read from the database in RAM.

15.3.5.4. Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)?

If you don't use persistent connections when communicating with memcached, there will be a small increase in the latency of opening the connection each time. The effect is comparable to use nonpersistent connections with MySQL.

In general, the chance of locking or other issues with persistent connections is minimal, because there is very little locking within memcached. If there is a problem, eventually your request will time out and return no result, so your application will need to load from MySQL again.

15.3.5.5. How is an event such as a crash of one of the memcached servers handled by the memcached client?

There is no automatic handling of this. If your client fails to get a response from a server, code a fallback mechanism to load the data from the MySQL database.

The client APIs all provide the ability to add and remove memcached instances on the fly. If within your application you notice that memcached server is no longer responding, you can remove the server from the list of servers, and keys will automatically be redistributed to another memcached server in the list. If retaining the cache content on all your servers is important, make sure you use an API that supports a consistent hashing algorithm. For more information, see Section 15.3.2.4, “memcached Hashing/Distribution Types”.

15.3.5.6. What is a recommended hardware configuration for a memcached server?

memcached has a very low processing overhead. All that is required is spare physical RAM capacity. A memcached server does not require a dedicated machine. If you have web, application, or database servers that have spare RAM capacity, then use them with memcached.

To build and deploy a dedicated memcached server, use a relatively low-power CPU, lots of RAM, and one or more Gigabit Ethernet interfaces.

15.3.5.7. Is memcached more effective for video and audio as opposed to textual read/writes?

memcached works equally well for all kinds of data. To memcached, any value you store is just a stream of data. Remember, though, that the maximum size of an object you can store in memcached is 1MB, but can be configured to be larger by using the -I option in memcached 1.4.2

This documentation is for an older version. If you're
This documentation is for an older version. If you're

and later, or by modifying the source in versions before 1.4.2. If you plan on using memcached with audio and video content, you will probably want to increase the maximum object size. Also remember that memcached is a solution for caching information for reading. It shouldn't be used for writes, except when updating the information in the cache.

15.3.5.8. Can memcached work with ASPX?

There are ports and interfaces for many languages and environments. ASPX relies on an underlying language such as C# or VisualBasic, and if you are using ASP.NET then there is a C# memcached library. For more information, see https://sourceforge.net/projects/memcacheddotnet/.

15.3.5.9. How expensive is it to establish a memcache connection? Should those connections be pooled?

Opening the connection is relatively inexpensive, because there is no security, authentication or other handshake taking place before you can start sending requests and getting results. Most APIs support a persistent connection to a memcached instance to reduce the latency. Connection pooling would depend on the API you are using, but if you are communicating directly over TCP/IP, then connection pooling would provide some small performance benefit.

15.3.5.10. How is the data handled when the memcached server is down?

The behavior is entirely application dependent. Most applications fall back to loading the data from the database (just as if they were updating the memcached information). If you are using multiple memcached servers, you might also remove a downed server from the list to prevent it from affecting performance. Otherwise, the client will still attempt to communicate with the memcached server that corresponds to the key you are trying to load.

15.3.5.11. How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached?

They aren't. There is no relationship between MySQL and memcached unless your application (or, if you are using the MySQL UDFs for memcached, your database definition) creates one.

If you are storing information based on an auto-increment key into multiple instances of memcached, the information is only stored on one of the memcached instances anyway. The client uses the key value to determine which memcached instance to store the information. It doesn't store the same information across all the instances, as that would be a waste of cache memory.

15.3.5.12. Is compression available?

Yes. Most of the client APIs support some sort of compression, and some even allow you to specify the threshold at which a value is deemed appropriate for compression during storage.

15.3.5.13. Can we implement different types of memcached as different nodes in the same server, so can there be deterministic and non-deterministic in the same server?

Yes. You can run multiple instances of memcached on a single server, and in your client configuration you choose the list of servers you want to use.

15.3.5.14. What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of memcached configuration changes? And would you recommend keeping the configuration very simple to start?

The best way to test the performance is to start up a memcached instance. First, modify your application so that it stores the data just before the data is about to be used or displayed into memcached. Since the APIs handle the serialization of the data, it should just be a one-line modification to your code. Then, modify the start of the process that would normally load that

This documentation is for an older version. If you're

information from MySQL with the code that requests the data from memcached. If the data cannot be loaded from memcached, default to the MySQL process.

All of the changes required will probably amount to just a few lines of code. To get the best benefit, make sure you cache entire objects (for example, all the components of a web page, blog post, discussion thread, and so on), rather than using memcached as a simple cache of individual rows of MySQL tables.

Keeping the configuration simple at the start, or even over the long term, is easy with memcached. Once you have the basic structure up and running, often the only ongoing change is to add more servers into the list of servers used by your applications. You don't need to manage the memcached servers, and there is no complex configuration; just add more servers to the list and let the client API and the memcached servers make the decisions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 16 Replication

Table of Contents

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous - slaves need not be connected permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database.

For answers to some questions often asked by those who are new to MySQL Replication, see Section A.13, “MySQL 5.0 FAQ: Replication”.

The target uses for replication in MySQL include:

- Scale-out solutions - spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.
- Data security - because data is replicated to the slave, and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.
- Analytics - live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.
- Long-distance data distribution - if a branch office would like to work with a copy of your main data, you can use replication to create a local copy of the data for their use without requiring permanent access to the master.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see Chapter 17, *MySQL Cluster*).

There are a number of solutions available for setting up replication between two servers, but the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see Section 16.1.1, “How to Set Up Replication”.

Replication is controlled through a number of different options and variables. These control the core operation of the replication, timeouts, and the databases and filters that can be applied on databases and tables. For more information on the available options, see Section 16.1.2, “Replication and Binary Logging Options and Variables”.

You can use replication to solve a number of different problems, including problems with performance, supporting the backup of different databases, and as part of a larger solution to alleviate system failures. For information on how to address these issues, see Section 16.3, “Replication Solutions”.

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and problems and their resolution, including an FAQ, see Section 16.4, “Replication Notes and Tips”.

For detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replication, see Section 16.2, “Replication Implementation”.

16.1 Replication Configuration

Replication between servers in MySQL is based on the binary logging mechanism. The MySQL instance operating as the master (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave's local database.

The master is “dumb” in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each slave receives a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed; you cannot configure the master to log only certain events. If you do not specify otherwise, all events in the master binary log are executed on the slave. If required, you can configure the slave to process only events that apply to particular databases or tables.

Each slave keeps a record of the binary log coordinates: The file name and position within the file that it has read and processed from the master. This means that multiple slaves can be connected to the master and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then “catch up” by continuing from the recorded position.

Both the master and each slave must be configured with a unique ID (using the `server-id` option). In addition, each slave must be configured with information about the master host name, log file name, and position within that file. These details can be controlled from within a MySQL session using the `CHANGE MASTER TO` statement on the slave. The details are stored within the slave's `master.info` file.

This section describes the setup and configuration required for a replication environment, including step-by-step instructions for creating a new replication environment. The major components of this section are:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- For a guide to setting up two or more servers for replication, Section 16.1.1, “How to Set Up Replication”, deals with the configuration of the systems and provides methods for copying data between the master and slaves.
- Detailed information on the different configuration options and variables that apply to replication is provided in Section 16.1.2, “Replication and Binary Logging Options and Variables”.
- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to execute, see Section 16.1.3, “Common Replication Administration Tasks”.

16.1.1 How to Set Up Replication

This section describes how to set up complete replication of a MySQL server. There are a number of different methods for setting up replication, and the exact method to use depends on how you are setting up replication, and whether you already have data within your master database.

There are some generic tasks that are common to all replication setups:

- On the master, you must enable binary logging and configure a unique server ID. This might require a server restart. See Section 16.1.1.1, “Setting the Replication Master Configuration”.
- On each slave that you want to connect to the master, you must configure a unique server ID. This might require a server restart. See Section 16.1.1.2, “Setting the Replication Slave Configuration”.
- You may want to create a separate user that will be used by your slaves to authenticate with the master to read the binary log for replication. The step is optional. See Section 16.1.1.3, “Creating a User for Replication”.
- Before creating a data snapshot or starting the replication process, you should record the position of the binary log on the master. You will need this information when configuring the slave so that the slave knows where within the binary log to start executing events. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.
- If you already have data on your master and you want to use it to synchronize your slave, you will need to create a data snapshot. You can create a snapshot using `mysqldump` (see Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”) or by copying the data files directly (see Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”).
- You will need to configure the slave with settings for connecting to the master, such as the host name, login credentials, and binary log file name and position. See Section 16.1.1.10, “Setting the Master Configuration on the Slave”.

Once you have configured the basic options, you will need to follow the instructions for your replication setup. A number of alternatives are provided:

- If you are establishing a new MySQL master and one or more slaves, you need only set up the configuration, as you have no data to exchange. For guidance on setting up replication in this situation, see Section 16.1.1.7, “Setting Up Replication with New Master and Slaves”.
- If you are already running a MySQL server, and therefore already have data that must be transferred to your slaves before replication starts, have not previously configured the binary log and are able to shut down your MySQL server for a short period during the process, see Section 16.1.1.8, “Setting Up Replication with Existing Data”.
- If you are adding slaves to an existing replication environment, you can set up the slaves without affecting the master. See Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you will be administering MySQL replication servers, we suggest that you read this entire chapter through and try all statements mentioned in Section 13.4.1, “SQL Statements for Controlling Master Servers”, and Section 13.4.2, “SQL Statements for Controlling Slave Servers”. You should also familiarize yourself with the replication startup options described in Section 16.1.2, “Replication and Binary Logging Options and Variables”.

Note

Note that certain steps within the setup process require the `SUPER` privilege. If you do not have this privilege, it might not be possible to enable replication.

16.1.1.1 Setting the Replication Master Configuration

On a replication master, you must enable binary logging and establish a unique server ID. If this has not already been done, this part of master setup requires a server restart.

Binary logging *must* be enabled on the master because the binary log is the basis for sending data changes from the master to its slaves. If binary logging is not enabled, replication will not be possible.

Each server within a replication group must be configured with a unique server ID. This ID is used to identify individual servers within the group, and must be a positive integer between 1 and (2^{32})−1. How you organize and select the numbers is entirely up to you.

To configure the binary log and server ID options, you will need to shut down your MySQL server and edit the `my.cnf` or `my.ini` file. Add the following options to the configuration file within the `[mysqld]` section. If these options already exist, but are commented out, uncomment the options and alter them according to your needs. For example, to enable binary logging using a log file name prefix of `mysql-bin`, and configure a server ID of 1, use these lines:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

After making the changes, restart the server.

Note

If you omit `server-id` (or set it explicitly to its default value of 0), a master refuses connections from all slaves.

Note

For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in the master `my.cnf` file.

Note

Ensure that the `skip-networking` option is not enabled on your replication master. If networking has been disabled, your slave will not able to communicate with the master and replication will fail.

16.1.1.2 Setting the Replication Slave Configuration

On a replication slave, you must establish a unique server ID. If this has not already been done, this part of slave setup requires a server restart.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the slave server ID is not already set, or the current value conflicts with the value that you have chosen for the master server, you should shut down your slave server and edit the configuration to specify a unique server ID. For example:

```
[mysqld]
server-id=2
```

After making the changes, restart the server.

If you are setting up multiple slaves, each one must have a unique `server-id` value that differs from that of the master and from each of the other slaves. Think of `server-id` values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

Note

If you omit `server-id` (or set it explicitly to its default value of 0), a slave refuses to connect to a master.

You do not have to enable binary logging on the slave for replication to be enabled. However, if you enable binary logging on the slave, you can use the binary log for data backups and crash recovery on the slave, and also use the slave as part of a more complex replication topology (for example, where the slave acts as a master to other slaves).

16.1.1.3 Creating a User for Replication

Each slave must connect to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect. Any account can be used for this operation, providing it has been granted the `REPLICATION SLAVE` privilege. You may wish to create a different account for each slave, or connect to the master using the same account for each slave.

You need not create an account specifically for replication. However, you should be aware that the user name and password will be stored in plain text within the `master.info` file (see Section 16.2.2.2, “Slave Status Logs”). Therefore, you may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new acccount, use `CREATE USER`. To grant this account the privileges required for replication, use the `GRANT` statement. If you create an account solely for the purposes of replication, that account needs only the `REPLICATION SLAVE` privilege. For example, to set up a new user, `repl`, that can connect for replication from any host within the `mydomain.com` domain, issue these statements on the master:

```
mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.mydomain.com';
```

See Section 13.7.1, “Account Management Statements”, for more information on statements for manipulation of user accounts.

16.1.1.4 Obtaining the Replication Master Binary Log Coordinates

To configure replication on the slave you must determine the master's current coordinates within its binary log. You will need this information so that when the slave starts the replication process, it is able to start processing events from the binary log at the correct point.

If you have existing data on your master that you want to synchronize on your slaves before starting the replication process, you must stop processing statements on the master, and then obtain its current binary

This documentation is for an older version. If you're

log coordinates and dump its data, before permitting the master to continue executing statements. If you do not stop the execution of statements, the data dump and the master status information that you use will not match and you will end up with inconsistent or corrupted databases on the slaves.

To obtain the master binary log coordinates, follow these steps:

1. Start a session on the master by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   For `InnoDB` tables, note that `FLUSH TABLES WITH READ LOCK` also blocks `COMMIT` operations.

   Warning

   Leave the client from which you issued the `FLUSH TABLES` statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. In a different session on the master, use the `SHOW MASTER STATUS` statement to determine the current binary log file name and position:

   ```
   mysql > SHOW MASTER STATUS;
   +------------------+----------+--------------+------------------+
   | File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
   +------------------+----------+--------------+------------------+
   | mysql-bin.000003 | 73       | test         | manual,mysql     |
   +------------------+----------+--------------+------------------+
   ```

   The `File` column shows the name of the log file and `Position` shows the position within the file. In this example, the binary log file is `mysql-bin.000003` and the position is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

   If the master has been running previously without binary logging enabled, the log file name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`''`) and `4`.

You now have the information you need to enable the slave to start reading from the binary log in the correct place to start replication.

If you have existing data that needs be to synchronized with the slave before you start replication, leave the client running so that the lock remains in place and then proceed to Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”, or Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”. The idea here is to prevent any further changes so that the data copied to the slaves is in synchrony with the master.

If you are setting up a brand new master and slave replication group, you can exit the first session to release the read lock.

16.1.1.5 Creating a Data Snapshot Using mysqldump

One way to create a snapshot of the data in an existing master database is to use the `mysqldump` tool to create a dump of all the databases you want to replicate. Once the data dump has been completed, you then import this data into the slave before starting the replication process.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The example shown here dumps all databases to a file named `dbdump.db`, and includes the `--master-data` option which automatically appends the `CHANGE MASTER TO` statement required on the slave to start the replication process:

```
shell> mysqldump --all-databases --master-data > dbdump.db
```

If you do not use `--master-data`, then it is necessary to lock all tables in a separate session manually (using `FLUSH TABLES WITH READ LOCK`) prior to running `mysqldump`, then exiting or running UNLOCK TABLES from the second session to release the locks. You must also obtain binary log position information matching the snapshot, using `SHOW MASTER STATUS`, and use this to issue the appropriate `CHANGE MASTER TO` statement when starting the slave.

When choosing databases to include in the dump, remember that you need to filter out databases on each slave that you do not want to include in the replication process.

To import the data, either copy the dump file to the slave, or access the file from the master when connecting remotely to the slave.

16.1.1.6 Creating a Data Snapshot Using Raw Data Files

If your database is particularly large, copying the raw data files may be more efficient than using `mysqldump` and importing the file on each slave.

However, using this method with tables in storage engines with complex caching or logging algorithms may not give you a perfect “in time” snapshot as cache information and logging updates may not have been applied, even if you have acquired a global read lock. How the storage engine responds to this depends on its crash recovery abilities.

In addition, this method does not work reliably if the master and slave have different values for `ft_stopword_file`, `ft_min_word_len`, or `ft_max_word_len` and you are copying tables having full-text indexes.

If you are using `InnoDB` tables, you can use the MySQL Enterprise Backup tool to obtain a consistent snapshot. This tool records the log name and offset corresponding to the snapshot to be later used on the slave. MySQL Enterprise Backup is a nonfree (commercial) tool that is not included in the standard MySQL distribution. See Section 22.2, “MySQL Enterprise Backup Overview” for detailed information.

Otherwise, you can obtain a reliable binary snapshot of `InnoDB` tables only after shutting down the MySQL Server.

To create a raw data snapshot of `MyISAM` tables you can use standard copy tools such as `cp` or `copy`, a remote copy tool such as `scp` or `rsync`, an archiving tool such as `zip` or `tar`, or a file system snapshot tool such as `dump`, providing that your MySQL data files exist on a single file system. If you are replicating only certain databases then make sure you copy only those files that related to those tables. (For `InnoDB`, all tables in all databases are stored in the shared tablespace files, unless you have the `innodb_file_per_table` option enabled.)

You may want to specifically exclude the following files from your archive:

- Files relating to the `mysql` database.
- The `master.info` file.
- The master's binary log files.
- Any relay log files.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To get the most consistent results with a raw data snapshot you should shut down the master server during the process, as follows:

1. Acquire a read lock and get the master's status. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.
2. In a separate session, shut down the master server:

```
shell> mysqladmin shutdown
```

3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. Restart the master server.

If you are not using `InnoDB` tables, you can get a snapshot of the system from a master without shutting down the server as described in the following steps:

1. Acquire a read lock and get the master's status. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.
2. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

3. In the client where you acquired the read lock, release the lock:

```
mysql> UNLOCK TABLES;
```

Once you have created the archive or copy of the database, you will need to copy the files to each slave before starting the slave replication process.

16.1.1.7 Setting Up Replication with New Master and Slaves

The easiest and most straightforward method for setting up replication is to use new master and slave servers.

You can also use this method if you are setting up new servers but have an existing dump of the databases from a different server that you want to load into your replication configuration. By loading the data into a new master, the data will be automatically replicated to the slaves.

To set up replication between a new master and slave:

1. Configure the MySQL master with the necessary configuration properties. See Section 16.1.1.1, “Setting the Replication Master Configuration”.
2. Start up the MySQL master.
3. Set up a user. See Section 16.1.1.3, “Creating a User for Replication”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

4. Obtain the master status information. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.

5. On the master, release the read lock:

```
mysql> UNLOCK TABLES;
```

6. On the slave, edit the MySQL configuration. See Section 16.1.1.2, “Setting the Replication Slave Configuration”.

7. Start up the MySQL slave.

8. Execute a `CHANGE MASTER TO` statement to set the master replication server configuration. See Section 16.1.1.10, “Setting the Master Configuration on the Slave”.

Perform the slave setup steps on each slave.

Because there is no data to load or exchange on a new server configuration you do not need to copy or import any information.

If you are setting up a new replication environment using the data from a different existing database server, you will now need to run the dump file generated from that server on the new master. The database updates will automatically be propagated to the slaves:

```
shell> mysql -h master < fulldb.dump
```

16.1.1.8 Setting Up Replication with Existing Data

When setting up replication with existing data, you will need to decide how best to get the data from the master to the slave before starting the replication service.

The basic process for setting up replication with existing data is as follows:

1. With the MySQL master running, create a user to be used by the slave when connecting to the master during replication. See Section 16.1.1.3, “Creating a User for Replication”.

2. If you have not already configured the `server-id` and enabled binary logging on the master server, you will need to shut it down to configure these options. See Section 16.1.1.1, “Setting the Replication Master Configuration”.

   If you have to shut down your master server, this is a good opportunity to take a snapshot of its databases. You should obtain the master status (see Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”) before taking down the master, updating the configuration and taking a snapshot. For information on how to create a snapshot using raw data files, see Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”.

3. If your master server is already correctly configured, obtain its status (see Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”) and then use `mysqldump` to take a snapshot (see Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”) or take a raw snapshot of the live server using the guide in Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”.

4. Update the configuration of the slave. See Section 16.1.1.2, “Setting the Replication Slave Configuration”.

5. The next step depends on how you created the snapshot of data on the master.

   If you used `mysqldump`:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

a. Start the slave, using the `--skip-slave-start` option so that replication does not start.

b. Import the dump file:

```
shell> mysql < fulldb.dump
```

If you created a snapshot using the raw data files:

a. Extract the data files into your slave data directory. For example:

```
shell> tar xvf dbdump.tar
```

You may need to set permissions and ownership on the files so that the slave server can access and modify them.

b. Start the slave, using the `--skip-slave-start` option so that replication does not start.

6. Configure the slave with the replication coordinates from the master. This tells the slave the binary log file and position within the file where replication needs to start. Also, configure the slave with the login credentials and host name of the master. For more information on the `CHANGE MASTER TO` statement required, see Section 16.1.1.10, “Setting the Master Configuration on the Slave”.

7. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` option for the master, slaves cannot connect to it.

If you have forgotten to set the `server-id` option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. Do *not* remove or edit these files unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave will use the values specified in the statement to update the status files automatically.

Note

The content of `master.info` overrides some of the server options specified on the command line or in `my.cnf`. See Section 16.1.2, “Replication and Binary Logging Options and Variables”, for more details.

A single snapshot of the master suffices for multiple slaves. To set up additional slaves, use the same master snapshot and follow the slave portion of the procedure just described.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

16.1.1.9 Introducing Additional Slaves to an Existing Replication Environment

To add another slave to an existing replication configuration, you can do so without stopping the master. Instead, set up the new slave by making a copy of an existing slave, except that you configure the new slave with a different `server-id` value.

To duplicate an existing slave:

1. Shut down the existing slave:

   ```
   shell> mysqladmin shutdown
   ```

2. Copy the data directory from the existing slave to the new slave. You can do this by creating an archive using `tar` or `WinZip`, or by performing a direct copy using a tool such as `cp` or `rsync`. Ensure that you also copy the log files and relay log files.

   A common problem that is encountered when adding new replication slaves is that the new slave fails with a series of warning and error messages like these:

   ```
   071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
   replication may break when this MySQL server acts as a slave and has his hostname
   changed!! Please use '--relay-log=new_slave_hostname-relay-bin' to avoid this problem.
   071118 16:44:10 [ERROR] Failed to open the relay log './old_slave_hostname-relay-bin.003525'
   (relay_log_pos 22940879)
   071118 16:44:10 [ERROR] Could not find target log during relay log initialization
   071118 16:44:10 [ERROR] Failed to initialize the master info structure
   ```

   This is due to the fact that, if the `--relay-log` option is not specified, the relay log files contain the host name as part of their file names. (This is also true of the relay log index file if the `--relay-log-index` option is not used. See Section 16.1.2, “Replication and Binary Logging Options and Variables”, for more information about these options.)

   To avoid this problem, use the same value for `--relay-log` on the new slave that was used on the existing slave. (If this option was not set explicitly on the existing slave, use *`existing_slave_hostname`*`-relay-bin`.) If this is not feasible, copy the existing slave's relay log index file to the new slave and set the `--relay-log-index` option on the new slave to match what was used on the existing slave. (If this option was not set explicitly on the existing slave, use *`existing_slave_hostname`*`-relay-bin.index`.) Alternatively—if you have already tried to start the new slave (after following the remaining steps in this section) and have encountered errors like those described previously—then perform the following steps:

   a. If you have not already done so, issue a `STOP SLAVE` on the new slave.

      If you have already started the existing slave again, issue a `STOP SLAVE` on the existing slave as well.

   b. Copy the contents of the existing slave's relay log index file into the new slave's relay log index file, making sure to overwrite any content already in the file.

   c. Proceed with the remaining steps in this section.

3. Copy the `master.info` and `relay-log.info` files from the existing slave to the new slave if they were not located in the data directory. These files hold the current log coordinates for the master's binary log and the slave's relay log.

4. Start the existing slave.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

5. On the new slave, edit the configuration and give the new slave a unique `server-id` not used by the master or any of the existing slaves.

6. Start the new slave. The slave will use the information in its `master.info` file to start the replication process.

16.1.1.10 Setting the Master Configuration on the Slave

To set up the slave to communicate with the master for replication, you must tell the slave the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

The `CHANGE MASTER TO` statement has other options as well. For example, it is possible to set up secure replication using SSL. For a full list of options, and information about the maximum permissible length for the string-valued options, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

16.1.2 Replication and Binary Logging Options and Variables

The next few sections contain information about `mysqld` options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication masters and replication slaves are covered separately, as are options and variables relating to binary logging. A set of quick-reference tables providing basic information about these options and variables is also included (in the next section following this one).

Of particular importance is the `--server-id` option.

Command-Line Format	`--server-id=#`	
System Variable	**Name**	`server_id`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`

This option is common to both master and slave replication servers, and is used in replication to enable master and slave servers to identify themselves uniquely. For additional information, see Section 16.1.2.2,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

"Replication Master Options and Variables", and Section 16.1.2.3, "Replication Slave Options and Variables".

On the master and each slave, you *must* use the `--server-id` option to establish a unique replication ID in the range from 1 to $2^{32} - 1$. "Unique", means that each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

If you omit `--server-id`, the default ID is 0, in which case the master refuses connections from all slaves, and slaves refuse to connect to the master. In MySQL 5.0, whether the server ID is set to 0 explicitly or the default is allowed to be used, the server sets the `server_id` system variable to 1; this is a known issue that is fixed in MySQL 5.7.

For more information, see Section 16.1.1.2, "Setting the Replication Slave Configuration".

16.1.2.1 Replication and Binary Logging Option and Variable Reference

The following tables list basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

Table 16.1 Summary of Replication options and variables in MySQL 5.0

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
`abort-slave-event-count`		
Yes	No	No
Yes		No
DESCRIPTION: Option used by mysql-test for debugging and testing of replication		
`Com_change_master`		
No	No	Yes
No	Both	No
DESCRIPTION: Count of CHANGE MASTER TO statements		
`Com_show_master_status`		
No	No	Yes
No	Both	No
DESCRIPTION: Count of SHOW MASTER STATUS statements		
`Com_show_new_master`		
No	No	Yes
No	Both	No
DESCRIPTION: Count of SHOW NEW MASTER statements		
`Com_show_slave_hosts`		
No	No	Yes
No	Both	No
DESCRIPTION: Count of SHOW SLAVE HOSTS statements		
`Com_show_slave_status`		
No	No	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
No	Both	No
DESCRIPTION: Count of SHOW SLAVE STATUS statements		
Com_slave_start		
No	No	Yes
No	Both	No
DESCRIPTION: Count of START SLAVE statements		
Com_slave_stop		
No	No	Yes
No	Both	No
DESCRIPTION: Count of STOP SLAVE statements		
disconnect-slave-event-count		
Yes	No	No
Yes		No
DESCRIPTION: Option used by mysql-test for debugging and testing of replication		
init_slave		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Statements that are executed when a slave connects to a master		
log-slave-updates		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Tells the slave to log the updates performed by its SQL thread to its own binary log		
log_slave_updates		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Whether the slave should log the updates performed by its SQL thread to its own binary log. Read-only; set using the --log-slave-updates server option.		
master-connect-retry		
Yes	No	No
Yes		No
DESCRIPTION: Number of seconds the slave thread will sleep before retrying to connect to the master in case the master goes down or the connection is lost		
master-host		
Yes	No	No
Yes		No
DESCRIPTION: Master host name or IP address for replication		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
`master-info-file`		
Yes	No	No
Yes		No
DESCRIPTION: The location and name of the file that remembers the master and where the I/O replication thread is in the master's binary logs		
`master-password`		
Yes	No	No
Yes		No
DESCRIPTION: The password the slave thread will authenticate with when connecting to master		
`master-port`		
Yes	No	No
Yes		No
DESCRIPTION: The port the master is listening on		
`master-retry-count`		
Yes	No	No
Yes		No
DESCRIPTION: Number of tries the slave makes to connect to the master before giving up		
`master-ssl`		
Yes	No	No
Yes		No
DESCRIPTION: Enable the slave to connect to the master using SSL		
`master-ssl-ca`		
Yes	No	No
Yes		No
DESCRIPTION: Master SSL CA file; applies only if master-ssl is enabled		
`master-ssl-capath`		
Yes	No	No
Yes		No
DESCRIPTION: Master SSL CA path; applies only if master-ssl is enabled		
`master-ssl-cert`		
Yes	No	No
Yes		No
DESCRIPTION: Master SSL certificate file name; applies only if master-ssl is enabled		
`master-ssl-cipher`		
Yes	No	No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
Yes		No
DESCRIPTION: Master SSL cipher; applies only if master-ssl is enabled `master-ssl-key`		
Yes	No	No
Yes		No
DESCRIPTION: Master SSL key file name; applies only if master-ssl is enabled `master-user`		
Yes	No	No
Yes		No
DESCRIPTION: The user name the slave thread will use for authentication when connecting to master. The user must have FILE privilege. If the master user is not set, user test is assumed. The value in master.info will take precedence if it can be read `relay-log`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: The location and base name to use for relay logs `relay-log-index`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: The location and name to use for the file that keeps a list of the last relay logs `relay-log-info-file`		
Yes	No	No
Yes		No
DESCRIPTION: The location and name of the file that remembers where the SQL replication thread is in the relay logs `relay_log_index`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: The name of the relay log index file `relay_log_info_file`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: The name of the file in which the slave records information about the relay logs `relay_log_purge`		
Yes	Yes	No
Yes	Global	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
DESCRIPTION: Determines whether relay logs are purged `relay_log_space_limit`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Maximum space to use for all relay logs `replicate-do-db`		
Yes	No	No
Yes		No
DESCRIPTION: Tells the slave SQL thread to restrict replication to the specified database `replicate-do-table`		
Yes	No	No
Yes		No
DESCRIPTION: Tells the slave SQL thread to restrict replication to the specified table `replicate-ignore-db`		
Yes	No	No
Yes		No
DESCRIPTION: Tells the slave SQL thread not to replicate to the specified database `replicate-ignore-table`		
Yes	No	No
Yes		No
DESCRIPTION: Tells the slave SQL thread not to replicate to the specified table `replicate-rewrite-db`		
Yes	No	No
Yes		No
DESCRIPTION: Updates to a database with a different name than the original `replicate-same-server-id`		
Yes	No	No
Yes		No
DESCRIPTION: In replication, if set to 1, do not skip events having our server id `replicate-wild-do-table`		
Yes	No	No
Yes		No
DESCRIPTION: Tells the slave thread to restrict replication to the tables that match the specified wildcard pattern `replicate-wild-ignore-table`		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
Yes	No	No
Yes		No
DESCRIPTION: Tells the slave thread not to replicate to the tables that match the given wildcard pattern		
`report-host`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Host name or IP of the slave to be reported to the master during slave registration		
`report-password`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: An arbitrary password that the slave server should report to the master. Not the same as the password for the MySQL replication user account.		
`report-port`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Port for connecting to slave reported to the master during slave registration		
`report-user`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: An arbitrary user name that a slave server should report to the master. Not the same as the name used with the MySQL replication user account.		
`rpl_recovery_rank`		
No	Yes	No
No	Global	Yes
DESCRIPTION: Not used; removed in later versions		
`Rpl_status`		
No	No	Yes
No	Global	No
DESCRIPTION: The status of fail-safe replication (not implemented)		
`show-slave-auth-info`		
Yes	No	No
Yes		No
DESCRIPTION: Show user name and password in SHOW SLAVE HOSTS on this master		
`skip-slave-start`		
Yes	No	No
Yes		No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
DESCRIPTION: If set, slave is not autostarted		
`slave-load-tmpdir`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: The location where the slave should put its temporary files when replicating a LOAD DATA INFILE statement		
`slave_net_timeout`		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Number of seconds to wait for more data from a master/slave connection before aborting the read		
`slave-skip-errors`		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Tells the slave thread to continue replication when a query returns an error from the provided list		
`slave_compressed_protocol`		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Use compression on master/slave protocol		
`Slave_open_temp_tables`		
No	No	Yes
No	Global	No
DESCRIPTION: Number of temporary tables that the slave SQL thread currently has open		
`Slave_retried_transactions`		
No	No	Yes
No	Global	No
DESCRIPTION: The total number of times since startup that the replication slave SQL thread has retried transactions		
`Slave_running`		
No	No	Yes
No	Global	No
DESCRIPTION: The state of this server as a replication slave (slave I/O thread status)		
`slave_transaction_retries`		
Yes	Yes	No
Yes	Global	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
DESCRIPTION: Number of times the slave SQL thread will retry a transaction in case it failed with a deadlock or elapsed lock wait timeout, before giving up and stopping		
`sql_slave_skip_counter`		
No	Yes	No
No	Global	Yes
DESCRIPTION: Number of events from the master that a slave server should skip. Not compatible with GTID replication.		
`sync_binlog`		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Synchronously flush binary log to disk after every #th event		

Section 16.1.2.2, “Replication Master Options and Variables”, provides more detailed information about options and variables relating to replication master servers. For more information about options and variables relating to replication slaves, see Section 16.1.2.3, “Replication Slave Options and Variables”.

Table 16.2 Summary of Binary Logging options and variables in MySQL 5.0

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
`binlog-do-db`		
Yes	No	No
Yes		No
DESCRIPTION: Limits binary logging to specific databases		
`binlog-ignore-db`		
Yes	No	No
Yes		No
DESCRIPTION: Tells the master that updates to the given database should not be logged to the binary log		
`Binlog_cache_disk_use`		
No	No	Yes
No	Global	No
DESCRIPTION: Number of transactions that used a temporary file instead of the binary log cache		
`binlog_cache_size`		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Size of the cache to hold the SQL statements for the binary log during a transaction		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
`Binlog_cache_use`		
No	No	Yes
No	Global	No
DESCRIPTION: Number of transactions that used the temporary binary log cache		
`Com_show_binlog_events`		
No	No	Yes
No	Both	No
DESCRIPTION: Count of SHOW BINLOG EVENTS statements		
`Com_show_binlogs`		
No	No	Yes
No	Both	No
DESCRIPTION: Count of SHOW BINLOGS statements		
`max-binlog-dump-events`		
Yes	No	No
Yes		No
DESCRIPTION: Option used by mysql-test for debugging and testing of replication		
`max_binlog_cache_size`		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Can be used to restrict the total size used to cache a multi-statement transaction		
`max_binlog_size`		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Binary log will be rotated automatically when size exceeds this value		
`sporadic-binlog-dump-fail`		
Yes	No	No
Yes		No
DESCRIPTION: Option used by mysql-test for debugging and testing of replication		

Section 16.1.2.4, “Binary Log Options and Variables”, provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see Section 5.4.3, “The Binary Log”.

For information about the `sql_log_bin` and `sql_log_off` variables, see Section 5.1.4, “Server System Variables”.

For a table showing *all* command-line options, system and status variables used with `mysqld`, see Section 5.1.1, “Server Option and Variable Reference”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

16.1.2.2 Replication Master Options and Variables

This section describes the server options and system variables that you can use on replication master servers. You can specify the options either on the command line or in an option file. You can specify system variable values using `SET`.

On the master and each slave, you must use the `server-id` option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to $2^{32} - 1$, and each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

For options used on the master for controlling binary logging, see Section 16.1.2.4, “Binary Log Options and Variables”.

System Variables Used on Replication Masters

The following system variables are used in controlling replication masters:

- `auto_increment_increment`

Introduced	5.0.2	
System Variable	**Name**	`auto_increment_increment`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`1`
	Min Value	`1`
	Max Value	`65535`

`auto_increment_increment` and `auto_increment_offset` are intended for use with master-to-master replication, and can be used to control the operation of `AUTO_INCREMENT` columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of `auto_increment_increment` or `auto_increment_offset` to a noninteger value gives rise to an error, and the actual value of the variable remains unchanged.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> CREATE TABLE autoinc1
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
  Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)
```

- `auto_increment_offset` determines the starting point for the `AUTO_INCREMENT` column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for `auto_increment_increment`:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
|   5 |
|  15 |
|  25 |
|  35 |
+-----+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
4 rows in set (0.02 sec)
```

If the value of `auto_increment_offset` is greater than that of `auto_increment_increment`, the value of `auto_increment_offset` is ignored.

Should one or both of these variables be changed and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. In other words, the series is calculated like so:

`auto_increment_offset` + *N* × `auto_increment_increment`

where *N* is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
|  35 |
|  45 |
|  55 |
|  65 |
+-----+
8 rows in set (0.00 sec)
```

The values shown for `auto_increment_increment` and `auto_increment_offset` generate the series 5 + *N* × 10, that is, [5, 15, 25, 35, 45, ...]. The greatest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is not possible to confine the effects of these two variables to a single table, and thus they do not take the place of the sequences offered by some other database management systems; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If the global

This documentation is for an older version. If you're
This documentation is for an older version. If you're

value of either variable is set, its effects persist until the global value is changed or overridden by setting the session value, or until `mysqld` is restarted. If the local value is set, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The `auto_increment_increment` variable was added in MySQL 5.0.2. Its default value is 1. See Section 16.4.1.1, “Replication and AUTO_INCREMENT”.

`auto_increment_increment` is supported for use with `NDB` tables beginning with MySQL 5.0.46. Previously, setting it when using MySQL Cluster tables produced unpredictable results.

- `auto_increment_offset`

Introduced	5.0.2	
System Variable	**Name**	`auto_increment_offset`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`1`
	Min Value	`1`
	Max Value	`65535`

This variable was introduced in MySQL 5.0.2. Its default value is 1. For particulars, see the description for `auto_increment_increment`.

`auto_increment_offset` is supported for use with `NDB` tables beginning with MySQL 5.0.46. Previously, setting it when using MySQL Cluster tables produced unpredictable results.

16.1.2.3 Replication Slave Options and Variables

Startup Options for Replication Slaves

System Variables Used on Replication Slaves

This section describes the server options and system variables that apply to slave replication servers. You can specify the options either on the command line or in an option file. Many of the options can be set while the server is running by using the `CHANGE MASTER TO` statement. You can specify system variable values using `SET`.

Server ID. On the master and each slave, you must use the `server-id` option to establish a unique replication ID in the range from 1 to $2^{32} - 1$. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. Example `my.cnf` file:

```
[mysqld]
server-id=3
```

Some slave server replication options are handled in a special way, in the sense that each is ignored if a `master.info` file exists when the slave starts and contains a value for the option. The following options are handled this way:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`
- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

The `master.info` file format in MySQL 5.0 includes as its first line the number of lines in the file. (See Section 16.2.2, “Replication Relay and Status Logs”.) If you upgrade an older server (before MySQL 4.1.1) to a newer version, the new server upgrades the `master.info` file to the new format automatically when it starts. However, if you downgrade a newer server to a version older than 4.1.1, you should manually remove the first line before starting the older server for the first time. Note that, in this case, the downgraded server can no longer use an SSL connection to communicate with the master.

If no `master.info` file exists when the slave server starts, it uses the values for those options that are specified in option files or on the command line. This occurs when you start the server as a replication slave for the very first time, or when you have run `RESET SLAVE` and then have shut down and restarted the slave.

If the `master.info` file exists when the slave server starts, the server uses its contents and ignores any startup options that correspond to the values listed in the file. Thus, if you start the slave server with different values of the startup options that correspond to values in the `master.info` file, the different values have no effect because the server continues to use the `master.info` file. To use different values, the preferred method is to use the `CHANGE MASTER TO` statement to reset the values while the slave is running. Alternatively, you can stop the server, remove the `master.info` file, and restart the server with different option values.

Suppose that you specify this option in your `my.cnf` file:

```
[mysqld]
master-host=some_host
```

The first time you start the server as a replication slave, it reads and uses that option from the `my.cnf` file. The server then records the value in the `master.info` file. The next time you start the server, it reads the master host value from the `master.info` file only and ignores the value in the option file. If you modify the `my.cnf` file to specify a different master host of *`some_other_host`*, the change still has no effect. You should use `CHANGE MASTER TO` instead.

Because the server gives an existing `master.info` file precedence over the startup options just described, you might prefer not to use startup options for these values at all, and instead specify them by using the `CHANGE MASTER TO` statement. See Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

This example shows a more extensive use of startup options to configure a slave server:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

Startup Options for Replication Slaves

The following list describes startup options for controlling replication slave servers. Many of these options can be set while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts. Replication-related system variables are discussed later in this section.

- `--abort-slave-event-count`

Command-Line Format	`--abort-slave-event-count=#`	
Permitted Values	Type	`integer`
	Default	`0`
	Min Value	`0`

When this option is set to some positive integer *`value`* other than 0 (the default) it affects replication behavior as follows: After the slave SQL thread has started, *`value`* log events are permitted to be executed; after that, the slave SQL thread does not receive any more events, just as if the network connection from the master were cut. The slave thread continues to run, and the output from `SHOW SLAVE STATUS` displays `Yes` in both the `Slave_IO_Running` and the `Slave_SQL_Running` columns, but no further events are read from the relay log.

This option is used internally by the MySQL test suite for replication testing and debugging. It is not intended for use in a production setting.

- `--disconnect-slave-event-count`

Command-Line Format	`--disconnect-slave-event-count=#`	
Permitted Values	Type	`integer`
	Default	`0`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--log-slave-updates`

Command-Line Format	`--log-slave-updates`	
System Variable	Name	`log_slave_updates`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`boolean`
	Default	`OFF`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` option to enable binary logging. `--log-slave-updates` is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A -> B -> C
```

Here, `A` serves as the master for the slave `B`, and `B` serves as the master for the slave `C`. For this to work, `B` must be both a master *and* a slave. You must start both `A` and `B` with `--log-bin` to enable binary logging, and `B` with the `--log-slave-updates` option so that updates received from `A` are logged by `B` to its binary log.

- `--log-warnings[=level]`

Command-Line Format	`--log-warnings[=#]`	
System Variable	**Name**	`log_warnings`
	Variable Scope	Global, Session
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`1`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`1`
	Min Value	`0`
	Max Value	`18446744073709547520`

This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/connection failure, and informs you as to how each slave thread started. This option is enabled (1) by default; to disable it, use `--log-warnings=0`. Aborted connections are not logged to the error log unless the value is greater than 1.

Note that the effects of this option are not limited to replication. It produces warnings across a spectrum of server activities.

- `--master-connect-retry=seconds`

Command-Line Format	`--master-connect-retry=#`	
Permitted Values	**Type**	`integer`
	Default	`60`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The number of seconds that the slave thread sleeps before trying to reconnect to the master in case the master goes down or the connection is lost. The value in the `master.info` file takes precedence if it can be read. If not set, the default is 60. Connection retries are not invoked until the slave times out reading data from the master according to the value of `--slave-net-timeout`. The number of reconnection attempts is limited by the `--master-retry-count` option.

- `--master-host=host_name`

Command-Line Format	`--master-host=name`	
Permitted Values	Type	`string`

The host name or IP address of the master replication server. The value in `master.info` takes precedence if it can be read. If no master host is specified, the slave thread does not start.

- `--master-info-file=file_name`

Command-Line Format	`--master-info-file=file_name`	
Permitted Values	Type	`file name`
	Default	`master.info`

The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory. For information about the format of this file, see Section 16.2.2.2, “Slave Status Logs”.

- `--master-password=password`

Command-Line Format	`--master-password=name`	
Permitted Values	Type	`string`

The password of the account that the slave thread uses for authentication when it connects to the master. The value in the `master.info` file takes precedence if it can be read. If not set, an empty password is assumed.

- `--master-port=port_number`

Command-Line Format	`--master-port=#`	
Permitted Values	Type	`integer`
	Default	`3306`

The TCP/IP port number that the master is listening on. The value in the `master.info` file takes precedence if it can be read. If not set, the compiled-in setting is assumed (normally 3306).

- `--master-retry-count=count`

Command-Line Format	`--master-retry-count=#`	
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`86400`
	Min Value	`0`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Max Value	4294967295
Permitted Values (64-bit platforms)	**Type**	integer
	Default	86400
	Min Value	0
	Max Value	18446744073709551615

The number of times that the slave tries to connect to the master before giving up. Reconnects are attempted at intervals set by the `--master-connect-retry` option (or the `MASTER_CONNECT_RETRY` option of the `CHANGE MASTER TO` statement) and reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` option. The default value is 86400. A value of 0 means “infinite”; the slave attempts to connect forever.

- `--master-ssl`, `--master-ssl-ca=`*`file_name`*, `--master-ssl-capath=`*`directory_name`*, `--master-ssl-cert=`*`file_name`*, `--master-ssl-cipher=`*`cipher_list`*, `--master-ssl-key=`*`file_name`*

  These options are used for setting up a secure replication connection to the master server using SSL. Their meanings are the same as the corresponding `--ssl`, `--ssl-ca`, `--ssl-capath`, `--ssl-cert`, `--ssl-cipher`, `--ssl-key` options that are described in Section 6.3.6.5, “Command Options for Secure Connections”. The values in the `master.info` file take precedence if they can be read.

- `--master-user=`*`user_name`*

Command-Line Format	`--master-user=name`	
Permitted Values	**Type**	string
	Default	test

The user name of the account that the slave thread uses for authentication when it connects to the master. This account must have the `REPLICATION SLAVE` privilege. The value in the `master.info` file takes precedence if it can be read. If the master user name is not set, the name `test` is assumed.

- `--max-relay-log-size=`*`size`*

Command-Line Format	`--max_relay_log_size=#`	
System Variable	**Name**	max_relay_log_size
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	0
	Min Value	0
	Max Value	1073741824

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The size at which the server rotates relay log files automatically. If this value is nonzero, the relay log is rotated automatically when its size exceeds this value. If this value is zero (the default), the size at which relay log rotation occurs is determined by the value of `max_binlog_size`. For more information, see Section 16.2.2.1, “The Slave Relay Log”.

- `--relay-log=file_name`

Command-Line Format	--relay-log=file_name	
System Variable	Name	relay_log
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	file name

The base name for the relay log. The default base name is *`host_name`*`-relay-bin`. The server writes the file in the data directory unless the base name is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the base name.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default base name is used only if the option is not actually specified*. If you use the `--relay-log` option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.3, “Specifying Program Options”.

If you specify this option, the value specified is also used as the base name for the relay log index file. You can override this behavior by specifying a different relay log index file base name using the `--relay-log-index` option.

You may find the `--relay-log` option useful in performing the following tasks:

- Creating relay logs whose names are independent of host names.
- If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease `max_relay_log_size`.
- To increase speed by using load-balancing between disks.

- `--relay-log-index=file_name`

Command-Line Format	--relay-log-index=file_name	
System Variable	Name	relay_log_index
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	file name

This documentation is for an older version. If you're

The name to use for the relay log index file. The default name is *`host_name`*`-relay-bin.index` in the data directory, where *`host_name`* is the name of the slave server.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default base name is used only if the option is not actually specified*. If you use the `--relay-log-index` option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.3, "Specifying Program Options".

If you specify this option, the value specified is also used as the base name for the relay logs. You can override this behavior by specifying a different relay log file base name using the `--relay-log` option.

- `--relay-log-info-file=`*`file_name`*

Command-Line Format	`--relay-log-info-file=file_name`	
Permitted Values	**Type**	`file name`
	Default	`relay-log.info`

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory. For information about the format of this file, see Section 16.2.2.2, "Slave Status Logs".

- `--relay-log-purge={0|1}`

Command-Line Format	`--relay_log_purge`	
System Variable	**Name**	`relay_log_purge`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = `*`N`*.

- `--relay-log-space-limit=`*`size`*

Command-Line Format	`--relay_log_space_limit=#`	
System Variable	**Name**	`relay_log_space_limit`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`0`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`18446744073709547520`

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit.” This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` temporarily.

- `--replicate-do-db=`*`db_name`*

Command-Line Format	`--replicate-do-db=name`	
Permitted Values	**Type**	`string`

Tell the slave SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database. Note that this does not replicate cross-database statements such as `UPDATE` *`some_db.some_table`* `SET foo='bar'` while having selected a different database or no database.

Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

An example of what does not work as you might expect: If the slave is started with `--replicate-do-db=sales` and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` or multiple-table `UPDATE` statements that go across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` instead. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

Beginning with MySQL 5.0.84, this option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements. (Bug #43263)

- `--replicate-ignore-db=db_name`

Command-Line Format	`--replicate-ignore-db=name`	
Permitted Values	Type	`string`

Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database to ignore, use this option multiple times, once for each database. You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

An example of what does not work as you might expect: If the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master, the `UPDATE` statement *is* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

Note

In the preceding example the statement is replicated because `--replicate-ignore-db` only applies to the default database (set through the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=db_name.%` instead. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

Beginning with MySQL 5.0.84, this option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements. (Bug #43263)

- `--replicate-do-table=db_name.tbl_name`

Command-Line Format	`--replicate-do-table=name`	
Permitted Values	Type	`string`

Tells the slave SQL thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to `--replicate-do-db`. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-ignore-table=db_name.tbl_name`

Command-Line Format	`--replicate-ignore-table=name`

This documentation is for an older version. If you're

Permitted Values	Type	string

Tells the slave SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db`. See Section 16.2.3, "How Servers Evaluate Replication Filtering Rules".

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-rewrite-db=`*`from_name->to_name`*

Command-Line Format	--replicate-rewrite-db=old_name->new_name	
Permitted Values	Type	string

Tells the slave to translate the default database (that is, the one selected by `USE`) to *`to_name`* if it was *`from_name`* on the master. Only statements involving tables are affected (not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if *`from_name`* is the default database on the master. To specify multiple rewrites, use this option multiple times. The server uses the first one with a *`from_name`* value that matches. The database name translation is done *before* the `--replicate-*` rules are tested.

Statements in which table names are qualified with database names when using this option do not work with table-level replication filtering options such as `--replicate-do-table`. Suppose we have a database named `a` on the master, one named `b` on the slave, each containing a table `t`, and have started the master with `--replicate-rewrite-db='a->b'`. At a later point in time, we execute `DELETE FROM a.t`. In this case, no relevant filtering rule works, for the reasons shown here:

1. `--replicate-do-table=a.t` does not work because the slave has table `t` in database `b`.
2. `--replicate-do-table=b.t` does not match the original statement and so is ignored.
3. `--replicate-do-table=*.t` is handled identically to `--replicate-do-table=a.t`, and thus does not work, either.

Similarly, the `--replication-rewrite-db` option does not work with cross-database updates.

If you use this option on the command line and the "`>`" character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id`

Introduced	5.0.1	
Command-Line Format	--replicate-same-server-id	
Permitted Values	Type	boolean
	Default	FALSE

To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

used. By default, the slave I/O thread does not write binary log events to the relay log if they have the slave's server ID (this optimization helps save disk usage). If you want to use `--replicate-same-server-id`, be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name`

Command-Line Format	`--replicate-wild-do-table=name`	
Permitted Values	Type	`string`

Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the "%" and "_" wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See Section 16.2.3, "How Servers Evaluate Replication Filtering Rules".

This option applies to tables, views, and triggers. It does not apply to stored procedures and functions. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Example: `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%`, database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the "_" and "%" characters like this: `--replicate-wild-do-table=my\_own\%db`. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=db_name.tbl_name`

Command-Line Format	`--replicate-wild-ignore-table=name`	
Permitted Values	Type	`string`

Tells the slave thread not to replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See Section 16.2.3, "How Servers Evaluate Replication Filtering Rules".

Example: `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--report-host=host_name`

This documentation is for an older version. If you're

Command-Line Format	`--report-host=host_name`

This documentation is for an older version. If you're

Permitted Values	Type	string

The host name or IP address of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP address of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `--report-password=`*`password`*

Command-Line Format	--report-password=name	
Permitted Values	Type	string

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` option is given.

Although the name of this option might imply otherwise, `--report-password` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the password for the MySQL replication user account.

- `--report-port=`*`slave_port_num`*

Command-Line Format	--report-port=#	
Permitted Values	Type	integer
	Default	3306
	Min Value	0
	Max Value	65535

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a nondefault port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

- `--report-user=`*`user_name`*

Command-Line Format	--report-user=name	
Permitted Values	Type	string

The account user name of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` option is given.

Although the name of this option might imply otherwise, `--report-user` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the name of the MySQL replication user account.

- `--show-slave-auth-info`

This documentation is for an older version. If you're

Command-Line Format	--show-slave-auth-info

This documentation is for an older version. If you're

Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Display slave user names and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` and `--report-password` options.

- `--skip-slave-start`

Command-Line Format	`--skip-slave-start`	
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}`

Command-Line Format	`--slave_compressed_protocol`	
System Variable	**Name**	`slave_compressed_protocol`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`
	Default	`OFF`

If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it. The default is 0 (no compression).

- `--slave-load-tmpdir=`*`dir_name`*

Command-Line Format	`--slave-load-tmpdir=dir_name`	
System Variable	**Name**	`slave_load_tmpdir`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`
	Default	`/tmp`

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` option to place the relay logs in that file system.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=`*`seconds`*

Command-Line Format	`--slave-net-timeout=#`	
System Variable	Name	`slave_net_timeout`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`3600`
	Min Value	`1`

The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `CHANGE MASTER TO` statement or `--master-connect-retry` option and the number of reconnection attempts is limited by the `--master-retry-count` option. The default is 3600 seconds (one hour).

- `--slave-skip-errors=[`*`err_code1`*`,`*`err_code2`*`,...|all]`

Command-Line Format	`--slave-skip-errors=name`	
System Variable	Name	`slave_skip_errors`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`string`
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. Appendix B, *Errors, Error Codes, and Common Problems*, lists server error codes.

You can also (but should not) use the very nonrecommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned*.

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

System Variables Used on Replication Slaves

The following list describes system variables for controlling replication slave servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replication slaves are listed earlier in this section.

- `init_slave`

Command-Line Format	`--init-slave=name`	
System Variable	**Name**	`init_slave`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`string`

This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable.

Note

The SQL thread sends an acknowledgment to the client before it executes `init_slave`. Therefore, it is not guaranteed that `init_slave` has been executed when `START SLAVE` returns. See Section 13.4.2.7, “START SLAVE Syntax”, for more information.

- `relay_log`

Command-Line Format	`--relay-log=file_name`	
System Variable	**Name**	`relay_log`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

The name of the relay log file.

This documentation is for an older version. If you're

- `relay_log_index`

Command-Line Format	`--relay-log-index`	
System Variable	Name	`relay_log_index`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`file name`
	Default	`*host_name*-relay-bin.index`

The name of the relay log index file. The default name is *`host_name`*`-relay-bin.index` in the data directory, where *`host_name`* is the name of the slave server.

- `relay_log_info_file`

Command-Line Format	`--relay-log-info-file=file_name`	
System Variable	Name	`relay_log_info_file`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`file name`
	Default	`relay-log.info`

The name of the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `rpl_recovery_rank`

This variable is unused, and is removed in MySQL 5.6.

- `slave_compressed_protocol`

Command-Line Format	`--slave_compressed_protocol`	
System Variable	Name	`slave_compressed_protocol`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`boolean`
	Default	`OFF`

Whether to use compression of the slave/master protocol if both the slave and the master support it.

- `slave_load_tmpdir`

Command-Line Format	`--slave-load-tmpdir=dir_name`	
System Variable	Name	`slave_load_tmpdir`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`directory name`
	Default	`/tmp`

The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements.

- `slave_net_timeout`

Command-Line Format	`--slave-net-timeout=#`	
System Variable	**Name**	`slave_net_timeout`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`integer`
	Default	`3600`
	Min Value	`1`

The number of seconds to wait for more data from a master/slave connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made using Unix socket files, named pipes, or shared memory.

- `slave_skip_errors`

Command-Line Format	`--slave-skip-errors=name`	
System Variable	**Name**	`slave_skip_errors`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`string`
	Default	`OFF`
	Valid Values	`OFF`
		`[list of error codes]`
		`all`

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_transaction_retries`

Introduced	5.0.3

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command-Line Format	--slave_transaction_retries=#	
System Variable	Name	slave_transaction_retries
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	10
	Min Value	0
	Max Value	4294967295
Permitted Values (64-bit platforms)	Type	integer
	Default	10
	Min Value	0
	Max Value	18446744073709547520

If a replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because the transaction's execution time exceeded InnoDB's innodb_lock_wait_timeout or NDBCLUSTER's TransactionDeadlockDetectionTimeout or TransactionInactiveTimeout, it automatically retries slave_transaction_retries times before stopping with an error. Prior to MySQL 5.0.3, the default is 0, and you must explicitly set the value greater than 0 to enable the “retry” behavior. In MySQL 5.0.3 or newer, the default is 10.

- sql_slave_skip_counter

System Variable	Name	sql_slave_skip_counter
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer

The number of events from the master that a slave server should skip.

Important

If skipping the number of events specified by setting this variable would cause the slave to begin in the middle of an event group, the slave continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see Section 13.4.2.6, “SET GLOBAL sql_slave_skip_counter Syntax”.

16.1.2.4 Binary Log Options and Variables

Startup Options Used with Binary Logging

This documentation is for an older version. If you're
This documentation is for an older version. If you're

System Variables Used with Binary Logging

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see Section 5.4.3, “The Binary Log”. For additional information about using MySQL server options and system variables, see Section 5.1.3, “Server Command Options”, and Section 5.1.4, “Server System Variables”.

Startup Options Used with Binary Logging

The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--log-bin[=base_name]`

Command-Line Format	`--log-bin`	
System Variable	**Name**	`log_bin`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	**Type**	`file name`

Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See Section 5.4.3, “The Binary Log”.

The option value, if given, is the base name for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the base name. It is recommended that you specify a base name (see Section B.5.7, “Known Issues in MySQL”, for the reason). Otherwise, MySQL uses *`host_name`*`-bin` as the base name.

Setting this option causes the `log_bin` system variable to be set to `ON` (or `1`), and not to the base name. This is a known issue; see Bug #19614 for more information.

- `--log-bin-index[=file_name]`

Command-Line Format	`--log-bin-index=file_name`	
Permitted Values	**Type**	`file name`

The index file for binary log file names. See Section 5.4.3, “The Binary Log”. If you omit the file name, and if you did not specify one with `--log-bin`, MySQL uses *`host_name`*`-bin.index` as the file name.

- `--log-bin-trust-function-creators[={0|1}]`

Introduced	5.0.16	
Command-Line Format	`--log-bin-trust-function-creators`	
System Variable	**Name**	`log_bin_trust_function_creators`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	`boolean`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	FALSE

This option sets the corresponding log_bin_trust_function_creators system variable. If no argument is given, the option sets the variable to 1. log_bin_trust_function_creators affects how MySQL enforces restrictions on stored function and trigger creation. See Section 18.6, "Binary Logging of Stored Programs".

This option was added in MySQL 5.0.16.

- --log-bin-trust-routine-creators[={0|1}]

Introduced	5.0.6	
Deprecated	5.0.16	
Command-Line Format	--log-bin-trust-routine-creators	
System Variable	**Name**	log_bin_trust_routine_creators
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	FALSE

This is the old name for --log-bin-trust-function-creators. Before MySQL 5.0.16, it also applies to stored procedures, not just stored functions and sets the log_bin_trust_routine_creators system variable. As of 5.0.16, this option is deprecated. It is recognized for backward compatibility but its use results in a warning.

This option was added in MySQL 5.0.6.

Statement selection options. The options in the following list affect which statements are written to the binary log, and thus sent by a replication master server to its slaves. There are also options for slave servers that control which statements received from the master should be executed or ignored. For details, see Section 16.1.2.3, "Replication Slave Options and Variables".

- --binlog-do-db=*db_name*

Command-Line Format	--binlog-do-db=name	
Permitted Values	**Type**	string

This option affects binary logging in a manner similar to the way that --replicate-do-db affects replication.

Tell the server to restrict binary logging to updates for which the default database is *db_name* (that is, the database selected by USE). All other databases that are not explicitly mentioned are ignored. If you use this option, you should ensure that you do updates only in the default database.

There is an exception to this for CREATE DATABASE, ALTER DATABASE, and DROP DATABASE statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with --binlog-do-db=sales and you issue the following statements, the UPDATE statement is *not* logged:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

Important

To log multiple databases, use this option multiple times, specifying the option once for each database to be logged. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

- `--binlog-ignore-db=`*`db_name`*

Command-Line Format	`--binlog-ignore-db=name`	
Permitted Values	**Type**	`string`

This option affects binary logging in a manner similar to the way that `--replicate-ignore-db` affects replication.

Tell the server to suppress binary logging of updates for which the default database is *`db_name`* (that is, the database selected by `USE`). If you use this option, you should ensure that you do updates only in the default database.

As with the `--binlog-do-db` option, there is an exception for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `binlog-ignore-db=sales`, and you run `USE prices; UPDATE sales.january SET amount = amount + 1000;`, this statement *is* written into the binary log.

Important

To ignore multiple databases, use this option multiple times, specifying the option once for each database to be ignored. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

Testing and debugging options. The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--max-binlog-dump-events=N`

Command-Line Format	`--max-binlog-dump-events=#`	
Permitted Values	Type	`integer`
	Default	`0`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--sporadic-binlog-dump-fail`

Command-Line Format	`--sporadic-binlog-dump-fail`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

This option is used internally by the MySQL test suite for replication testing and debugging.

System Variables Used with Binary Logging

The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used to control binary logging are listed earlier in this section. For information about the `sql_log_bin` and `sql_log_off` variables, see Section 5.1.4, “Server System Variables”.

- `log_bin`

System Variable	Name	`log_bin`
	Variable Scope	Global
	Dynamic Variable	No

Whether the binary log is enabled. If the `--log-bin` option is used, then the value of this variable is `ON`; otherwise it is `OFF`. This variable reports only on the status of binary logging (enabled or disabled); it does not actually report the value to which `--log-bin` is set.

See Section 5.4.3, “The Binary Log”.

- `log_slave_updates`

Command-Line Format	`--log-slave-updates`	
System Variable	Name	`log_slave_updates`
	Variable Scope	Global
	Dynamic Variable	No
Permitted Values	Type	`boolean`
	Default	`FALSE`

Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. See Section 16.1.2.3, “Replication Slave Options and Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `max_binlog_cache_size`

Command-Line Format	`--max_binlog_cache_size=#`	
System Variable	Name	`max_binlog_cache_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	`integer`
	Default	`4294967295`
	Min Value	`4096`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	`integer`
	Default	`18446744073709547520`
	Min Value	`4096`
	Max Value	`18446744073709547520`

If a multiple-statement transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096. The maximum and default values are 4GB on 32-bit platforms and 16EB (exabytes) on 64-bit platforms. The maximum recommended value on 64-bit platforms is 4GB; this is due to the fact that MySQL currently cannot work with binary log positions greater than 4GB.

In MySQL 5.0, a change in `max_binlog_cache_size` takes immediate effect for all active sessions.

- `max_binlog_size`

Command-Line Format	`--max_binlog_size=#`	
System Variable	Name	`max_binlog_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	`integer`
	Default	`1073741824`
	Min Value	`4096`
	Max Value	`1073741824`

This documentation is for an older version. If you're

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than `max_binlog_size`.

If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well.

- `sync_binlog`

Introduced	5.0.1	
Command-Line Format	`--sync-binlog=#`	
System Variable	**Name**	`sync_binlog`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	`integer`
	Default	`0`
	Min Value	`0`
	Max Value	`18446744073709547520`

If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using `fdatasync()`) after every `sync_binlog` writes to the binary log. There is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of `sync_binlog` is 0, which does no synchronizing to disk. A value of 1 is the safest choice because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

If the value of `sync_binlog` is 0 (the default), no extra flushing is done. The server relies on the operating system to flush the file contents occasionally as for any other file.

16.1.3 Common Replication Administration Tasks

Once replication has been started it should execute without requiring much regular administration. Depending on your replication environment, you will want to check the replication status of each slave periodically, daily, or even more frequently.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

16.1.3.1 Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the master. The primary statement for this is `SHOW SLAVE STATUS`, which you must execute on each slave:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: master1
                Master_User: root
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000004
        Read_Master_Log_Pos: 931
             Relay_Log_File: slave1-relay-bin.000056
              Relay_Log_Pos: 950
      Relay_Master_Log_File: mysql-bin.000004
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB:
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 931
            Relay_Log_Space: 1365
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
```

The key fields from the status report to examine are:

- `Slave_IO_State`: The current status of the slave. See Section 8.14.6, “Replication Slave I/O Thread States”, and Section 8.14.7, “Replication Slave SQL Thread States”, for more information.
- `Slave_IO_Running`: Whether the I/O thread for reading the master's binary log is running. Normally, you want this to be `Yes` unless you have not yet started replication or have explicitly stopped it with `STOP SLAVE`.
- `Slave_SQL_Running`: Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be `Yes`.
- `Last_Error`: The last error registered when processing the relay log. Ideally this should be blank, indicating no error.
- `Seconds_Behind_Master`: The number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to handle events from the master in a timely fashion.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A value of 0 for `Seconds_Behind_Master` can usually be interpreted as meaning that the slave has caught up with the master, but there are some cases where this is not strictly true. For example, this can occur if the network connection between master and slave is broken but the slave I/O thread has not yet noticed this—that is, `slave_net_timeout` has not yet elapsed.

It is also possible that transient values for `Seconds_Behind_Master` may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, `Seconds_Behind_Master` displays 0; but when the slave I/O thread is still queuing up a new event, `Seconds_Behind_Master` may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have old timestamps; in such cases, if you execute `SHOW SLAVE STATUS` several times in a relatively short period, you may see this value change back and forth repeatedly between 0 and a relatively large value.

Several pairs of fields provide information about the progress of the slave in reading events from the master binary log and processing them in the relay log:

- (`Master_Log_file`, `Read_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave I/O thread has read events from that log.
- (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave SQL thread has executed events received from that log.
- (`Relay_Log_File`, `Relay_Log_Pos`): Coordinates in the slave relay log indicating how far the slave SQL thread has executed the relay log. These correspond to the preceding coordinates, but are expressed in slave relay log coordinates rather than master binary log coordinates.

On the master, you can check the status of connected slaves using `SHOW PROCESSLIST` to examine the list of running processes. Slave connections have `Binlog Dump` in the `Command` field:

```
mysql> SHOW PROCESSLIST \G;
*************************** 4. row ***************************
     Id: 10
   User: root
   Host: slave1:58371
     db: NULL
Command: Binlog Dump
   Time: 777
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

Because it is the slave that drives the replication process, very little information is available in this report.

For slaves that were started with the `--report-host` option and are connected to the master, the `SHOW SLAVE HOSTS` statement on the master shows basic information about the slaves. The output includes the ID of the slave server, the value of the `--report-host` option, the connecting port, and master ID:

```
mysql> SHOW SLAVE HOSTS;
+-----------+--------+------+------------------+-----------+
| Server_id | Host   | Port | Rpl_recovery_rank | Master_id |
+-----------+--------+------+------------------+-----------+
|        10 | slave1 | 3306 |                0 |         1 |
+-----------+--------+------+------------------+-----------+
1 row in set (0.00 sec)
```

16.1.3.2 Pausing Replication on the Slave

You can stop and start the replication of statements on the slave using the `STOP SLAVE` and `START SLAVE` statements.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To stop processing of the binary log from the master, use `STOP SLAVE`:

```
mysql> STOP SLAVE;
```

When replication is stopped, the slave I/O thread stops reading events from the master binary log and writing them to the relay log, and the SQL thread stops reading events from the relay log and executing them. You can pause the I/O or SQL thread individually by specifying the thread type:

```
mysql> STOP SLAVE IO_THREAD;
mysql> STOP SLAVE SQL_THREAD;
```

To start execution again, use the `START SLAVE` statement:

```
mysql> START SLAVE;
```

To start a particular thread, specify the thread type:

```
mysql> START SLAVE IO_THREAD;
mysql> START SLAVE SQL_THREAD;
```

For a slave that performs updates only by processing events from the master, stopping only the SQL thread can be useful if you want to perform a backup or other task. The I/O thread will continue to read events from the master but they are not executed. This makes it easier for the slave to catch up when you restart the SQL thread.

Stopping only the I/O thread enables the events in the relay log to be executed by the SQL thread up to the point where the relay log ends. This can be useful when you want to pause execution to catch up with events already received from the master, when you want to perform administration on the slave but also ensure that it has processed all updates to a specific point. This method can also be used to pause event receipt on the slave while you conduct administration on the master. Stopping the I/O thread but permitting the SQL thread to run helps ensure that there is not a massive backlog of events to be executed when replication is started again.

16.2 Replication Implementation

Replication is based on the master server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify database structure or content (data) from the moment the server was started. Typically, `SELECT` statements are not recorded because they modify neither database structure nor content.

Each slave that connects to the master requests a copy of the binary log. That is, it pulls the data from the master, rather than the master pushing the data to the slave. The slave also executes the events from the binary log that it receives. This has the effect of repeating the original changes just as they were made on the master. Tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the master.

Because each slave is independent, the replaying of the changes from the master's binary log occurs independently on each slave that is connected to the master. In addition, because each slave receives a copy of the binary log only by requesting it from the master, the slave is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the master or slave side.

For more information on the specifics of the replication implementation, see Section 16.2.1, “Replication Implementation Details”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Masters and slaves report their status in respect of the replication process regularly so that you can monitor them. See Section 8.14, “Examining Thread Information”, for descriptions of all replicated-related states.

The master binary log is written to a local relay log on the slave before it is processed. The slave also records information about the current position with the master's binary log and the local relay log. See Section 16.2.2, “Replication Relay and Status Logs”.

Database changes are filtered on the slave according to a set of rules that are applied according to the various configuration options and variables that control event evaluation. For details on how these rules are applied, see Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

16.2.1 Replication Implementation Details

MySQL replication capabilities are implemented using three threads, one on the master server and two on the slave:

- **Binlog dump thread.** The master creates a thread to send the binary log contents to a slave when the slave connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the master as the `Binlog Dump` thread.

  The binary log dump thread acquires a lock on the master's binary log for reading each event that is to be sent to the slave. As soon as the event has been read, the lock is released, even before the event is sent to the slave.

- **Slave I/O thread.** When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs.

  The slave I/O thread reads the updates that the master's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the slave's relay log.

  The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS` or as `Slave_running` in the output of `SHOW STATUS`.

- **Slave SQL thread.** The slave creates an SQL thread to read the relay log that is written by the slave I/O thread and execute the events contained therein.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one binary log dump thread for each currently connected slave, and each slave has its own I/O and SQL threads.

A slave uses two threads to separate reading updates from the master and executing them into independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. For information on master states, see Section 8.14.5, “Replication Master Thread States”. For slave states, see Section 8.14.6, “Replication Slave I/O Thread States”, and Section 8.14.7, “Replication Slave SQL Thread States”.

The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Here, thread 2 is a `Binlog Dump` replication thread that services a connected slave. The `State` information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, no slaves are currently connected.

On a slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

The `State` information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See Section A.13, “MySQL 5.0 FAQ: Replication”. If sufficient time elapses on the master side without activity on the `Binlog Dump` thread, the master determines that the slave is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see Section 5.1.4, “Server System Variables”.

The `SHOW SLAVE STATUS` statement provides additional information about replication processing on a slave server. See Section 16.1.3.1, “Checking Replication Status”.

16.2.2 Replication Relay and Status Logs

During replication, a slave server creates several logs that hold the binary log events relayed from the master to the slave, and to record information about the current status and location within the relay log. There are three types of logs used in the process, listed here:

- The *relay log* consists of the events read from the binary log of the master and written by the slave I/O thread. Events in the relay log are executed on the slave as part of the SQL thread.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The *master info log* contains status and current configuration information for the slave's connection to the master. This log holds information on the master host name, login credentials, and coordinates indicating how far the slave has read from the master's binary log.
- The *relay log info log* holds status information about the execution point within the slave's relay log.

16.2.2.1 The Slave Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog` (see Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”).

By default, relay log file names have the form *`host_name`*`-relay-bin.`*`nnnnnn`* in the data directory, where *`host_name`* is the name of the slave server host and *`nnnnnn`* is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001`. The slave uses an index file to track the relay log files currently in use. The default relay log index file name is *`host_name`*`-relay-bin.index` in the data directory.

The default relay log file and relay log index file names can be overridden with, respectively, the `--relay-log` and `--relay-log-index` server options (see Section 16.1.2, “Replication and Binary Logging Options and Variables”).

If a slave uses the default host-based relay log file names, changing a slave's host name after replication has been set up can cause replication to fail with the errors `Failed to open the relay log` and `Could not find target log during relay log initialization`. This is a known issue (see Bug #2122). If you anticipate that a slave's host name might change in the future (for example, if networking is set up on the slave such that its host name can be modified using DHCP), you can avoid this issue entirely by using the `--relay-log` and `--relay-log-index` options to specify relay log file names explicitly when you initially set up the slave. This will make the names independent of server host name changes.

If you encounter the issue after replication has already begun, one way to work around it is to stop the slave server, prepend the contents of the old relay log index file to the new one, and then restart the slave. On a Unix system, this can be done as shown here:

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
shell> mv old_relay_log_name.index new_relay_log_name.index
```

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with `FLUSH LOGS` or `mysqladmin flush-logs`.
- When the size of the current relay log file becomes “too large,” determined as follows:
  - If the value of `max_relay_log_size` is greater than 0, that is the maximum relay log file size.
  - If the value of `max_relay_log_size` is 0, `max_binlog_size` determines the maximum relay log file size.

The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread

This documentation is for an older version. If you're
This documentation is for an older version. If you're

takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

16.2.2.2 Slave Status Logs

A replication slave server creates two logs. By default, these logs are files named `master.info` and `relay-log.info` and created in the data directory. The names and locations of these files can be changed by using the `--master-info-file` and `--relay-log-info-file` options, respectively. See Section 16.1.2, “Replication and Binary Logging Options and Variables”.

The two status logs contain information like that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in Section 13.4.2, “SQL Statements for Controlling Slave Servers”. Because the status logs are stored on disk, they survive a slave server's shutdown. The next time the slave starts up, it reads the two logs to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The master info log should be protected because it contains the password for connecting to the master. See Section 6.1.2.3, “Passwords and Logging”.

The slave I/O thread updates the master info log. The following table shows the correspondence between the lines in the `master.info` file and the columns displayed by `SHOW SLAVE STATUS`.

Line in `master.info`	`SHOW SLAVE STATUS` Column	Description
1		Number of lines in the file
2	`Master_Log_File`	The name of the master binary log currently being read from the master
3	`Read_Master_Log_Pos`	The current position within the master binary log that have been read from the master
4	`Master_Host`	The host name of the master
5	`Master_User`	The user name used to connect to the master
6	Password (not shown by `SHOW SLAVE STATUS`)	The password used to connect to the master
7	`Master_Port`	The network port used to connect to the master
8	`Connect_Retry`	The period (in seconds) that the slave will wait before trying to reconnect to the master
9	`Master_SSL_Allowed`	Indicates whether the server supports SSL connections
10	`Master_SSL_CA_File`	The file used for the Certificate Authority (CA) certificate
11	`Master_SSL_CA_Path`	The path to the Certificate Authority (CA) certificates
12	`Master_SSL_Cert`	The name of the SSL certificate file
13	`Master_SSL_Cipher`	The list of possible ciphers used in the handshake for the SSL connection
14	`Master_SSL_Key`	The name of the SSL key file

The slave SQL thread updates the relay log info log. The following table shows the correspondence between the lines in the `relay-log.info` file and the columns displayed by `SHOW SLAVE STATUS`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Line in `relay-log.info`	`SHOW SLAVE STATUS` Column	Description
1	`Relay_Log_File`	The name of the current relay log file
2	`Relay_Log_Pos`	The current position within the relay log file; events up to this position have been executed on the slave database
3	`Relay_Master_Log_File`	The name of the master binary log file from which the events in the relay log file were read
4	`Exec_Master_Log_Pos`	The equivalent position within the master's binary log file of events that have already been executed

The contents of the `relay-log.info` file and the states shown by the `SHOW SLAVE STATUS` statement might not match if the `relay-log.info` file has not been flushed to disk. Ideally, you should only view `relay-log.info` on a slave that is offline (that is, `mysqld` is not running). For a running system, `SHOW SLAVE STATUS` should be used.

When you back up the slave's data, you should back up these two status logs, along with the relay log files. The status logs are needed to resume replication after you restore the data from the slave. If you lose the relay logs but still have the relay log info log, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master.

16.2.3 How Servers Evaluate Replication Filtering Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master, you can control which databases to log changes for by using the `--binlog-do-db` and `--binlog-ignore-db` options to control binary logging. For a description of the rules that servers use in evaluating these options, see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”. You should not use these options to control which databases and tables are replicated. Instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the `--replicate-*` options that the slave was started with. (See Section 16.1.2, “Replication and Binary Logging Options and Variables”.)

In the simplest case, when there are no `--replicate-*` options, the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given.

Database-level options (`--replicate-do-db`, `--replicate-ignore-db`) are checked first; see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”, for a description of this process. If no database-level options are used, option checking proceeds to any table-level options that may be in use (see Section 16.2.3.2, “Evaluation of Table-Level Replication Options”, for a discussion of these). If one or more database-level options are used but none are matched, the statement is not replicated.

To make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and nonwildcard options. An example of the latter that may have unintended effects is the use of `--replicate-do-db` and `--replicate-wild-do-table` together,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

where `--replicate-wild-do-table` uses a pattern for the database name that matches the name given for `--replicate-do-db`. Suppose a replication slave is started with `--replicate-do-db=dbx --replicate-wild-do-table=db%.t1`. Then, suppose that on the master, you issue the statement `CREATE DATABASE dbx`. Although you might expect it, this statement is not replicated because it does not reference a table named `t1`.

If any `--replicate-rewrite-db` options were specified, they are applied before the `--replicate-*` filtering rules are tested.

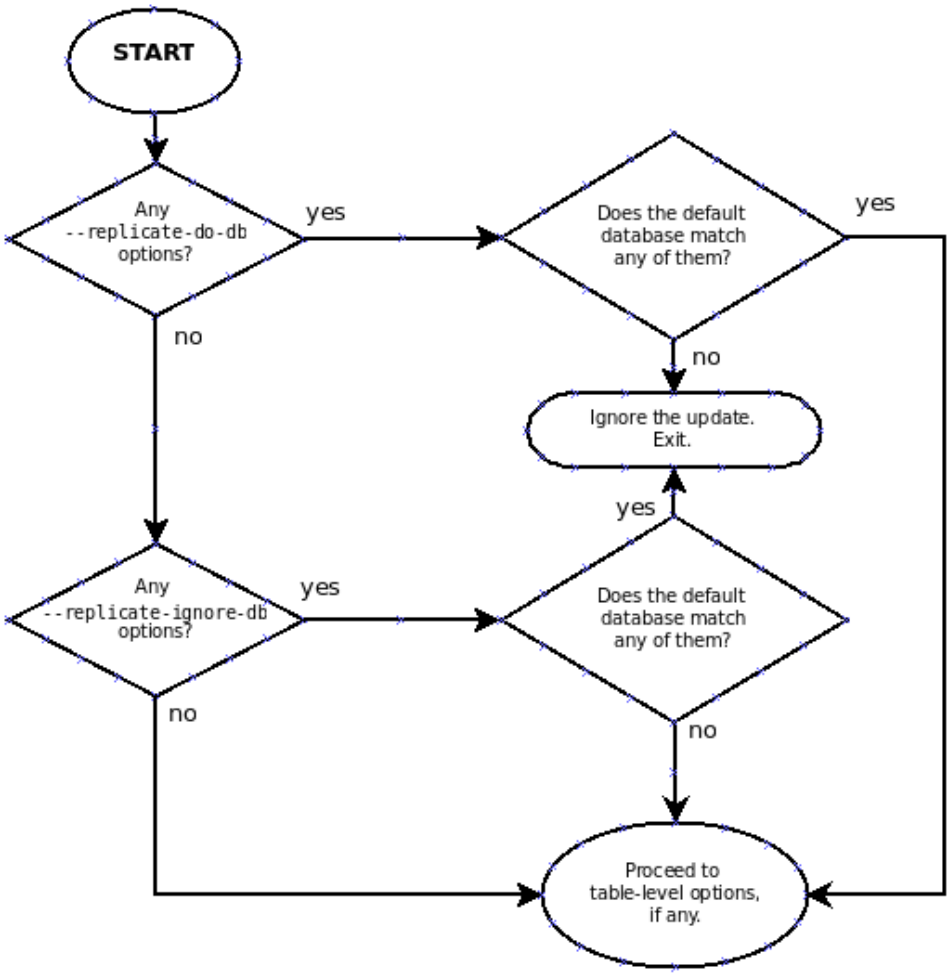
Note

Database-level filtering options are case-sensitive on platforms supporting case sensitivity in filenames, whereas table-level filtering options are not (regardless of platform). This is true regardless of the value of the `lower_case_table_names` system variable.

16.2.3.1 Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the slave begins by checking to see whether there are any `--replicate-do-db` or `--replicate-ignore-db` options that apply. When using `--binlog-do-db` or `--binlog-ignore-db`, the process is similar, but the options are checked on the master.

Checking of the database-level options proceeds as shown in the following diagram.



This documentation is for an older version. If you're

The steps involved are listed here:

1. Are there any `--replicate-do-db` options?
   - **Yes.** Do any of them match the database?
     - **Yes.** Execute the statement and exit.
     - **No.** Ignore the statement and exit.
   - **No.** Continue to step 2.
2. Are there any `--replicate-ignore-db` options?
   - **Yes.** Do any of them match the database?
     - **Yes.** Ignore the statement and exit.
     - **No.** Continue to step 3.
   - **No.** Continue to step 3.
3. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see Section 16.2.3.2, “Evaluation of Table-Level Replication Options”.

Important

A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db` or `--binlog-ignore-db` options?
   - **Yes.** Continue to step 2.
   - **No.** Log the statement and exit.
2. Is there a default database (has any database been selected by `USE`)?
   - **Yes.** Continue to step 3.
   - **No.** Ignore the statement and exit.
3. There is a default database. Are there any `--binlog-do-db` options?
   - **Yes.** Do any of them match the database?
     - **Yes.** Log the statement and exit.
     - **No.** Ignore the statement and exit.
   - **No.** Continue to step 4.
4. Do any of the `--binlog-ignore-db` options match the database?
   - **Yes.** Ignore the statement and exit.
   - **No.** Log the statement and exit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Important

An exception is made in the rules just given for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In those cases, the database being *created, altered, or dropped* replaces the default database when determining whether to log or to ignore updates.

`--binlog-do-db` can sometimes mean “ignore other databases”. For example, a server running with only `--binlog-do-db=sales` does not write to the binary log statements for which the default database differs from `sales`.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog`.

16.2.3.2 Evaluation of Table-Level Replication Options

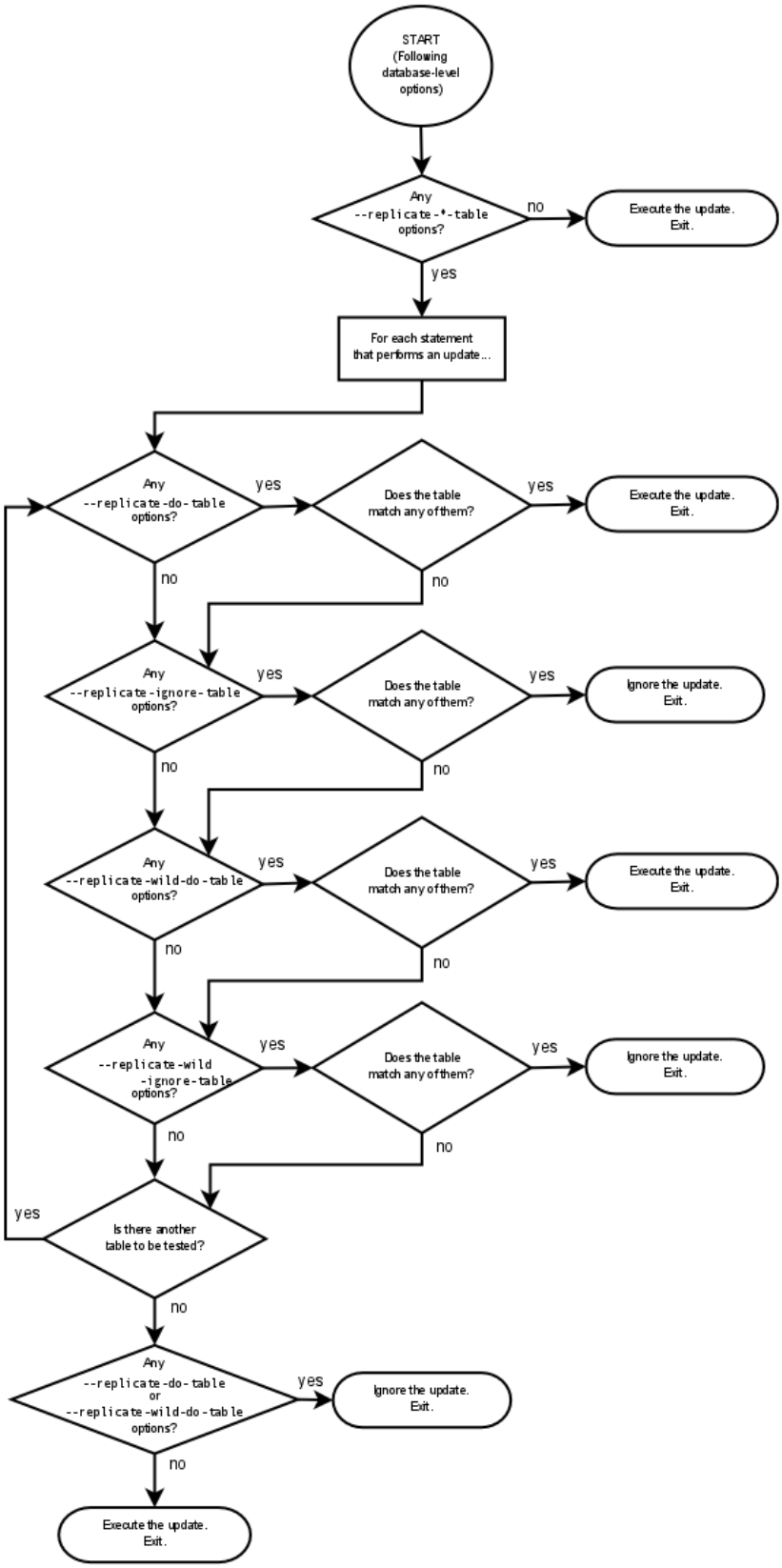
The slave checks for and evaluates table options only if either of the following two conditions is true:

- No matching database options were found.
- One or more database options were found, and were evaluated to arrive at an “execute” condition according to the rules described in the previous section (see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”).

First, as a preliminary condition, the slave checks whether the statement occurs within a stored function, in which case the slave executes the statement and exits.

Having reached this point, if there are no table options, the slave simply executes all statements. If there are any `--replicate-do-table` or `--replicate-wild-do-table` options, the statement must match one of these if it is to be executed; otherwise, it is ignored. If there are any `--replicate-ignore-table` or `--replicate-wild-ignore-table` options, all statements are executed except those that match any of these options. This process is illustrated in the following diagram.

This documentation is for an older version. If you're
This documentation is for an older version. If you're



This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `master.info` file should be protected because it contains the password for connecting to the master. See Section 6.1.2.3, “Passwords and Logging”.

The following steps describe this evaluation in more detail:

1. Are there any table options?
   - **Yes.** Continue to step 2.
   - **No.** Execute the statement and exit.
2. Are there any `--replicate-do-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Execute the statement and exit.
     - **No.** Continue to step 3.
   - **No.** Continue to step 3.
3. Are there any `--replicate-ignore-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Ignore the statement and exit.
     - **No.** Continue to step 4.
   - **No.** Continue to step 4.
4. Are there any `--replicate-wild-do-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Execute the statement and exit.
     - **No.** Continue to step 5.
   - **No.** Continue to step 5.
5. Are there any `--replicate-wild-ignore-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Ignore the statement and exit.
     - **No.** Continue to step 6.
   - **No.** Continue to step 6.
6. Are there any `--replicate-do-table` or `--replicate-wild-do-table` options?
   - **Yes.** Ignore the statement and exit.
   - **No.** Execute the statement and exit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

16.2.3.3 Replication Rule Application

This section provides additional explanation and examples of usage for different combinations of replication filtering options.

Some typical combinations of replication filter rule types are given in the following table:

Condition (Types of Options)	Outcome
No `--replicate-*` options at all:	The slave executes all events that it receives from the master.
`--replicate-*-db` options, but no table options:	The slave accepts or ignores statements using the database options. It executes all statements permitted by those options because there are no table restrictions.
`--replicate-*-table` options, but no database options:	All statements are accepted at the database-checking stage because there are no database conditions. The slave executes or ignores statements based solely on the table options.
A combination of database and table options:	The slave accepts or ignores statements using the database options. Then it evaluates all statements permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive; see the text for an example.

A more complex example follows.

Suppose that we have two tables `mytbl1` in database `db1` and `mytbl2` in database `db2` on the master, and the slave is running with the following options (and no other replication filtering options):

```
replicate-ignore-db = db1
replicate-do-table  = db2.tbl2
```

Now we execute the following statements on the master:

```
USE db1;
INSERT INTO db2.tbl2 VALUES (1);
```

The outcome may not match initial expectations, because the `USE` statement causes `db1` to be the default database. Thus the `--replicate-ignore-db` option matches, which causes the `INSERT` statement to be ignored. Because there was a match with a database-level option, the table options are not checked; processing immediately moves to the next statement executed on the master.

16.3 Replication Solutions

Replication can be used in many different environments for a range of purposes. This section provides general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to back up, see Section 16.3.1, “Using Replication for Backups”.

For advice and tips on using different storage engines on the master and slaves, see Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”.

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See Section 16.3.3, “Using Replication for Scale-Out”.

For performance or data distribution reasons, you may want to replicate different databases to different replication slaves. See Section 16.3.4, “Replicating Different Databases to Different Slaves”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As the number of replication slaves increases, the load on the master can increase and lead to reduced performance (because of the need to replicate the binary log to each slave). For tips on improving your replication performance, including using a single secondary server as a replication master, see Section 16.3.5, “Improving Replication Performance”.

For guidance on switching masters, or converting slaves into masters as part of an emergency failover solution, see Section 16.3.6, “Switching Masters During Failover”.

To secure your replication communication, you can encrypt the communication channel. For step-by-step instructions, see Section 16.3.7, “Setting Up Replication to Use Secure Connections”.

16.3.1 Using Replication for Backups

To use replication as a backup solution, replicate data from the master to a slave, and then back up the data slave. The slave can be paused and shut down without affecting the running operation of the master, so you can produce an effective snapshot of “live” data that would otherwise require the master to be shut down.

How you back up a database depends on its size and whether you are backing up only the data, or the data and the replication slave state so that you can rebuild the slave in the event of failure. There are therefore two choices:

- If you are using replication as a solution to enable you to back up the data on the master, and the size of your database is not too large, the `mysqldump` tool may be suitable. See Section 16.3.1.1, “Backing Up a Slave Using mysqldump”.
- For larger databases, where `mysqldump` would be impractical or inefficient, you can back up the raw data files instead. Using the raw data files option also means that you can back up the binary and relay logs that will enable you to recreate the slave in the event of a slave failure. For more information, see Section 16.3.1.2, “Backing Up Raw Data from a Slave”.

16.3.1.1 Backing Up a Slave Using mysqldump

Using `mysqldump` to create a copy of a database enables you to capture all of the data in the database in a format that enables the information to be imported into another instance of MySQL Server (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). Because the format of the information is SQL statements, the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large, `mysqldump` may be impractical.

When using `mysqldump`, you should stop replication on the slave before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the slave from processing requests. You can stop replication completely on the slave using `mysqladmin`:

   ```
   shell> mysqladmin stop-slave
   ```

   Alternatively, you can stop only the slave SQL thread to pause event execution:

   ```
   shell> mysql -e 'STOP SLAVE SQL_THREAD;'
   ```

   This enables the slave to continue to receive data change events from the master's binary log and store them in the relay logs using the I/O thread, but prevents the slave from executing these events and changing its data. Within busy replication environments, permitting the I/O thread to run during backup may speed up the catch-up process when you restart the slave SQL thread.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. Run `mysqldump` to dump your databases. You may either dump all databases or select databases to be dumped. For example, to dump all databases:

```
shell> mysqldump --all-databases > fulldb.dump
```

3. Once the dump has completed, start slave operations again:

```
shell> mysqladmin start-slave
```

In the preceding example, you may want to add login credentials (user name, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the slave replication process to ensure that the time taken to run the backup does not affect the slave's ability to keep up with events from the master. See Section 16.1.3.1, “Checking Replication Status”. If the slave is unable to keep up, you may want to add another slave and distribute the backup process. For an example of how to configure this scenario, see Section 16.3.4, “Replicating Different Databases to Different Slaves”.

16.3.1.2 Backing Up Raw Data from a Slave

To guarantee the integrity of the files that are copied, backing up the raw data files on your MySQL replication slave should take place while your slave server is shut down. If the MySQL server is still running, background tasks may still be updating the database files, particularly those involving storage engines with background processes such as `InnoDB`. With `InnoDB`, these problems should be resolved during crash recovery, but since the slave server can be shut down during the backup process without affecting the execution of the master it makes sense to take advantage of this capability.

To shut down the server and back up the files:

1. Shut down the slave MySQL server:

```
shell> mysqladmin shutdown
```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`. For example, assuming that the data directory is located under the current directory, you can archive the entire directory as follows:

```
shell> tar cf /tmp/dbbackup.tar ./data
```

3. Start the MySQL server again. Under Unix:

```
shell> mysqld_safe &
```

Under Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"
```

Normally you should back up the entire data directory for the slave MySQL server. If you want to be able to restore the data and operate as a slave (for example, in the event of failure of the slave), then in addition to the slave's data, you should also back up the slave status files, `master.info` and `relay-log.info`, along with the relay log files. These files are needed to resume replication after you restore the slave's data.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume

This documentation is for an older version. If you're
This documentation is for an older version. If you're

replication of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` option. If the server was not started with that option, the directory location is the value of the `tmpdir` system variable.

16.3.2 Using Replication with Different Master and Slave Storage Engines

It does not matter for the replication process whether the source table on the master and the replicated table on the slave use different engine types. In fact, the system variables `storage_engine` and `table_type` are not replicated.

This provides a number of benefits in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scale-out scenario (see Section 16.3.3, “Using Replication for Scale-Out”), you want to use `InnoDB` tables on the master to take advantage of the transactional functionality, but use `MyISAM` on the slaves where transaction support is not required because the data is only read. When using replication in a data-logging environment you may want to use the `Archive` storage engine on the slave.

Configuring different engines on the master and slave depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your master, you could edit the dump file text to change the engine type used on each table.

  Another alternative for `mysqldump` is to disable engine types that you do not want to use on the slave before using the dump to build the data on the slave. For example, you can add the `--skip-innodb` option on your slave to disable the `InnoDB` engine. If a specific engine does not exist for a table to be created, MySQL will use the default engine type, usually `MyISAM`. (This requires that the `NO_ENGINE_SUBSTITUTION` SQL mode is not enabled.) If you want to disable additional engines in this way, you may want to consider building a special binary to be used on the slave that only supports the engines you want.

- If you are using raw data files (a binary backup) to set up the slave, you will be unable to change the initial table format. Instead, use `ALTER TABLE` to change the table types after the slave has been started.

- For new master/slave replication setups where there are currently no tables on the master, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the slave from running replication updates:

   ```
   mysql> STOP SLAVE;
   ```

   This will enable you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... ENGINE='engine_type'` for each table to be changed.

3. Start the slave replication process again:

   ```
   mysql> START SLAVE;
   ```

Although the `storage_engine` and `table_type` variables are not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification will be correctly replicated to the slave. For example, if you have a CSV table and you execute:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The above statement will be replicated to the slave and the engine type on the slave will be converted to `MyISAM`, even if you have previously changed the table type on the slave to an engine other than CSV. If you want to retain engine differences on the master and slave, you should be careful to use the `storage_engine` variable on the master when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET storage_engine=MyISAM;
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the `storage_engine` variable will be ignored, and the `CREATE TABLE` statement will execute on the slave using the slave's default engine.

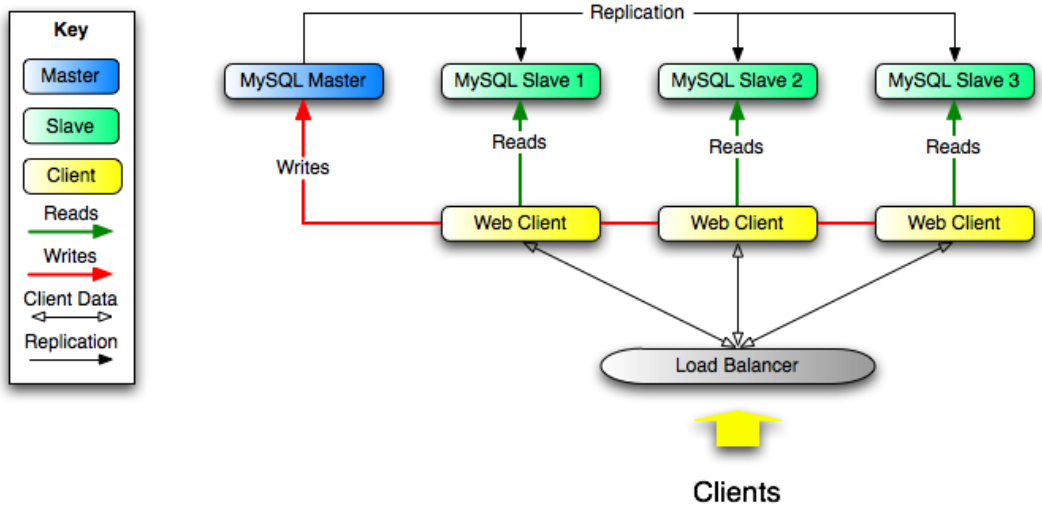
16.3.3 Using Replication for Scale-Out

You can use replication as a scale-out solution; that is, where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one master to one or more slaves, using replication for scale-out works best in an environment where you have a high number of reads and low number of writes/updates. Most Web sites fit into this category, where users are browsing the Web site, reading articles, posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replication slaves, while still enabling your web servers to communicate with the replication master when a write is required. You can see a sample replication layout for this scenario in Figure 16.1, “Using Replication to Improve Performance During Scale-Out”.

Figure 16.1 Using Replication to Improve Performance During Scale-Out



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of

This documentation is for an older version. If you're

your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

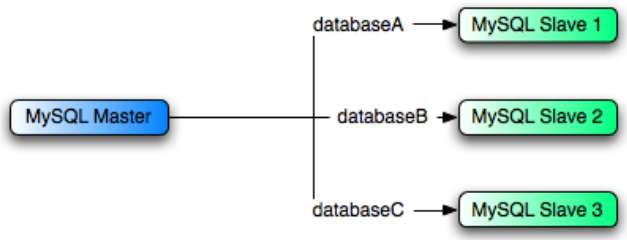
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the `replace` utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

16.3.4 Replicating Different Databases to Different Slaves

There may be situations where you have a single master and want to replicate different databases to different slaves. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in Figure 16.2, “Using Replication to Replicate Databases to Separate Replication Slaves”.

Figure 16.2 Using Replication to Replicate Databases to Separate Replication Slaves



You can achieve this separation by configuring the master and slaves as normal, and then limiting the binary log statements that each slave processes by using the `--replicate-wild-do-table` configuration option on each slave.

Important

You should *not* use `--replicate-do-db` for this purpose, since its affects vary according to the database that is currently selected.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For example, to support the separation as shown in Figure 16.2, “Using Replication to Replicate Databases to Separate Replication Slaves”, you should configure each replication slave as follows, before executing `START SLAVE`:

- Replication slave 1 should use `--replicate-wild-do-table=databaseA.%`.
- Replication slave 2 should use `--replicate-wild-do-table=databaseB.%`.
- Replication slave 3 should use `--replicate-wild-do-table=databaseC.%`.

Each slave in this configuration receives the entire binary log from the master, but executes only those events from the binary log that apply to the databases and tables included by the `--replicate-wild-do-table` option in effect on that slave.

If you have data that must be synchronized to the slaves before replication starts, you have a number of choices:

- Synchronize all the data to each slave, and delete the databases, tables, or both that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each slave.
- Use a raw data file dump and include only the specific files and databases that you need for each slave.

Note

This does not work with `InnoDB` databases unless you use `innodb_file_per_table`.

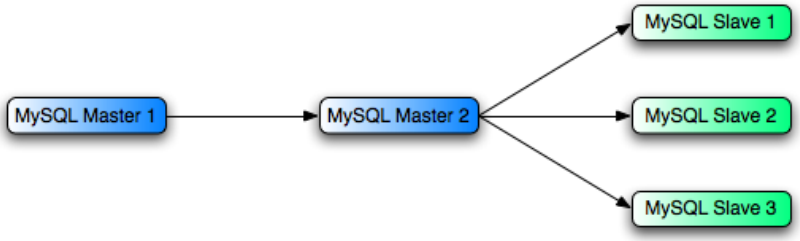
16.3.5 Improving Replication Performance

As the number of slaves connecting to a master increases, the load, although minimal, also increases, as each slave uses a client connection to the master. Also, as each slave must receive a full copy of the master binary log, the network load on the master may also increase and create a bottleneck.

If you are using a large number of slaves connected to one master, and that master is also busy processing requests (for example, as part of a scale-out solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the master to replicate to only one slave, and for the remaining slaves to connect to this primary slave for their individual replication requirements. A sample of this structure is shown in Figure 16.3, “Using an Additional Replication Host to Improve Performance”.

Figure 16.3 Using an Additional Replication Host to Improve Performance



This documentation is for an older version. If you're
This documentation is for an older version. If you're

For this to work, you must configure the MySQL instances as follows:

- Master 1 is the primary master where all changes and updates are written to the database. Binary logging should be enabled on this machine.
- Master 2 is the slave to the Master 1 that provides the replication functionality to the remainder of the slaves in the replication structure. Master 2 is the only machine permitted to connect to Master 1. Master 2 also has binary logging enabled, and the `--log-slave-updates` option so that replication instructions from Master 1 are also written to Master 2's binary log so that they can then be replicated to the true slaves.
- Slave 1, Slave 2, and Slave 3 act as slaves to Master 2, and replicate the information from Master 2, which actually consists of the upgrades logged on Master 1.

The above solution reduces the client load and the network interface load on the primary master, which should improve the overall performance of the primary master when used as a direct database solution.

If your slaves are having trouble keeping up with the replication process on the master, there are a number of options available:

- If possible, put the relay logs and the data files on different physical drives. To do this, use the `--relay-log` option to specify the location of the relay log.
- If the slaves are significantly slower than the master, you may want to divide up the responsibility for replicating different databases to different slaves. See Section 16.3.4, “Replicating Different Databases to Different Slaves”.
- If your master makes use of transactions and you are not concerned about transaction support on your slaves, use `MyISAM` or another nontransactional engine on the slaves. See Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”.
- If your slaves are not acting as masters, and you have a potential solution in place to ensure that you can bring up a master in the event of failure, then you can switch off `--log-slave-updates`. This prevents “dumb” slaves from also logging events they have executed into their own binary log.

16.3.6 Switching Masters During Failover

There is in MySQL 5.0 no official solution for providing failover between master and slaves in the event of a failure. Instead, you must set up a master and one or more slaves; then, you need to write an application or script that monitors the master to check whether it is up, and instructs the slaves and applications to change master in case of failure. This section discusses some of the issues encountered when setting up failover in this fashion.

Note

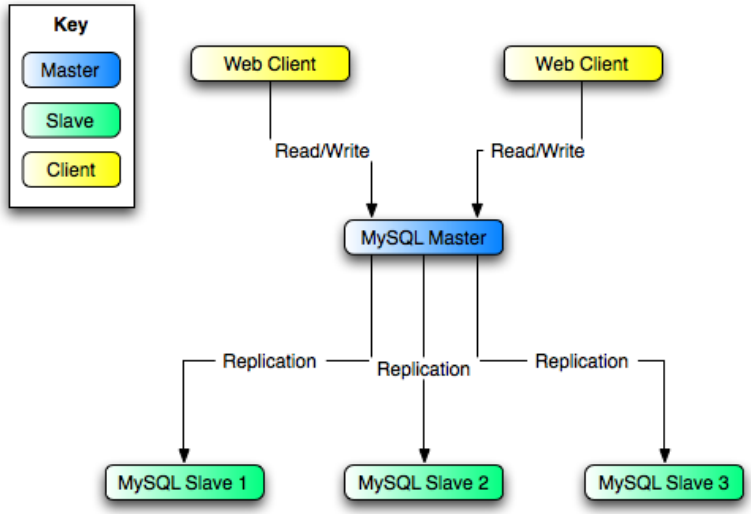
The MySQL Utilities include a `mysqlfailover` tool that provides failover capability using GTIDs, support for which requires MySQL 5.6 or later. For more information, see `mysqlfailover` — Automatic replication health monitoring and failover, and Replication with Global Transaction Identifiers.

You can tell a slave to change to a new master using the `CHANGE MASTER TO` statement. The slave does not check whether the databases on the master are compatible with those on the slave; it simply begins reading and executing events from the specified coordinates in the new master's binary log. In a failover situation, all the servers in the group are typically executing the same events from the same binary log file, so changing the source of the events should not affect the structure or integrity of the database, provided that you exercise care in making the change.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Slaves should be run with the `--log-bin` option and without `--log-slave-updates`. In this way, the slave is ready to become a master without restarting the slave `mysqld`. Assume that you have the structure shown in Figure 16.4, “Redundancy Using Replication, Initial Structure”.

Figure 16.4 Redundancy Using Replication, Initial Structure



In this diagram, the `MySQL Master` holds the master database, the `MySQL Slave` hosts are replication slaves, and the `Web Client` machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the slaves) are not shown, as they do not need to switch to a new server in the event of failure. For a more detailed example of a read/write scale-out replication structure, see Section 16.3.3, “Using Replication for Scale-Out”.

Each MySQL Slave (`Slave 1`, `Slave 2`, and `Slave 3`) is a slave running with `--log-bin` and without `--log-slave-updates`. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` is specified, the binary log on each slave is empty initially. If for some reason `MySQL Master` becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick `Slave 1`, all `Web Clients` should be redirected to `Slave 1`, which writes the updates to its binary log. `Slave 2` and `Slave 3` should then replicate from `Slave 1`.

The reason for running the slave without `--log-slave-updates` is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. If `Slave 1` has `--log-slave-updates` enabled, it writes any updates that it receives from `Master` in its own binary log. This means that, when `Slave 2` changes from `Master` to `Slave 1` as its master, it may receive updates from `Slave 1` that it has already received from `Master`.

Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave `Slave 1` being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

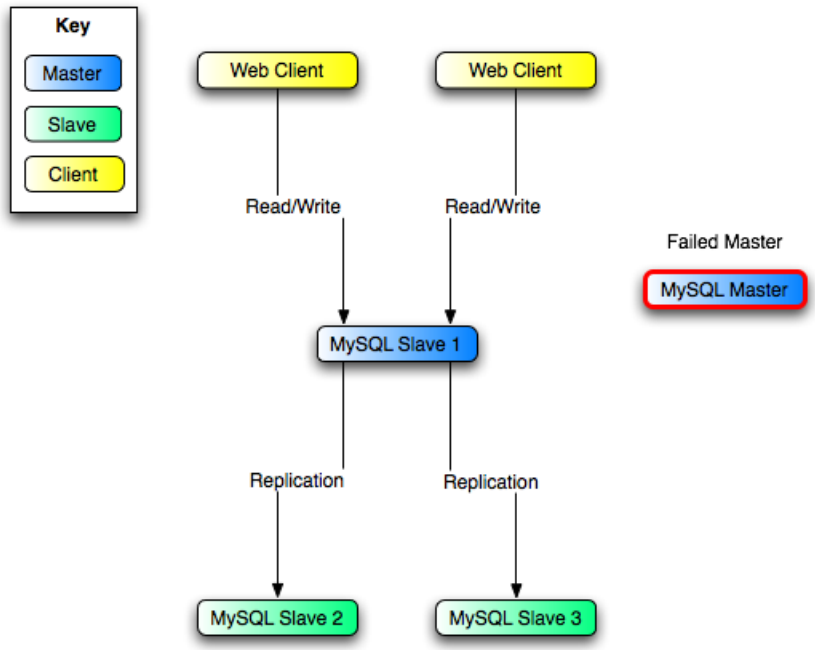
On the other slaves `Slave 2` and `Slave 3`, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='Slave1'` (where `'Slave1'` represents the real host name of `Slave 1`). To use `CHANGE MASTER TO`, add all information about how to connect to `Slave 1` from `Slave 2` or `Slave 3` (*`user`*, *`password`*, *`port`*). When issuing the `CHANGE MASTER TO` statement in this, there is no need to specify the name of the `Slave 1` binary log file or log position to read from, since the first binary log file and position 4, are the defaults. Finally, execute `START SLAVE` on `Slave 2` and `Slave 3`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Once the new replication setup is in place, you need to tell each `Web Client` to direct its statements to `Slave 1`. From that point on, all updates statements sent by `Web Client` to `Slave 1` are written to the binary log of `Slave 1`, which then contains every update statement sent to `Slave 1` since `Master` died.

The resulting server structure is shown in Figure 16.5, “Redundancy Using Replication, After Master Failure”.

Figure 16.5 Redundancy Using Replication, After Master Failure



When `Master` becomes available again, you should make it a slave of `Slave 1`. To do this, issue on `Master` the same `CHANGE MASTER TO` statement as that issued on `Slave 2` and `Slave 3` previously. `Master` then becomes a slave of `S1ave 1` and picks up the `Web Client` writes that it missed while it was offline.

To make `Master` a master again, use the preceding procedure as if `Slave 1` was unavailable and `Master` was to be the new master. During this procedure, do not forget to run `RESET MASTER` on `Master` before making `Slave 1`, `Slave 2`, and `Slave 3` slaves of `Master`. If you fail to do this, the slaves may pick up stale writes from the `Web Client` applications dating from before the point at which `Master` became unavailable.

You should be aware that that there is no synchronization between slaves, even when they share the same master, and thus some slaves might be considerably ahead of others. This means that in some cases the procedure outlined in the previous example might not work as expected. In practice, however, relay logs on all slaves should be relatively close together.

One way to keep applications informed about the location of the master is to have a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to update the DNS dynamically.

16.3.7 Setting Up Replication to Use Secure Connections

To use a secure connection for encrypting the transfer of the binary log required during replication, both the master and the slave servers must support encrypted network connections. If either server does not

This documentation is for an older version. If you're
This documentation is for an older version. If you're

support secure connections (because it has not been compiled or configured for them), replication through an encrypted connection is not possible.

Setting up secure connections for replication is similar to doing so for client/server connections. You must obtain (or create) a suitable security certificate that you can use on the master, and a similar certificate (from the same certificate authority) on each slave. You must also obtain suitable key files.

For more information on setting up a server and client for secure connections, see Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”.

To enable secure connections on the master, you must create or obtain suitable certificate and key files, and then add the following configuration options to the master's configuration within the `[mysqld]` section of the master's `my.cnf` file, changing the file names as necessary:

```
[mysqld]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

The paths to the files may be relative or absolute; we recommend that you always use complete paths for this purpose.

The options are as follows:

- `ssl-ca` identifies the Certificate Authority (CA) certificate.
- `ssl-cert` identifies the server public key certificate. This can be sent to the client and authenticated against the CA certificate that it has.
- `ssl-key` identifies the server private key.

On the slave, there are two ways to specify the information required for connecting securely to the master. You can either name the slave certificate and key files in the `[client]` section of the slave's `my.cnf` file, or you can explicitly specify that information using the `CHANGE MASTER TO` statement:

- To name the slave certificate and key files using an option file, add the following lines to the `[client]` section of the slave's `my.cnf` file, changing the file names as necessary:

  ```
  [client]
  ssl-ca=cacert.pem
  ssl-cert=client-cert.pem
  ssl-key=client-key.pem
  ```

  Restart the slave server, using the `--skip-slave-start` option to prevent the slave from connecting to the master. Use `CHANGE MASTER TO` to specify the master configuration, using the `MASTER_SSL` option to connect securely:

  ```
  mysql> CHANGE MASTER TO
      -> MASTER_HOST='master_hostname',
      -> MASTER_USER='replicate',
      -> MASTER_PASSWORD='password',
      -> MASTER_SSL=1;
  ```

- To specify the certificate and key names using the `CHANGE MASTER TO` statement, append the appropriate `MASTER_SSL_xxx` options:

  ```
  mysql> CHANGE MASTER TO
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    -> MASTER_HOST='master_hostname',
    -> MASTER_USER='replicate',
    -> MASTER_PASSWORD='password',
    -> MASTER_SSL=1,
    -> MASTER_SSL_CA = 'ca_file_name',
    -> MASTER_SSL_CAPATH = 'ca_directory_name',
    -> MASTER_SSL_CERT = 'cert_file_name',
    -> MASTER_SSL_KEY = 'key_file_name';
```

After the master information has been updated, start the slave replication process:

```
mysql> START SLAVE;
```

You can use the `SHOW SLAVE STATUS` statement to confirm that a secure connection was established successfully.

For more information on the `CHANGE MASTER TO` statement, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

If you want to enforce the use of secure connections during replication, then create a user with the `REPLICATION SLAVE` privilege and use the `REQUIRE SSL` option for that user. For example:

```
mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'%.mydomain.com' REQUIRE SSL;
```

If the account already exists, you can add `REQUIRE SSL` to it with this statement:

```
mysql> GRANT USAGE ON *.*
    -> TO 'repl'@'%.mydomain.com' REQUIRE SSL;
```

16.4 Replication Notes and Tips

16.4.1 Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

Statement-based replication depends on compatibility at the SQL level between the master and slave. In others, successful SBR requires that any SQL features used be supported by both the master and the slave servers. For example, if you use a feature on the master server that is available only in MySQL 5.0 (or later), you cannot replicate to a slave that uses MySQL 4.1 (or earlier).

Such incompatibilities also can occur within a release series when using pre-production releases of MySQL. For example, the `SLEEP()` function is available beginning with MySQL 5.0.12. If you use this function on the master, you cannot replicate to a slave that uses MySQL 5.0.11 or earlier.

For this reason, use Generally Available (GA) releases of MySQL for statement-based replication in a production setting, since we do not introduce new SQL statements or change their behavior within a given release series once that series reaches GA release status.

If you are planning to use replication between MySQL 5.0 and a previous MySQL release series, it is also a good idea to consult the edition of the *MySQL Reference Manual* corresponding to the earlier release series for information regarding the replication characteristics of that series.

For additional information specific to replication and `InnoDB`, see Section 14.2.3.5, “InnoDB and MySQL Replication”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

16.4.1.1 Replication and AUTO_INCREMENT

Replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()`, and `TIMESTAMP` values is done correctly, subject to the following exceptions.

- `AUTO_INCREMENT` columns in tables on the slave must match the same columns on the master; that is, `AUTO_INCREMENT` columns must be replicated to `AUTO_INCREMENT` columns.

  This is a known issue which is fixed in MySQL 5.5. (Bug #12669186)

- `INSERT DELAYED ... VALUES(LAST_INSERT_ID())` inserts a different value on the master and the slave. (Bug #20819) This is fixed in MySQL 5.1 when using row-based or mixed-format binary logging. For more information, see Replication Formats.

- Before MySQL 5.0.26, a stored procedure that uses `LAST_INSERT_ID()` does not replicate properly.

- When a statement uses a stored function that inserts into an `AUTO_INCREMENT` column, the generated `AUTO_INCREMENT` value is not written into the binary log, so a different value can in some cases be inserted on the slave. This is also true of a trigger that causes an `INSERT` into an `AUTO_INCREMENT` column.

- An insert into an `AUTO_INCREMENT` column caused by a stored routine or trigger running on a master that uses MySQL 5.0.60 or earlier does not replicate correctly to a slave running MySQL 5.1.12 through 5.1.23 (inclusive). (Bug #33029)

- An `INSERT` into a table that has a composite primary key that includes an `AUTO_INCREMENT` column that is not the first column of this composite key is not logged or replicated correctly.

  This issue does not affect tables using the `InnoDB` storage engine, since `InnoDB` does not allow the creation of a composite key that includes an `AUTO_INCREMENT` column that is not the first column in the key.

- Adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to a table `t1` that has columns `col1` and `col2`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, the `ORDER BY` clause must name *all* columns of `t1`.

The instructions just given are subject to the limitations of `CREATE TABLE ... LIKE`: Foreign key definitions are ignored, as are the `DATA DIRECTORY` and `INDEX DIRECTORY` table options. If a table definition includes any of those characteristics, create `t2` using a `CREATE TABLE` statement that is identical to the one used to create `t1`, but with the addition of the `AUTO_INCREMENT` column.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also Section B.5.6.1, “Problems with ALTER TABLE”.

16.4.1.2 Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- If the master has databases with a character set different from the global `character_set_server` value, you should design your `CREATE TABLE` statements so that they do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.

16.4.1.3 Replication and CHECKSUM TABLE

`CHECKSUM TABLE` returns a checksum that is calculated row by row, using a method that depends on the table row storage format, which is not guaranteed to remain the same between MySQL release series. For example, the storage format for `VARCHAR` changed between MySQL 4.1 and 5.0, so if a 4.1 table is upgraded to MySQL 5.0, the checksum value may change.

16.4.1.4 Replication of CREATE TABLE ... SELECT Statements

This section discusses the rules that are applied when a `CREATE TABLE ... SELECT` statement is replicated.

Note

`CREATE TABLE ... SELECT` always performs an implicit commit (Section 13.3.3, “Statements That Cause an Implicit Commit”).

Statement succeeds. A successful `CREATE TABLE ... SELECT` is itself replicated.

Statement fails. A failed `CREATE TABLE ... SELECT` replicates as follows:

- **Statement does not use IF NOT EXISTS.** The statement has no effect. However, the implicit commit caused by the statement is logged. This is true regardless of the storage engine used and the reason for which the statement failed.
- **Statement uses IF NOT EXISTS.** The `CREATE TABLE IF NOT EXISTS ... SELECT` is logged with an error.

16.4.1.5 Replication of DROP ... IF EXISTS Statements

The `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, and `DROP VIEW IF EXISTS` statements are always replicated, even if the database, table, or view to be dropped does not exist on the master. This is to ensure that the object to be dropped no longer exists on either the master or the slave, once the slave has caught up with the master.

Beginning with MySQL 5.0.82, `DROP ... IF EXISTS` statements for stored programs (stored procedures and functions, triggers, and events) are also replicated, even if the stored program to be dropped does not exist on the master. (Bug #13684)

16.4.1.6 Replication and DIRECTORY Table Options

This documentation is for an older version. If you're

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host file system or if it exists but is not accessible to the slave server. This can be overridden by using the `NO_DIR_IN_CREATE` server SQL mode on the slave, which causes the slave to ignore the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

For more information, see Section 5.1.7, “Server SQL Modes”.

16.4.1.7 Replication and Floating-Point Values

With statement-based replication, values are converted from decimal to binary. Because conversions between decimal and binary representations of them may be approximate, comparisons involving floating-point values are inexact. This is true for operations that use floating-point values explicitly, or that use values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See Section 12.2, “Type Conversion in Expression Evaluation”, and Section B.5.4.8, “Problems with Floating-Point Values”.

16.4.1.8 Replication and FLUSH

Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a slave: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see Section 13.7.6.2, “FLUSH Syntax”. The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the slaves to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you must issue `FLUSH TABLES` manually on the slaves. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

16.4.1.9 Replication and System Functions

Certain functions do not replicate well under some conditions:

- The `USER()`, `CURRENT_USER()`, `UUID()`, `VERSION()`, and `LOAD_FILE()` functions are replicated without change and thus do not work reliably on the slave.
- For `NOW()`, the binary log includes the timestamp. This means that the value *as returned by the call to this function on the master* is replicated to the slave. This can lead to a possibly unexpected result when replicating between MySQL servers in different time zones. Suppose that the master is located in New York, the slave is located in Stockholm, and both servers are using local time. Suppose further that, on the master, you create a table `mytable`, perform an `INSERT` statement on this table, and then select from the table, as shown here:

```
mysql> CREATE TABLE mytable (mycol TEXT);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO mytable VALUES ( NOW() );
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM mytable;
+---------------------+
| mycol               |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

Local time in Stockholm is 6 hours later than in New York; so, if you issue `SELECT NOW()` on the slave at that exact same instant, the value `2009-09-01 18:00:00` is returned. For this reason, if you select from the slave's copy of `mytable` after the `CREATE TABLE` and `INSERT` statements just shown have been replicated, you might expect `mycol` to contain the value `2009-09-01 18:00:00`. However, this is not the case; when you select from the slave's copy of `mytable`, you obtain exactly the same result as on the master:

```
mysql> SELECT * FROM mytable;
+---------------------+
| mycol               |
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

As of MySQL 5.0.13, the `SYSDATE()` function is no longer equivalent to `NOW()`. Implications are that `SYSDATE()` is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is nondeterministic. To avoid this, you can start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.

See also Section 16.4.1.25, “Replication and Time Zones”.

- The `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, and `IS_USED_LOCK()` functions that handle user-level locks are replicated without the slave knowing the concurrency context on the master. Therefore, these functions should not be used to insert into a master table because the content on the slave would differ. For example, do not issue a statement such as `INSERT INTO mytable VALUES(GET_LOCK(...))`.

As a workaround for the preceding limitations, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a later statement. For example, the following single-row `INSERT` is problematic due to the reference to the `UUID()` function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of `@my_uuid` is stored in the binary log as a user-variable event prior to the `INSERT` statement and is available for use in the `INSERT`.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

Non-delayed `INSERT` statements that refer to `RAND()` or user-defined variables replicate correctly. However, changing the statements to use `INSERT DELAYED` can result in different results on master and slave.

Within a stored function, `RAND()` replicates correctly as long as it is invoked only once during the execution of the function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

The `FOUND_ROWS()` and `ROW_COUNT()` functions are also not replicated reliably. A workaround is to store the result of the function call in a user variable, and then use that in the `INSERT` statement. For example, if you wish to store the result in a table named `mytable`, you might normally do so like this:

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

However, if you are replicating `mytable`, you should use `SELECT ... INTO`, and then store the variable in the table, like this:

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

In this way, the user variable is replicated as part of the context, and applied on the slave correctly.

16.4.1.10 Replication and LIMIT

Replication of `LIMIT` clauses in `DELETE`, `UPDATE`, and `INSERT ... SELECT` statements is not guaranteed, since the order of the rows affected is not defined. Such statements can be replicated correctly only if they also contain an `ORDER BY` clause.

16.4.1.11 Replication and LOAD Operations

Using `LOAD TABLE FROM MASTER` where the master is running MySQL 4.1 and the slave is running MySQL 5.0 may corrupt the table data, and is not supported. (Bug #16261)

The `LOAD DATA INFILE` statement `CONCURRENT` option is not replicated; that is, `LOAD DATA CONCURRENT INFILE` is replicated as `LOAD DATA INFILE`, and `LOAD DATA CONCURRENT LOCAL INFILE` is replicated as `LOAD DATA LOCAL INFILE`. (Bug #34628)

The following applies only if either the master or the slave is running MySQL 5.0.3 or older: If on the master a `LOAD DATA INFILE` is interrupted (integrity constraint violation, killed connection, and so on), the slave skips the `LOAD DATA INFILE` entirely. This means that if this command permanently inserted or updated table records before being interrupted, these modifications are not replicated to the slave.

16.4.1.12 Replication and the Slow Query Log

Replication slaves do not write replicated queries to the slow query log, even if the same queries were written to the slow query log on the master. This is a known issue. (Bug #23300)

16.4.1.13 Replication and REPAIR TABLE

When used on a corrupted or otherwise damaged table, it is possible for the `REPAIR TABLE` statement to delete rows that cannot be recovered. However, any such modifications of table data performed by this statement are not replicated, which can cause master and slave to lose synchronization. For this reason,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

in the event that a table on the master becomes damaged and you use `REPAIR TABLE` to repair it, you should first stop replication (if it is still running) before using `REPAIR TABLE`, then afterward compare the master's and slave's copies of the table and be prepared to correct any discrepancies manually, before restarting replication.

16.4.1.14 Replication and Master or Slave Shutdowns

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the `CHANGE MASTER TO` statement or `--master-connect-retry` option. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for `slave_net_timeout` seconds. If your outages are short, you may want to decrease `slave_net_timeout`. See Section 5.1.4, “Server System Variables”.

An unclean shutdown (for example, a crash) on the master side can result in the master binary log having a final position less than the most recent position read by the slave, due to the master binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting `sync_binlog=1` in the master `my.cnf` file helps to minimize this problem because it causes the master to flush its binary log more frequently.

Unclean master shutdowns may cause inconsistencies between the content of tables and the binary log. This can be avoided by using `InnoDB` tables and the `--innodb-safe-binlog` option on the master. See Section 5.4.3, “The Binary Log”.

Note

`--innodb-safe-binlog` is unneeded as of MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support.

Shutting down a slave cleanly is safe because it keeps track of where it left off. However, be careful that the slave does not have temporary tables open; see Section 16.4.1.16, “Replication and Temporary Tables”. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the slave commits and then updates `relay-log.info`. If a crash occurs between these two operations, relay log processing will have proceeded further than the information file indicates and the slave will re-execute the events from the last transaction in the relay log after it has been restarted.
- A similar problem can occur if the slave updates `relay-log.info` but the server host crashes before the write has been flushed to disk. Writes are not forced to disk because the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

16.4.1.15 Replication and MEMORY Tables

When a master server shuts down and restarts, its `MEMORY` (`HEAP`) tables become empty. To replicate this effect to slaves, the first time that the master uses a given `MEMORY` table after startup, it logs an event that notifies slaves that the table must to be emptied by writing a `DELETE` statement for that table to the binary log.

When a slave server shuts down and restarts, its `MEMORY` tables become empty. This causes the slave to be out of synchrony with the master and may lead to other failures or cause the slave to stop. For example,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`INSERT INTO ... SELECT FROM memory_table` may insert a different set of rows on the master and slave.

The safe way to restart a slave that is replicating `MEMORY` tables is to first drop or delete all rows from the `MEMORY` tables on the master and wait until those changes have replicated to the slave. Then it is safe to restart the slave.

The size of `MEMORY` tables is limited by the value of the `max_heap_table_size` system variable, which is not replicated (see Section 16.4.1.29, "Replication and Variables"). A change in `max_heap_table_size` takes effect for `MEMORY` tables that are created or updated using `ALTER TABLE ... ENGINE = MEMORY` or `TRUNCATE TABLE` following the change, or for all `MEMORY` tables following a server restart. If you increase the value of this variable on the master without doing so on the slave, it becomes possible for a table on the master to grow larger than its counterpart on the slave, leading to inserts that succeed on the master but fail on the slave with `Table is full` errors. This is a known issue (Bug #48666). In such cases, you must set the global value of `max_heap_table_size` on the slave as well as on the master, then restart replication. It is also recommended that you restart both the master and slave MySQL servers, to insure that the new value takes complete (global) effect on each of them.

See Section 14.4, "The MEMORY (HEAP) Storage Engine", for more information about `MEMORY` tables.

16.4.1.16 Replication and Temporary Tables

Safe slave shutdown when using temporary tables. Temporary tables are replicated except in the case where you stop the slave server (not just the slave threads) and you have replicated temporary tables that are open for use in updates that have not yet been executed on the slave. If you stop the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a `STOP SLAVE SQL_THREAD` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
3. If the value is not 0, restart the slave SQL thread with `START SLAVE SQL_THREAD` and repeat the procedure later.
4. When the value is 0, issue a `mysqladmin shutdown` command to stop the slave.

Temporary tables and replication options. By default, all temporary tables are replicated; this happens whether or not there are any matching `--replicate-do-db`, `--replicate-do-table`, or `--replicate-wild-do-table` options in effect. However, the `--replicate-ignore-table` and `--replicate-wild-ignore-table` options are honored for temporary tables.

A recommended practice when using replication is to designate a prefix for exclusive use in naming temporary tables that you do not want replicated, then employ a matching `--replicate-wild-ignore-table` option. For example, you might give all such tables names beginning with `norep` (such as `norepmytable`, `norepyourtable`, and so on), then use `--replicate-wild-ignore-table=norep%` to prevent the replication of these tables.

16.4.1.17 Replication of the mysql System Database

User privileges are replicated only if the `mysql` database is replicated. That is, the `GRANT`, `REVOKE`, `SET PASSWORD`, `CREATE USER`, and `DROP USER` statements take effect on the slave only if the replication setup includes the `mysql` database.

See also Section 16.4.1.18, "Replication and User Privileges".

This documentation is for an older version. If you're
This documentation is for an older version. If you're

16.4.1.18 Replication and User Privileges

User privileges are replicated only if the `mysql` database is replicated. That is, the `GRANT`, `REVOKE`, `SET PASSWORD`, `CREATE USER`, and `DROP USER` statements take effect on the slave only if the replication setup includes the `mysql` database.

If you are replicating all databases, but do not want statements that affect user privileges to be replicated, set up the slave not to replicate the `mysql` database, using the `--replicate-wild-ignore-table=mysql.%` option. The slave recognizes that privilege-related SQL statements have no effect, and thus it does not execute those statements.

See Section 16.4.1.17, "Replication of the mysql System Database", for more information.

16.4.1.19 Replication and the Query Optimizer

It is possible for the data on the master and slave to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left up the query optimizer. (In general, this is not a good practice, even outside of replication.) Examples of nondeterministic statements include `DELETE` or `UPDATE` statements that use `LIMIT` with no `ORDER BY` clause; see Section 16.4.1.10, "Replication and LIMIT", for a detailed discussion of these.

Also see Section B.5.7, "Known Issues in MySQL".

16.4.1.20 Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older master to a newer slave and you make use of identifiers on the master that are reserved words in the newer MySQL version running on the slave. An example of this is using a table column named `condition` on a 4.1 master that is replicating to a 5.0 or higher slave because `CONDITION` is a reserved word beginning in MySQL 5.0. Replication can fail in such cases with Error 1064 `You have an error in your SQL syntax...`, *even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the slave prior to execution, so that the slave knows which database object or objects would be affected; only after the event is parsed can the slave apply any filtering rules defined by `--replicate-do-db`, `--replicate-do-table`, `--replicate-ignore-db`, and `--replicate-ignore-table`.

To work around the problem of database, table, or column names on the master which would be regarded as reserved words by the slave, do one of the following:

- Use one or more `ALTER TABLE` statements on the master to change the names of any database objects where these names would be considered reserved words on the slave, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (`` ` ``).

For listings of reserved words by MySQL version, see Reserved Words, in the *MySQL Server Version Reference*. For identifier quoting rules, see Section 9.2, "Schema Object Names".

16.4.1.21 Slave Errors During Replication

If a statement produces the same error (identical error code) on both the master and the slave, the error is logged, but replication continues.

If a statement produces different errors on the master and the slave, the slave SQL thread terminates, and the slave writes a message to its error log and waits for the database administrator to decide what to do

This documentation is for an older version. If you're
This documentation is for an older version. If you're

about the error. This includes the case that a statement produces an error on the master or the slave, but not both. To address the issue, connect to the slave manually and determine the cause of the problem. `SHOW SLAVE STATUS` is useful for this. Then fix the problem and run `START SLAVE`. For example, you might need to create a nonexistent table before you can start the slave again.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` option.

For nontransactional storage engines such as `MyISAM`, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave expects execution of the statement to result in the same error code. If it does not, the slave SQL thread stops as described previously.

If you are replicating between tables that use different storage engines on the master and slave, keep in mind that the same statement might produce a different error when run against one version of the table, but not the other, or might cause an error for one version of the table, but not the other. For example, since `MyISAM` ignores foreign key constraints, an `INSERT` or `UPDATE` statement accessing an `InnoDB` table on the master might cause a foreign key violation but the same statement performed on a `MyISAM` version of the same table on the slave would produce no such error, causing replication to stop.

16.4.1.22 Replication and Server SQL Mode

Using different server SQL mode settings on the master and the slave may cause the same `INSERT` statements to be handled differently on the master and the slave, leading the master and slave to diverge. For best results, you should always use the same server SQL mode on the master and on the slave.

For more information, see Section 5.1.7, “Server SQL Modes”.

16.4.1.23 Replication Retries and Timeouts

In MySQL 5.0 (starting from 5.0.3), there is a global system variable `slave_transaction_retries`: If the slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because it exceeded the `InnoDB` `innodb_lock_wait_timeout` or the NDBCLUSTER `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the slave automatically retries the transaction `slave_transaction_retries` times before stopping with an error. The default value is 10. Starting from MySQL 5.0.4, the total retry count can be seen in the output of `SHOW STATUS`; see Section 5.1.6, “Server Status Variables”.

16.4.1.24 Replication and TIMESTAMP

Older versions of MySQL (prior to 4.1) differed significantly in several ways in their handling of the `TIMESTAMP` data type from what is supported in MySQL versions 5.0 and newer; these include syntax extensions which are deprecated in MySQL 5.1, and that no longer supported in MySQL 5.5. This this can cause problems (including replication failures) when replicating between MySQL Server versions, if you are using columns that are defined using the old `TIMESTAMP(N)` syntax. See Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”, for more information about the differences, how they can impact MySQL replication, and what you can do if you encounter such problems.

16.4.1.25 Replication and Time Zones

The same system time zone should be set for both master and slave. Otherwise, statements depending on the local time on the master are not replicated properly, such as statements that use the `NOW()` or `FROM_UNIXTIME()` functions. You can set the time zone in which MySQL server runs by using the `--timezone=`*`timezone_name`* option of the `mysqld_safe` script or by setting the `TZ` environment variable. See also Section 16.4.1.9, “Replication and System Functions”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`CONVERT_TZ(...,...,@@session.time_zone)` is properly replicated only if both master and slave are running MySQL 5.0.4 or newer.

16.4.1.26 Replication and Transactions

Mixing transactional and nontransactional statements within the same transaction. In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional and nontransactional tables and writes to any of them.

In MySQL 5.0 the server uses this rule for binary logging: If the initial statements in a transaction are nontransactional, they are written to the binary log immediately. The remaining statements in the transaction are cached and not written to the binary log until the transaction is committed. (If the transaction is rolled back, the cached statements are written to the binary log only if they make nontransactional changes that cannot be rolled back. Otherwise, they are discarded.)

To apply this rule, the server considers a statement nontransactional if the first changes it makes change nontransactional tables, transactional if the first changes it makes change transactional tables. “First” applies in the sense that a statement may have several effects if it involves such things as triggers, stored functions, or multiple-table updates.

In situations where transactions mix updates to transactional and nontransactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the nontransactional table before the first connection transaction is complete, statements can be logged out of order because the second connection update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

Using different storage engines on master and slave. It is possible to replicate transactional tables on the master using nontransactional tables on the slave. For example, you can replicate an `InnoDB` master table as a `MyISAM` slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN` ... `COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.

When the storage engine type of the slave is nontransactional, transactions on the master that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the master transactional table and the slave nontransactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the master to nontransactional tables on the slaves.

Beginning with MySQL 5.0.56, every transaction (including `autocommit` transactions) is recorded in the binary log as though it starts with a `BEGIN` statement, and ends with either a `COMMIT` or a `ROLLBACK` statement. However, this does *not* apply to nontransactional changes; any statements affecting tables using a nontransactional storage engine such as `MyISAM` are regarded for this purpose as nontransactional, even when `autocommit` is enabled. (Bug #26395)

16.4.1.27 Replication and Triggers

Known issue: In MySQL 5.0.17, the syntax for `CREATE TRIGGER` changed to include a `DEFINER` clause for specifying which access privileges to check at trigger invocation time. (See Section 13.1.11, “CREATE TRIGGER Syntax”, for more information.) However, if you attempt to replicate from a master server older than MySQL 5.0.17 to a slave running MySQL 5.0.17 through 5.0.19, replication of `CREATE TRIGGER` statements fails on the slave with a `Definer not fully qualified` error. A workaround is to create triggers on the master using a version-specific comment embedded in each `CREATE TRIGGER` statement:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE /*!50017 DEFINER = 'root'@'localhost' */ TRIGGER ... ;
```

`CREATE TRIGGER` statements written this way will replicate to newer slaves, which pick up the `DEFINER` clause from the comment and execute successfully.

This slave problem is fixed as of MySQL 5.0.20.

16.4.1.28 Replication and Views

Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by `replication-ignore-table` rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

16.4.1.29 Replication and Variables

The `foreign_key_checks`, `unique_checks`, and `sql_auto_is_null` variables are all replicated.

`sql_mode` is also replicated except for the `NO_DIR_IN_CREATE` mode. However, when `mysqlbinlog` parses a `SET @@sql_mode = mode` statement, the full *`mode`* value, including `NO_DIR_IN_CREATE`, is passed to the receiving server.

The `storage_engine` system variable is not replicated, regardless of the logging mode; this is intended to facilitate replication between different storage engines.

The `read_only` system variable is not replicated. In addition, the enabling this variable has different effects with regard to temporary tables, table locking, and the `SET PASSWORD` statement in different MySQL versions.

The `max_heap_table_size` system variable is not replicated. Increasing the value of this variable on the master without doing so on the slave can lead eventually to `Table is full` errors on the slave when trying to execute `INSERT` statements on a `MEMORY` table on the master that is thus permitted to grow larger than its counterpart on the slave. For more information, see Section 16.4.1.15, “Replication and MEMORY Tables”.

Starting from MySQL 5.0.3 (master and slave), replication works even if the master and slave have different global character set variables. Starting from MySQL 5.0.4 (master and slave), replication works even if the master and slave have different global time zone variables.

Session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements will not insert the same data on the master and the slave:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

This does not apply to the common sequence, which replicates correctly as of MySQL 5.0.4.

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

Update statements that refer to user-defined variables (that is, variables of the form *`@var_name`*) are replicated correctly in MySQL 5.0. However, this is not true for versions prior to 4.1. Note that user variable names are case insensitive starting in MySQL 5.0. You should take this into account when setting up replication between MySQL 5.0 and older versions.

In MySQL 5.0.46 and later, the following session variables are written to the binary log and honored by the replication slave when parsing the binary log, regardless of the logging format:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `sql_mode`
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

Important

Even though session variables relating to character sets and collations are written to the binary log, replication between different character sets is not supported.

It is strongly recommended that you always use the same setting for the `lower_case_table_names` system variable on both master and slave. In particular, when a case-sensitive file system is used, and this variable set to 1 on the slave, but to a different value on the master, names of databases are not converted to lowercase, which can cause replication to fail. This is a known issue, which is fixed in MySQL 5.6.

16.4.2 Replication Compatibility Between MySQL Versions

MySQL supports replication from one release series to the next higher release series. For example, you can replicate from a master running MySQL 4.1 to a slave running MySQL 5.0, from a master running MySQL 5.0 to a slave running MySQL 5.1, and so on.

However, one may encounter difficulties when replicating from an older master to a newer slave if the master uses statements or relies on behavior no longer supported in the version of MySQL used on the slave.

The use of more than two MySQL Server versions is not supported in replication setups involving multiple masters, regardless of the number of master or slave MySQL servers. This restriction applies not only to release series, but to version numbers within the same release series as well. For example, if you are using a chained or circular replication setup, you cannot use MySQL 5.0.21, MySQL 5.0.22, and MySQL 5.0.24 concurrently, although you could use any two of these releases together.

In some cases, it is also possible to replicate between a master and a slave that is more than one major release newer than the master. However, there are known issues with trying to replicate from a master running MySQL 4.1 or earlier to a slave running MySQL 5.1 or later. To work around such problems, you can insert a MySQL server running an intermediate version between the two; for example, rather than replicating directly from a MySQL 4.1 master to a MySQL 5.1 slave, it is possible to replicate from a MySQL 4.1 server to a MySQL 5.0 server, and then from the MySQL 5.0 server to a MySQL 5.1 server.

Important

It is strongly recommended to use the most recent release available within a given MySQL release series because replication (and other) capabilities are continually being improved. It is also recommended to upgrade masters and slaves that use early releases of a release series of MySQL to GA (production) releases when the latter become available for that release series.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Replication from newer masters to older slaves may be possible, but is generally not supported. This is due to a number of factors:

- **Binary log format changes.** The binary log format can change between major releases. While we attempt to maintain backward-compatiblity, this is not always possible. Major changes were made in MySQL 5.0.3 (for improvements to handling of character sets and `LOAD DATA INFILE`) and 5.0.4 (for improvements to handling of time zones). Because of these changes, replication from a MySQL 5.0.3 or later master to a MySQL 5.0.2 or earlier slave is not supported. This also means that replication from a MySQL 5.0.3 (or later) master to any MySQL 4.1 (or earlier) slave is generally not supported.

  This also has significant implications for upgrading replication servers; see Section 16.4.3, "Upgrading a Replication Setup", for more information.

- **Use of row-based replication.** Row-based replication was implemented in MySQL 5.1.5, so you cannot replicate using row-based replication from any MySQL 5.0 or later master to a slave older than MySQL 5.1.5.

  Note

  Row-based replication is not available in MySQL 5.0. For more information about row-based replication in MySQL 5.1, see Replication Formats.

- **SQL incompatibilities.** You cannot replicate from a newer master to an older slave using statement-based replication if the statements to be replicated use SQL features available on the master but not on the slave.

For more information on potential replication issues, see Section 16.4.1, "Replication Features and Issues".

16.4.3 Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

This section applies to upgrading replication from older versions of MySQL to MySQL 5.0. A 4.0 server should be 4.0.3 or newer.

When you upgrade a master to 5.0 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 5.0.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 5.0.x version, restart it, and restart replication. The 5.0 slave is able to read the old relay logs written prior to the upgrade and to execute the statements they contain. Relay logs created by the slave after the upgrade are in 5.0 format.

After the slaves have been upgraded, shut down the master, upgrade it to the same 5.0.x release as the slaves, and restart it. The 5.0 master is able to read the old binary logs written prior to the upgrade and to send them to the 5.0 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master subsequent to the upgrade are in 5.0 format. These too are recognized by the 5.0 slaves.

In other words, when upgrading to MySQL 5.0, the slaves must be MySQL 5.0 before you can upgrade the master to 5.0. Note that downgrading from 5.0 to older versions does not work so simply: You must ensure that any 5.0 binary log or relay log has been fully processed, so that you can remove it before proceeding with the downgrade.

Some upgrades may require that you drop and re-create database objects when you move from one MySQL series to the next. For example, collation changes might require that table indexes be rebuilt. Such operations, if necessary, will be detailed at Section 2.19.1.1, "Changes Affecting Upgrades to 5.0". It is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

safest to perform these operations separately on the slaves and the master, and to disable replication of these operations from the master to the slave. To achieve this, use the following procedure:

1. Stop all the slaves and upgrade them. Restart them with the `--skip-slave-start` option so that they do not connect to the master. Perform any table repair or rebuilding operations needed to re-create database objects, such as use of `REPAIR TABLE` or `ALTER TABLE`, or dumping and reloading tables or triggers.

2. Disable the binary log on the master. To do this without restarting the master, execute a `SET sql_log_bin = 0` statement. Alternatively, stop the master and restart it without the `--log-bin` option. If you restart the master, you might also want to disallow client connections. For example, if all clients connect using TCP/IP, use the `--skip-networking` option when you restart the master.

3. With the binary log disabled, perform any table repair or rebuilding operations needed to re-create database objects. The binary log must be disabled during this step to prevent these operations from being logged and sent to the slaves later.

4. Re-enable the binary log on the master. If you set `sql_log_bin` to 0 earlier, execute a `SET sql_log_bin = 1` statement. If you restarted the master to disable the binary log, restart it with `--log-bin`, and without `--skip-networking` so that clients and slaves can connect.

5. Restart the slaves, this time without the `--skip-slave-start` option.

16.4.4 Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` option.

- Verify that the master and slave both were started with the `--server-id` option and that the ID value is unique on each server.

- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` prevents the slave threads from starting until you issue a `START SLAVE` statement.

- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See Section 16.2.1, “Replication Implementation Details”. If the I/O thread state says `Connecting to master`, check the following:

  - Verify the privileges for the user being used for replication on the master.

  - Check that the host name of the master is correct and that you are using the correct port to connect to the master. The port used for replication is the same as used for client network communication (the default is `3306`). For the host name, ensure that the name resolves to the correct IP address.

  - Check that networking has not been disabled on the master or slave. Look for the `skip-networking` option in the configuration file. If present, comment it out or remove it.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - If the master has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
  - Check that you can reach the master by using `ping` or `traceroute`/`tracert` to reach the host.

- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in Section 16.4.1, “Replication Features and Issues”. If it is a bug, see Section 16.4.5, “How to Report Replication Bugs or Problems”, for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:

  1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.
  2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.
  3. If you decide that the slave can skip the next statement from the master, issue the following statements:

     ```
     mysql> SET GLOBAL sql_slave_skip_counter = N;
     mysql> START SLAVE;
     ```

     The value of *`N`* should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()`. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` is that they take two events in the binary log of the master.

     See also Section 13.4.2.6, “SET GLOBAL sql_slave_skip_counter Syntax”.
  4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

16.4.5 How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*

2. Run the slave with the `--log-slave-updates` and `--log-bin` options. These options cause the slave to log the updates that it receives from the master into its own binary logs.

3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:

   - All binary log files from the master
   - All binary log files from the slave
   - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem
   - The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
   - Error logs from the master and the slave

4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. *`log_file`* and *`log_pos`* are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

   ```
   shell> mysqlbinlog --start-position=log_pos log_file | head
   ```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at Section 1.7, “How to Report Bugs or Problems”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 17 MySQL Cluster

Table of Contents

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This chapter contains information about *MySQL Cluster*, a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the `NDBCLUSTER` storage engine to enable running several computers with MySQL servers and other software in a cluster. This storage engine is available in MySQL 5.0 binary releases and in RPMs compatible with most modern Linux distributions.

Beginning with MySQL 5.0.8, MySQL Cluster changes for MySQL 5.0 Server releases can be found in the MySQL 5.0 Server Release Notes. For release notes for older releases of MySQL Cluster (before 5.0.8), see MySQL Cluster 5.0 Release Notes.

Supported Platforms. MySQL Cluster is currently available and supported on a number of platforms. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to http://www.mysql.com/support/supportedplatforms/cluster.html.

Compatibility with standard MySQL. While many standard MySQL schemas and applications can work using MySQL Cluster, it is also true that unmodified applications and database schemas may be slightly incompatible or have suboptimal performance when run using MySQL Cluster (see Section 17.1.5, “Known Limitations of MySQL Cluster”). Most of these issues can be overcome, but this also means that you are very unlikely to be able to switch an existing application datastore—that currently uses, for example, `MyISAM` or `InnoDB`—to use the `NDB` storage engine without allowing for the possibility of changes in schemas, queries, and applications.

Beginning with MySQL Cluster NDB 7.1, MySQL Cluster is available for production use on Microsoft Windows. MySQL Cluster is not available for Microsoft Windows in MySQL 5.0. For more information, see MySQL Cluster NDB 6.1 - 7.1.

This chapter represents a work in progress, and its contents are subject to revision as MySQL Cluster continues to evolve. Additional information regarding MySQL Cluster can be found on the MySQL Web site at http://www.mysql.com/products/cluster/.

Additional Resources. More information about MySQL Cluster can be found in the following places:

- For answers to some commonly asked questions about MySQL Cluster, see Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”.
- The MySQL Cluster mailing list: http://lists.mysql.com/cluster.
- The MySQL Cluster Forum: http://forums.mysql.com/list.php?25.
- Many MySQL Cluster users and developers blog about their experiences with MySQL Cluster, and make feeds of these available through PlanetMySQL.

17.1 MySQL Cluster Overview

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

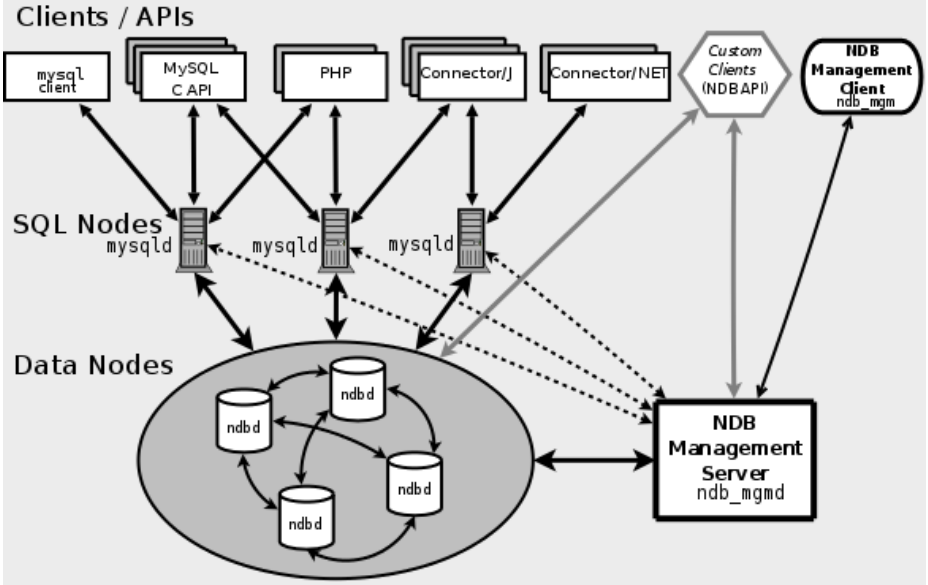
MySQL Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called NDB (which stands for "*N*etwork *D*ata*B*ase"). In our documentation, the term NDB refers to the part of the setup that is specific to the storage engine, whereas "MySQL Cluster" refers to the combination of one or more MySQL servers with the NDB storage engine.

A MySQL Cluster consists of a set of computers, known as *hosts*, each running one or more processes. These processes, known as *nodes*, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in a MySQL Cluster is shown here:

Figure 17.1 MySQL Cluster Components



All these programs work together to form a MySQL Cluster (see Section 17.4, "MySQL Cluster Programs". When data is stored by the NDB storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers (SQL nodes) in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

However, a MySQL server that is not connected to a MySQL Cluster cannot use the NDB storage engine and cannot access any MySQL Cluster data.

The data stored in the data nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades (see Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster"). For more information about data nodes, how they are organized in a MySQL Cluster, and how they handle and store MySQL Cluster data, see Section 17.1.2, "MySQL Cluster Nodes, Node Groups, Replicas, and Partitions".

Backing up and restoring MySQL Cluster databases can be done using the NDB native functionality found in the MySQL Cluster management client and the `ndb_restore` program included in the MySQL Cluster distribution. For more information, see Section 17.5.3, "Online Backup of MySQL Cluster", and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 17.4.14, “ndb_restore — Restore a MySQL Cluster Backup”. You can also use the standard MySQL functionality provided for this purpose in mysqldump and the MySQL server. See Section 4.5.4, “mysqldump — A Database Backup Program”, for more information.

MySQL Cluster nodes can use a number of different transport mechanisms for inter-node communications, including TCP/IP using standard 100 Mbps or faster Ethernet hardware. It is also possible to use the high-speed *Scalable Coherent Interface* (SCI) protocol with MySQL Cluster, although this is not required to use MySQL Cluster. SCI requires special hardware and software; see Section 17.3.4, “Using High-Speed Interconnects with MySQL Cluster”, for more about SCI and using it with MySQL Cluster.

17.1.1 MySQL Cluster Core Concepts

NDBCLUSTER (also known as NDB) is an in-memory storage engine offering high-availability and data-persistence features.

The NDBCLUSTER storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. MySQL Cluster's NDB storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The “Cluster” portion of MySQL Cluster is configured independently of the MySQL servers. In a MySQL Cluster, each part of the cluster is considered to be a *node*.

Note

In many contexts, the term “node” is used to indicate a computer, but when discussing MySQL Cluster it means a *process*. It is possible to run multiple nodes on a single computer; for a computer on which one or more cluster nodes are being run we use the term *cluster host*.

However, MySQL 5.0 does not support the use of multiple data nodes on a single computer in a production setting. See Section 17.1.5.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

There are three types of cluster nodes, and in a minimal MySQL Cluster configuration, there will be at least three nodes, one of each of these types:

- *Management node* (MGM node): The role of this type of node is to manage the other nodes within the MySQL Cluster, performing such functions as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. An MGM node is started with the command ndb_mgmd.

- *Data node*: This type of node stores cluster data. There are as many data nodes as there are replicas, times the number of fragments (see Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”). For example, with two replicas, each having two fragments, you need four data nodes. One replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have 2 (or more) replicas to provide redundancy, and thus high availability. A data node is started with the command ndbd (see Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”).

  MySQL Cluster tables in MySQL 5.0 are stored completely in memory rather than on disk (this is why we refer to MySQL cluster as an *in-memory* database). In MySQL 5.1, MySQL Cluster NDB 6.X, and later, some MySQL Cluster data can be stored on disk, but we do not expect to backport this functionality to MySQL 5.0; see MySQL Cluster Disk Data Tables, for more information.

- *SQL node*: This is a node that accesses the cluster data. In the case of MySQL Cluster, an SQL node is a traditional MySQL server that uses the NDBCLUSTER storage engine. An SQL node is a mysqld

This documentation is for an older version. If you're
This documentation is for an older version. If you're

process started with the `--ndbcluster` and `--ndb-connectstring` options, which are explained elsewhere in this chapter, possibly with additional MySQL server options as well.

An SQL node is actually just a specialized type of *API node*, which designates any application which accesses Cluster data. Another example of an API node is the `ndb_restore` utility that is used to restore a cluster backup. It is possible to write such applications using the NDB API. For basic information about the NDB API, see Getting Started with the NDB API.

Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; to benefit from MySQL Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, replicas, and partitions in MySQL Cluster, see Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”.

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. MySQL Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server (MGM node) manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These are of two types:

- **Standard MySQL clients.** MySQL Cluster can be used with existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on. Such client applications send SQL statements to and receive responses from MySQL servers acting as MySQL Cluster SQL nodes in much the same way that they interact with standalone MySQL servers. However, MySQL clients using a MySQL Cluster as a data source can be modified to take advantage of the ability to connect with multiple MySQL servers to achieve load balancing and failover. For example, Java clients using Connector/J 5.0.6 and later can use `jdbc:mysql:loadbalance://` URLs (improved in Connector/J 5.1.7) to achieve load balancing transparently.

- **Management clients.** These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on. Such clients—such as the `ndb_mgm` management client supplied with MySQL Cluster (see Section 17.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”)—are written using the MGM API, a C-language API that communicates directly with one or more MySQL Cluster management servers. For more information, see The MGM API.

Event logs. MySQL Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in Section 17.5.6, “Event Reports Generated in MySQL Cluster”. Event logs are of two types:

- **Cluster log.** Keeps a record of all desired reportable events for the cluster as a whole.

- **Node log.** A separate log which is also kept for each individual node.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Checkpoint. Generally speaking, when data is saved to disk, it is said that a checkpoint has been reached. More specific to Cluster, it is a point in time where all committed transactions are stored on disk. With regard to the `NDB` storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained:

- *Local Checkpoint (LCP)*: This is a checkpoint that is specific to a single node; however, LCPs take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.
- *Global Checkpoint (GCP)*: A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

For more information about the files and directories created by local checkpoints and global checkpoints, see MySQL Cluster Data Node File System Directory Files.

17.1.2 MySQL Cluster Nodes, Node Groups, Replicas, and Partitions

This section discusses the manner in which MySQL Cluster divides and duplicates data for storage.

Central to an understanding of this topic are the following concepts, listed here with brief definitions:

- **(Data) Node.** An `ndbd` process, which stores a *replica* —that is, a copy of the *partition* (see below) assigned to the node group of which the node is a member.

  Each data node should be located on a separate computer. While it is also possible to host multiple `ndbd` processes on a single computer, such a configuration is not supported.

  It is common for the terms “node” and “data node” to be used interchangeably when referring to an `ndbd` process; where mentioned, management (MGM) nodes (`ndb_mgmd` processes) and SQL nodes (`mysqld` processes) are specified as such in this discussion.

- **Node Group.** A node group consists of one or more nodes, and stores partitions, or sets of *replicas* (see next item).

  The number of node groups in a MySQL Cluster is not directly configurable; it is function of the number of data nodes and of the number of replicas (`NoOfReplicas` configuration parameter), as shown here:

  ```
  [number_of_node_groups] = number_of_data_nodes / NoOfReplicas
  ```

  Thus, a MySQL Cluster with 4 data nodes has 4 node groups if `NoOfReplicas` is set to 1 in the `config.ini` file, 2 node groups if `NoOfReplicas` is set to 2, and 1 node group if `NoOfReplicas` is set to 4. Replicas are discussed later in this section; for more information about `NoOfReplicas`, see Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”.

Note

All node groups in a MySQL Cluster must have the same number of data nodes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

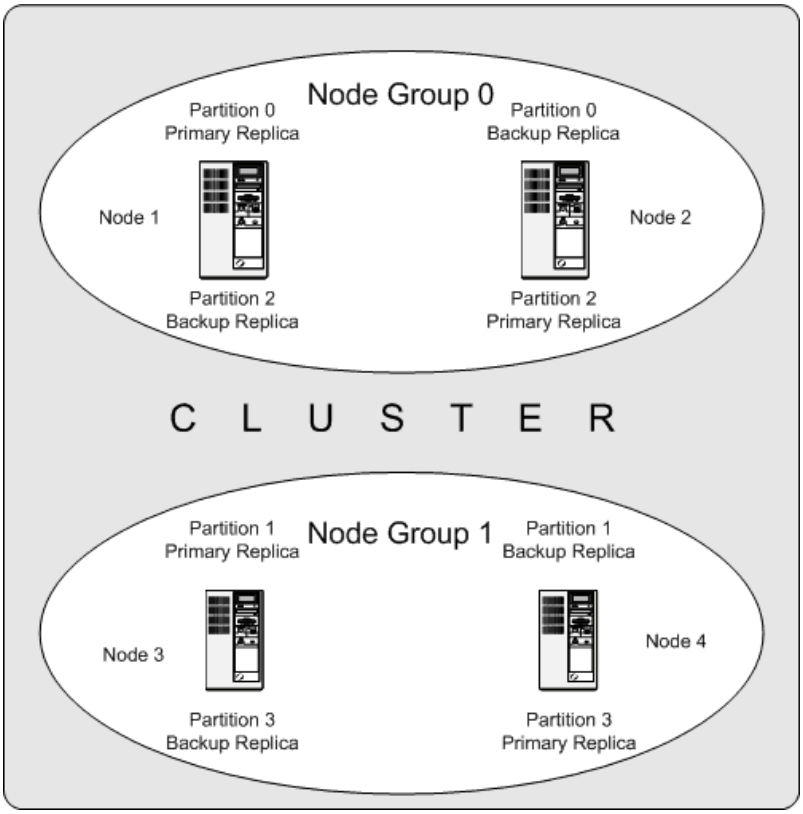
- **Partition.** This is a portion of the data stored by the cluster. There are as many cluster partitions as nodes participating in the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one replica) available to the cluster.

  A replica belongs entirely to a single node; a node can (and usually does) store several replicas.

- **Replica.** This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of replicas is equal to the number of nodes per node group.

The following diagram illustrates a MySQL Cluster with four data nodes, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1. Note that only data (`ndbd`) nodes are shown here; although a working cluster requires an `ndb_mgm` process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted in the figure for clarity.

Figure 17.2 MySQL Cluster with Two Node Groups



The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored—in multiple copies—on the same node group. Partitions are stored on alternate node groups:

- Partition 0 is stored on node group 0; a *primary replica* (primary copy) is stored on node 1, and a *backup replica* (backup copy of the partition) is stored on node 2.

- Partition 1 is stored on the other node group (node group 1); this partition's primary replica is on node 3, and its backup replica is on node 4.

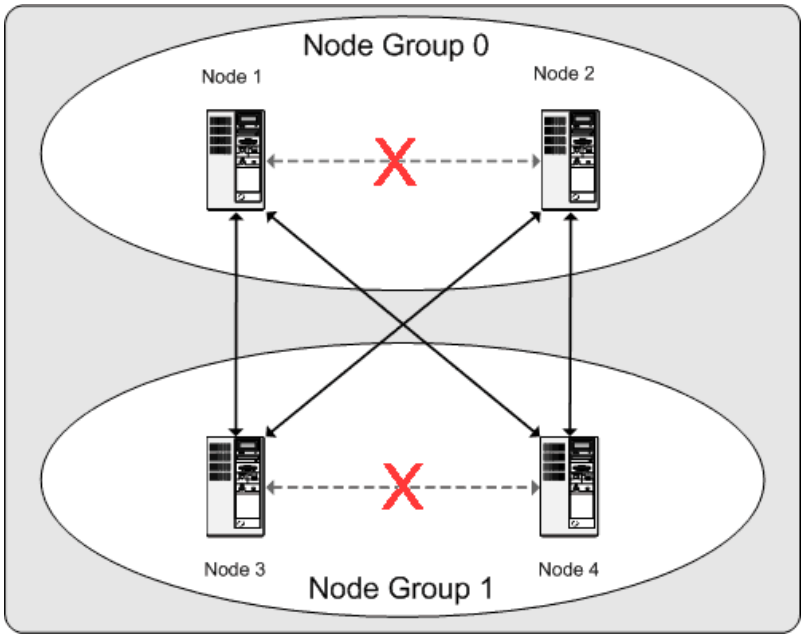
- Partition 2 is stored on node group 0. However, the placing of its two replicas is reversed from that of Partition 0; for Partition 2, the primary replica is stored on node 2, and the backup on node 1.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Partition 3 is stored on node group 1, and the placement of its two replicas are reversed from those of partition 1. That is, its primary replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of a MySQL Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.

Figure 17.3 Nodes Required for a 2x2 Cluster



In this example, where the cluster consists of two node groups of two nodes each, any combination of at least one node in node group 0 and at least one node in node group 1 is sufficient to keep the cluster “alive” (indicated by arrows in the diagram). However, if *both* nodes from *either* node group fail, the remaining two nodes are not sufficient (shown by the arrows marked out with an **X**); in either case, the cluster has lost an entire partition and so can no longer provide access to a complete set of all cluster data.

17.1.3 MySQL Cluster Hardware, Software, and Networking Requirements

One of the strengths of MySQL Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (It is possible to reduce this requirement using Disk Data tables, which are implemented in MySQL 5.1; however, we do not intend to backport this feature to MySQL 5.0.) Naturally, multiple and faster CPUs will enhance performance. Memory requirements for other MySQL Cluster processes are relatively small.

The software requirements for MySQL Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support MySQL Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of MySQL 5.0 to have Cluster support. It is not necessary to compile MySQL yourself merely to be able to use MySQL Cluster. We assume that you are using the server binary appropriate to your platform, available from the MySQL Cluster software downloads page at http://dev.mysql.com/downloads/cluster/.

For communication between nodes, MySQL Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or

This documentation is for an older version. If you're
This documentation is for an older version. If you're

router to provide network connectivity for the cluster as a whole. We strongly recommend that a MySQL Cluster be run on its own subnet which is not shared with machines not forming part of the cluster for the following reasons:

- **Security.** Communications between MySQL Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within a MySQL Cluster is to run the cluster on a protected network. If you intend to use MySQL Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.

  See Section 17.5.10.1, “MySQL Cluster Security and Networking Issues”, for more information.

- **Efficiency.** Setting up a MySQL Cluster on a private or protected network enables the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your MySQL Cluster not only helps protect against unauthorized access to cluster data, it also ensures that MySQL Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

Network communication and latency. MySQL Cluster requires communication between data nodes and API nodes (including SQL nodes), as well as between data nodes and other data nodes, to execute queries and updates. Communication latency between these processes can directly affect the observed performance and latency of user queries. In addition, to maintain consistency and service despite the silent failure of nodes, MySQL Cluster uses heartbeating and timeout mechanisms which treat an extended loss of communication from a node as node failure. This can lead to reduced redundancy. Recall that, to maintain data consistency, a MySQL Cluster shuts down when the last node in a node group fails. Thus, to avoid increasing the risk of a forced shutdown, breaks in communication between nodes should be avoided wherever possible.

The failure of a data or API node results in the abort of all uncommitted transactions involving the failed node. Data node recovery requires synchronization of the failed notde's data from a surviving data node, and re-establishment of disk-based redo and checkpoint logs, before the data node returns to service. This recovery can take some time, during which the Cluster operates with reduced redundancy.

Heartbeating relies on timely generation of heartbeat signals by all nodes. This may not be possible if the node is overloaded, has insufficient machine CPU due to sharing with other programs, or is experiencing delays due to swapping. If heartbeat generation is sufficiently delayed, other nodes treat the node that is slow to respond as failed.

This treatment of a slow node as a failed one may or may not be desireable in some circumstances, depending on the impact of the node's slowed operation on the rest of the cluster. When setting timeout values such as `HeartbeatIntervalDbDb` and `HeartbeatIntervalDbApi` for MySQL Cluster, care must be taken care to achieve quick detection, failover, and return to service, while avoiding potentially expensive false positives.

Where communication latencies between data nodes are expected to be higher than would be expected in a LAN environment (on the order of 100 µs), timeout parameters must be increased to ensure that any allowed periods of latency periods are well within configured timeouts. Increasing timeouts in this way has a corresponding effect on the worst-case time to detect failure and therefore time to service recovery.

LAN environments can typically be configured with stable low latency, and such that they can provide redundancy with fast failover. Individual link failures can be recovered from with minimal and controlled latency visible at the TCP level (where MySQL Cluster normally operates). WAN environments may offer a range of latencies, as well as redundancy with slower failover times. Individual link failures may require route changes to propagate before end-to-end connectivity is restored. At the TCP level this can appear as large latencies on individual channels. The worst-case observed TCP latency in these scenarios is related to the worst-case time for the IP layer to reroute around the failures.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SCI support. It is also possible to use the high-speed Scalable Coherent Interface (SCI) with MySQL Cluster, but this is not a requirement. See Section 17.3.4, “Using High-Speed Interconnects with MySQL Cluster”, for more about this protocol and its use with MySQL Cluster.

17.1.4 What is New in MySQL Cluster

In this section, we discuss changes in the implementation of MySQL Cluster in MySQL 5.0 as compared to MySQL 4.1.

There are relatively few changes between the NDB storage engine implementations in MySQL 4.1 and in 5.0, so the upgrade path should be relatively quick and painless.

All significantly new features being developed for MySQL Cluster are going into the MySQL Cluster NDB 7.x trees. For information on changes in the Cluster implementations in MySQL versions 5.1 and later, see MySQL Cluster Development History.

MySQL Cluster in MySQL 5.0 contains a number of features added since MySQL 4.1 that are likely to be of interest:

- **Condition pushdown.** Consider the following query:

  ```
  SELECT * FROM t1 WHERE non_indexed_attribute = 1;
  ```

  This query uses a full table scan and the condition is evaluated in the cluster's data nodes. Thus, it is not necessary to send the records across the network for evaluation. (That is, function transport is used, rather than data transport.) Please note that this feature is currently disabled by default (pending more thorough testing), but it should work in most cases. This feature can be enabled through the use of the `SET engine_condition_pushdown = On` statement. Alternatively, you can run `mysqld` with the this feature enabled by starting the MySQL server with the `--engine-condition-pushdown` option.

  A major benefit of this change is that queries can be executed in parallel. This means that queries against nonindexed columns can run faster than previously by a factor of as much as 5 to 10 times, *times the number of data nodes*, because multiple CPUs can work on the query in parallel.

  You can use `EXPLAIN` to determine when condition pushdown is being used. See Section 13.8.2, “EXPLAIN Syntax”.

- **Decreased `IndexMemory` Usage**: In MySQL 5.0, each record consumes approximately 25 bytes of index memory, and every unique index uses 25 bytes per record of index memory (in addition to some data memory because these are stored in a separate table). This is because the primary key is not stored in the index memory anymore.

- **Query Cache Enabled for MySQL Cluster**: See Section 8.10.3, “The MySQL Query Cache”, for information on configuring and using the query cache.

- **New optimizations.** One optimization that merits particular attention is that a batched read interface is now used in some queries. For example, consider the following query:

  ```
  SELECT * FROM t1 WHERE primary_key IN (1,2,3,4,5,6,7,8,9,10);
  ```

  This query will be executed 2 to 3 times more quickly than in previous MySQL Cluster versions due to the fact that all 10 key lookups are sent in a single batch rather than one at a time.

- **Limit On Number of Metadata Objects**: Beginning with MySQL 5.0.6, each Cluster database may contain a maximum of 20320 metadata objects—this includes database tables, system tables, indexes and `BLOB` values. (Previously, this number was 1600.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

17.1.5 Known Limitations of MySQL Cluster

In the sections that follow, we discuss known limitations of MySQL Cluster in MySQL 5.0 as compared with the features available when using the `MyISAM` and `InnoDB` storage engines. Currently, there are no plans to address these in coming releases of MySQL 5.0; however, we will attempt to supply fixes for these issues in subsequent release series. If you check the “Cluster” category in the MySQL bugs database at http://bugs.mysql.com, you can find known bugs in the following categories under “MySQL Server:” in the MySQL bugs database at http://bugs.mysql.com, which we intend to correct in upcoming releases of MySQL Cluster:

- Cluster
- Cluster Direct API (NDBAPI)
- Cluster Disk Data
- Cluster Replication

This information is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If we do not plan to fix the problem in MySQL 5.0, we will add it to the list.

See Section 17.1.5.10, “Previous MySQL Cluster Issues Resolved in MySQL 5.0” for a list of issues in MySQL Cluster in MySQL 4.1 that have been resolved in the current version.

17.1.5.1 Noncompliance with SQL Syntax in MySQL Cluster

Some SQL statements relating to certain MySQL features produce errors when used with `NDB` tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the `NDB` storage engine or to alter an existing temporary table to use `NDB` fails with the error `Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.
- **Indexes and keys in NDB tables.** Keys and indexes on MySQL Cluster tables are subject to the following limitations:
  - **Column width.** Attempting to create an index on an `NDB` table column whose width is greater than 3072 bytes succeeds, but only the first 3072 bytes are actually used for the index. In such cases, a warning `Specified key was too long; max key length is 3072 bytes` is issued, and a `SHOW CREATE TABLE` statement shows the length of the index as 3072.
  - **TEXT and BLOB columns.** You cannot create indexes on `NDB` table columns that use any of the `TEXT` or `BLOB` data types.
  - **FULLTEXT indexes.** The `NDB` storage engine does not support `FULLTEXT` indexes, which are possible for `MyISAM` tables only.

    However, you can create indexes on `VARCHAR` columns of `NDB` tables.
  - **USING HASH keys and NULL.** Using nullable columns in unique keys and primary keys means that queries using these columns are handled as full table scans. To work around this issue, make the column `NOT NULL`, or re-create the index without the `USING HASH` option.
  - **Prefixes.** There are no prefix indexes; only entire columns can be indexed. (The size of an `NDB` column index is always the same as the width of the column in bytes, up to and including 3072 bytes,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

as described earlier in this section. Also see Section 17.1.5.6, “Unsupported or Missing Features in MySQL Cluster”, for additional information.)

- **BIT columns.** A `BIT` column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.

- **AUTO_INCREMENT columns.** Like other MySQL storage engines, the `NDB` storage engine can handle a maximum of one `AUTO_INCREMENT` column per table, and this column must be indexed. However, in the case of a MySQL Cluster table with no explicit primary key, an `AUTO_INCREMENT` column is automatically defined and used as a “hidden” primary key. For this reason, you cannot create an `NDB` table having an `AUTO_INCREMENT` column and no explicit primary key.

- **MySQL Cluster and geometry data types.**
  Geometry data types (`WKT` and `WKB`) are supported in `NDB` tables in MySQL 5.0. However, spatial indexes are not supported.

17.1.5.2 Limits and Differences of MySQL Cluster from Standard MySQL Limits

In this section, we list limits found in MySQL Cluster that either differ from limits found in, or that are not found in, standard MySQL.

Memory usage and recovery. Memory consumed when data is inserted into an `NDB` table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:

- A `DELETE` statement on an `NDB` table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. However, this memory can be made available for general re-use by performing a rolling restart of the cluster. See Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”.

- A `DROP TABLE` or `TRUNCATE TABLE` operation on an `NDB` table frees the memory that was used by this table for re-use by any `NDB` table, either by the same table or by another `NDB` table.

Note

Recall that `TRUNCATE TABLE` drops and re-creates the table. See Section 13.1.21, “TRUNCATE TABLE Syntax”.

- **Limits imposed by the cluster's configuration.**
  A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in Section 17.3.3, “MySQL Cluster Configuration Files”. Most configuration parameters can be upgraded online. These hard limits include:

  - Database memory size and index memory size (`DataMemory` and `IndexMemory`, respectively).

    `DataMemory` is allocated as 32KB pages. As each `DataMemory` page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

    See Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”, for further information about `DataMemory` and `IndexMemory`.

  - The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` and `MaxNoOfLocalOperations`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

  - Different limits related to tables and indexes. For example, the maximum number of ordered indexes in the cluster is determined by `MaxNoOfOrderedIndexes`, and the maximum number of ordered inexes per table is 16.

- **Memory usage.** All Cluster table rows are of fixed length. This means (for example) that if a table has one or more `VARCHAR` fields containing only relatively small values, more memory and disk space is required when using the `NDB` storage engine than would be the case for the same table and data using the `MyISAM` engine. (In other words, in the case of a `VARCHAR` column, the column requires the same amount of storage as a `CHAR` column of the same size.)

- **Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:

  - The maximum number of data nodes is 48.

    A data node must have a node ID in the range of 1 to 48, inclusive. (Management and API nodes may use any integer in the range of 1-63 inclusive as a node ID.)

  - The total maximum number of nodes in a MySQL Cluster is 63. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.

  - The maximum number of metadata objects in MySQL 5.0 Cluster is 20320. This limit is hard-coded.

17.1.5.3 Limits Relating to Transaction Handling in MySQL Cluster

A number of limitations exist in MySQL Cluster with regard to the handling of transactions. These include the following:

- **Transaction isolation level.** The `NDBCLUSTER` storage engine supports only the `READ COMMITTED` transaction isolation level. (`InnoDB`, for example, supports `READ COMMITTED`, `READ UNCOMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.) See Section 17.5.3.4, “MySQL Cluster Backup Troubleshooting”, for information on how this can affect backing up and restoring Cluster databases.)

- **Transactions and BLOB or TEXT columns.** `NDBCLUSTER` stores only part of a column value that uses any of MySQL's `BLOB` or `TEXT` data types in the table visible to MySQL; the remainder of the `BLOB` or `TEXT` is stored in a separate internal table that is not accessible to MySQL. This gives rise to two related issues of which you should be aware whenever executing `SELECT` statements on tables that contain columns of these types:

  1. For any `SELECT` from a MySQL Cluster table: If the `SELECT` includes a `BLOB` or `TEXT` column, the `READ COMMITTED` transaction isolation level is converted to a read with read lock. This is done to guarantee consistency.

  2. For any `SELECT` which uses a primary key lookup or unique key lookup to retrieve any columns that use any of the `BLOB` or `TEXT` data types and that is executed within a transaction, a shared read lock is held on the table for the duration of the transaction—that is, until the transaction is either committed or aborted. This does not occur for queries that use index or table scans.

     For example, consider the table `t` defined by the following `CREATE TABLE` statement:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE TABLE t (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    b INT NOT NULL,
    c INT NOT NULL,
    d TEXT,
    INDEX i(b),
    UNIQUE KEY u(c)
) ENGINE = NDB,
```

Either of the following queries on `t` causes a shared read lock, because the first query uses a primary key lookup and the second uses a unique key lookup:

```
SELECT * FROM t WHERE a = 1;

SELECT * FROM t WHERE c = 1;
```

However, none of the four queries shown here causes a shared read lock:

```
SELECT * FROM t WHERE b = 1;

SELECT * FROM t WHERE d = '1';

SELECT * FROM t;

SELECT b,c WHERE a = 1;
```

This is because, of these four queries, the first uses an index scan, the second and third use table scans, and the fourth, while using a primary key lookup, does not retrieve the value of any `BLOB` or `TEXT` columns.

You can help minimize issues with shared read locks by avoiding queries that use primary key lookups or unique key lookups to retrieve `BLOB` or `TEXT` columns, or, in cases where such queries are not avoidable, by committing transactions as soon as possible afterward.

This limitation is lifted in MySQL Cluster NDB 7.0 (Bug #49190).

- **Rollbacks.** There are no partial transactions, and no partial rollbacks of transactions. A duplicate key or similar error aborts the entire transaction, and subsequent statements raise `ERROR 1296 (HY000): Got error 4350 'Transaction already aborted' from NDBCLUSTER`. In such cases, you must issue an explicit `ROLLBACK` and retry the entire transaction.

  This behavior differs from that of other transactional storage engines such as `InnoDB` that may roll back individual statements.

- **Transactions and memory usage.**
  As noted elsewhere in this chapter, MySQL Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behavior of a number of MySQL statements is effected as described in the following list:

  - `TRUNCATE TABLE` is not transactional when used on `NDB` tables. If a `TRUNCATE TABLE` fails to empty the table, then it must be re-run until it is successful.

  - `DELETE FROM` (even with no `WHERE` clause) *is* transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

statements to “chunk” the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE TABLE` instead.

- **LOAD DATA statements.** `LOAD DATA INFILE` is not transactional when used on `NDB` tables.

  Important

  When executing a `LOAD DATA INFILE` statement, the `NDB` engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place.

  `LOAD DATA FROM MASTER` is not supported in MySQL Cluster.

- **ALTER TABLE and transactions.** When copying an `NDB` table as part of an `ALTER TABLE`, the creation of the copy is nontransactional. (In any case, this operation is rolled back when the copy is deleted.)

17.1.5.4 MySQL Cluster Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `Temporary failure, distribution changed` and similar temporary errors.
- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also Section 17.1.5.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”.

17.1.5.5 Limits Associated with Database Objects in MySQL Cluster

Some database objects such as tables and indexes have different limitations when using the `NDBCLUSTER` storage engine:

- **Identifiers.** Database names, table names and attribute names cannot be as long in `NDB` tables as when using other table handlers. Attribute names are truncated to 31 characters, and if not unique after truncation give rise to errors. Database names and table names can total a maximum of 122 characters. In other words, the maximum length for an `NDB` table name is 122 characters, less the number of characters in the name of the database of which that table is a part.
- **Table names containing special characters.** `NDB` tables whose names contain characters other than letters, numbers, dashes, and underscores and which are created on one SQL node may not be discovered correctly by other SQL nodes. (Bug #31470)
- **Number of tables and other database objects.** The maximum number of tables in a Cluster database in MySQL 5.0 is limited to 1792. The maximum number of *all* `NDBCLUSTER` database objects in a single MySQL Cluster—including databases, tables, and indexes—is limited to 20320.
- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) per table is limited to 128.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- **Attributes per key.** The maximum number of attributes per key is 32.
- **Row size.** The maximum permitted size of any one row is 8052 bytes. Each `BLOB` or `TEXT` column contributes 256 + 8 = 264 bytes to this total.
- **BIT column storage per table.** The maximum combined width for all `BIT` columns used in a given `NDB` table is 4096.
- **Number of rows per partition.** The maximum number of rows that can be stored in a single MySQL Cluster partition varies with the number of replicas times the number of fragments. Since the number of partitions is the same as the number of data nodes in the cluster (see Section 17.1.2, "MySQL Cluster Nodes, Node Groups, Replicas, and Partitions"), you can increase the number of fragments—and thus the available space—by using more data nodes.

17.1.5.6 Unsupported or Missing Features in MySQL Cluster

A number of features supported by other storage engines are not supported for `NDB` tables. Trying to use any of these features in MySQL Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced. Statements referencing such features, even if effectively ignored by `NDB`, must be syntactically and otherwise valid.

- **Foreign key constraints.** The foreign key construct is ignored, just as it is in `MyISAM` tables.
- **Index prefixes.** Prefixes on indexes are not supported for `NDB` tables. If a prefix is used as part of an index specification in a statement such as `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`, the prefix is not created by `NDB`.

  A statement containing an index prefix, and creating or modifying an `NDB` table, must still be syntactically valid. For example, the following statement always fails with Error 1089 `Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys`, regardless of storage engine:

  ```
  CREATE TABLE t1 (
      c1 INT NOT NULL,
      c2 VARCHAR(100),
      INDEX i1 (c2(500))
  );
  ```

  This happens on account of the SQL syntax rule that no index may have a prefix larger than itself.
- **OPTIMIZE operations.** `OPTIMIZE` operations are not supported.
- **LOAD TABLE ... FROM MASTER.** `LOAD TABLE ... FROM MASTER` is not supported.
- **Savepoints and rollbacks.** Savepoints and rollbacks to savepoints are ignored as in `MyISAM`.
- **Durability of commits.** There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- **Replication.** Replication is not supported.

Note

See Section 17.1.5.3, "Limits Relating to Transaction Handling in MySQL Cluster", for more information relating to limitations on transaction handling in `NDB`.

17.1.5.7 Limitations Relating to Performance in MySQL Cluster

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The following performance issues are specific to or especially pronounced in MySQL Cluster:

- **Range scans.** There are query performance issues due to sequential access to the `NDB` storage engine; it is also relatively more expensive to do many range scans than it is with either `MyISAM` or `InnoDB`.
- **Reliability of Records in range.** The `Records in range` statistic is available but is not completely tested or officially supported. This may result in nonoptimal query plans in some cases. If necessary, you can employ `USE INDEX` or `FORCE INDEX` to alter the execution plan. See Section 8.9.2, “Index Hints”, for more information on how to do this.
- **Unique hash indexes.** Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.

17.1.5.8 Issues Exclusive to MySQL Cluster

The following are limitations specific to the `NDBCLUSTER` storage engine:

- **Machine architecture.** The following issues relate to physical architecture of cluster hosts:
  - All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running `mysql` or other clients that may be accessing the cluster's SQL nodes.
  - **Adding and dropping of data nodes.** Online adding or dropping of data nodes is not currently possible. In such cases, the entire cluster must be restarted.
  - **Backup and restore between architectures.** It is also not possible to perform a Cluster backup and restore between different architectures. For example, you cannot back up a cluster running on a big-endian platform and then restore from that backup to a cluster running on a little-endian system. (Bug #19255)
- **Online schema changes.** It is not possible to make online schema changes such as those accomplished using `ALTER TABLE` or `CREATE INDEX`, as the `NDB Cluster` engine does not support autodiscovery of such changes. (However, you can import or create a table that uses a different storage engine, and then convert it to `NDB` using `ALTER TABLE tbl_name ENGINE=NDBCLUSTER`. In such a case, you must issue a `FLUSH TABLES` statement to force the cluster to pick up the change.)
- **Binary logging.**
  MySQL Cluster has the following limitations or restrictions with regard to binary logging:
  - `sql_log_bin` has no effect on data operations; however, it is supported for schema operations.
  - MySQL Cluster cannot produce a binary log for tables having `BLOB` columns but no primary key.
  - Only the following schema operations are logged in a cluster binary log which is *not* on the `mysqld` executing the statement:
    - `CREATE TABLE`
    - `ALTER TABLE`
    - `DROP TABLE`
    - `CREATE DATABASE` / `CREATE SCHEMA`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `DROP DATABASE` / `DROP SCHEMA`

See also Section 17.1.5.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

17.1.5.9 Limitations Relating to Multiple MySQL Cluster Nodes

Multiple SQL nodes.
The following are issues relating to the use of multiple MySQL servers as MySQL Cluster SQL nodes, and are specific to the `NDBCLUSTER` storage engine:

- **No distributed table locks.** A `LOCK TABLES` works only for the SQL node on which the lock is issued; no other SQL node in the cluster “sees” this lock. This is also true for a lock issued by any statement that locks tables as part of its operations. (See next item for an example.)

- **ALTER TABLE operations.** `ALTER TABLE` is not fully locking when running multiple MySQL servers (SQL nodes). (As discussed in the previous item, MySQL Cluster does not support distributed table locks.)

- **Replication.** MySQL replication will not work correctly if updates are done on multiple MySQL servers. However, if the database partitioning scheme is done at the application level and no transactions take place across these partitions, replication can be made to work.

- **Database autodiscovery.** Autodiscovery of databases is not supported for multiple MySQL servers accessing the same MySQL Cluster. However, autodiscovery of tables is supported in such cases. What this means is that after a database named *`db_name`* is created or imported using one MySQL server, you should issue a `CREATE DATABASE` *`db_name`* statement on each additional MySQL server that accesses the same MySQL Cluster. (As of MySQL 5.0.2, you may also use `CREATE SCHEMA` *`db_name`*.) Once this has been done for a given MySQL server, that server should be able to detect the database tables without error.

- **DDL operations.** DDL operations (such as `CREATE TABLE` or `ALTER TABLE`) are not safe from data node failures. If a data node fails while trying to perform one of these, the data dictionary is locked and no further DDL statements can be executed without restarting the cluster.

Multiple management nodes.
When using multiple management servers:

- If any of the management servers are running on the same host, you must give nodes explicit IDs in connection strings because automatic allocation of node IDs does not work across multiple management servers on the same host. This is not required if every management server resides on a different host.

  In addition, all API nodes (including MySQL servers acting as SQL nodes), should list all management servers using the same order in their connection strings.

- You must take extreme care to have the same configurations for all management servers. No special checks for this are performed by the cluster.

- Prior to MySQL 5.0.14, all data nodes had to be restarted after bringing up the cluster for the management nodes to be able to see one another.

  (See Bug #12307 and Bug #13070 for more information.)

Multiple data node processes. While it is possible to run multiple cluster processes concurrently on a single host, it is not always advisable to do so for reasons of performance and high availability, as well as other considerations. In particular, in MySQL 5.0, we do not support for production use any MySQL Cluster deployment in which more than one `ndbd` process is run on a single physical machine.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

We may support multiple data nodes per host in a future MySQL release, following additional testing. However, in MySQL 5.0, such configurations can be considered experimental only.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.

Note

It is possible to use multiple network hardware *interfaces* (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it not possible to use more than one `[tcp]` section per connection in the `config.ini` file. See Section 17.3.3.8, "MySQL Cluster TCP/IP Connections", for more information.

17.1.5.10 Previous MySQL Cluster Issues Resolved in MySQL 5.0

The following Cluster limitations in MySQL 4.1 have been resolved in MySQL 5.0 as shown below:

- **Character set support.** The `NDBCLUSTER` storage engine supports all character sets and collations available in MySQL 5.0.
- **Character set directory.** Beginning with MySQL 5.0.21, it is possible to install MySQL with Cluster support to a nondefault location and change the search path for font description files using either the `--basedir` or `--character-sets-dir` options. (Previously, `ndbd` in MySQL 5.0 searched only the default path—typically `/usr/local/mysql/share/mysql/charsets`—for character sets.)
- **Metadata objects.** Prior to MySQL 5.0.6, the maximum number of metadata objects possible was 1600. Beginning with MySQL 5.0.6, this limit is increased to 20320.
- **Query cache.** Unlike the case in MySQL 4.1, the Cluster storage engine in MySQL 5.0 supports MySQL's query cache. See Section 8.10.3, "The MySQL Query Cache".
- **IGNORE and REPLACE functionality.**
  In MySQL 5.0.19 and earlier, `INSERT IGNORE`, `UPDATE IGNORE`, and `REPLACE` were supported only for primary keys, but not for unique keys. It was possible to work around this issue by removing the constraint, then dropping the unique index, performing any inserts, and then adding the unique index again.

  This limitation was removed for `INSERT IGNORE` and `REPLACE` in MySQL 5.0.20. (See Bug #17431.)
- **auto_increment_increment and auto_increment_offset.** The `auto_increment_increment` and `auto_increment_offset` server system variables are supported for `NDBCLUSTER` tables beginning with MySQL 5.0.46.

17.2 MySQL Cluster Installation and Upgrades

This section describes the basics for planning, installing, configuring, and running a MySQL Cluster. Whereas the examples in Section 17.3, "MySQL Cluster Configuration" provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable MySQL Cluster which meets the *minimum* requirements for availability and safeguarding of data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This section covers hardware and software requirements; networking issues; installation of MySQL Cluster; configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

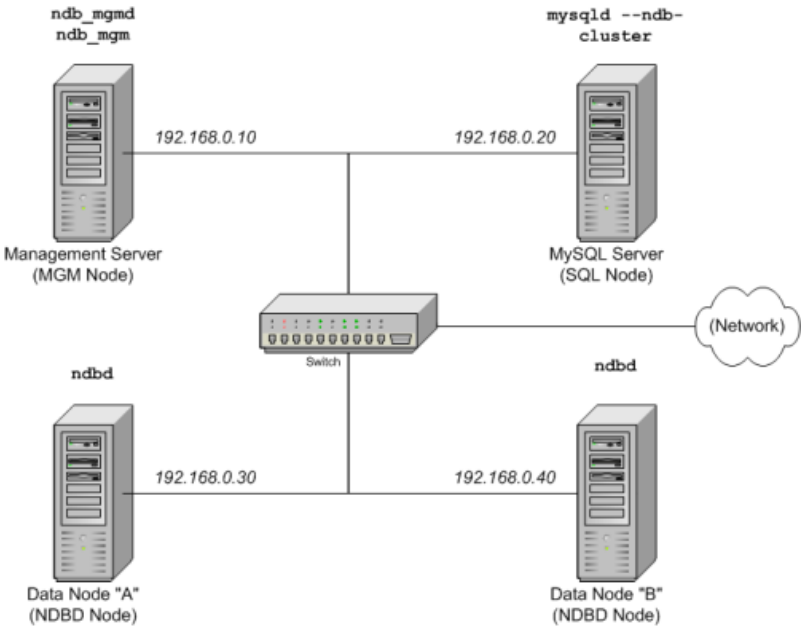
Assumptions. The following sections make a number of assumptions regarding the cluster's physical and network configuration. These assumptions are discussed in the next few paragraphs.

Cluster nodes and host computers. The cluster consists of four nodes, each on a separate host computer, and each with a fixed network address on a typical Ethernet network as shown here:

Node	IP Address
Management node (`mgmd`)	192.168.0.10
SQL node (`mysqld`)	192.168.0.20
Data node "A" (`ndbd`)	192.168.0.30
Data node "B" (`ndbd`)	192.168.0.40

This may be made clearer by the following diagram:

Figure 17.4 MySQL Cluster Multi-Computer Setup



Network addressing. In the interest of simplicity (and reliability), this *How-To* uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `hosts` file (typically `/etc/hosts` for Linux and other Unix-like operating systems, or your operating system's equivalent) for providing a means to do host lookup if such is available.

Potential hosts file issues. A common problem when trying to use host names for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own host name in the `/etc/hosts` during installation. Consider two machines with the host names `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
127.0.0.1   ndb1.cluster ndb1 localhost.localdomain localhost
```

```
#  ndb2 /etc/hosts:
127.0.0.1   ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (including OpenSUSE) places these entries in the machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb1.cluster ndb1
```

```
#  ndb2 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb2.cluster ndb2
```

In both instances, `ndb1` routes `ndb1.cluster` to a loopback IP address, but gets a public IP address from DNS for `ndb2.cluster`, while `ndb2` routes `ndb2.cluster` to a loopback address and obtains a public address for `ndb1.cluster`. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.

Caution

You cannot mix `localhost` and other host names or IP addresses in `config.ini`. For these reasons, the solution in such cases (other than to use IP addresses for *all* `config.ini` `HostName` entries) is to remove the fully qualified host names from `/etc/hosts` and use these in `config.ini` for all cluster hosts.

Host computer type. Each host computer in our installation scenario is an Intel-based desktop PC running a supported operating system installed to disk in a standard configuration, and running no unnecessary services. The core operating system with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the file systems on all hosts are set up identically. In the event that they are not, you should adapt these instructions accordingly.

Network hardware. Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected through a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) MySQL Cluster works in a 100 Mbps network; however, gigabit Ethernet provides better performance.

Important

MySQL Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps or which experiences a high degree of latency. For this reason (among others), attempting to run a MySQL Cluster over a wide area network such as the Internet is not likely to be successful, and is not supported in production.

Sample data. We use the `world` database which is available for download from the MySQL Web site (see http://dev.mysql.com/doc/index-other.html). We assume that each machine has sufficient memory for running the operating system, required MySQL Cluster processes, and (on the data nodes) storing the database.

For general information about installing MySQL, see Chapter 2, *Installing and Upgrading MySQL*. For information about installation of MySQL Cluster on Linux and other Unix-like operating systems, see Section 17.2.1, “Installing MySQL Cluster on Linux”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For general information about MySQL Cluster hardware, software, and networking requirements, see Section 17.1.3, “MySQL Cluster Hardware, Software, and Networking Requirements”.

17.2.1 Installing MySQL Cluster on Linux

This section covers installation of MySQL Cluster on Linux and other Unix-like operating systems. While the next few sections refer to a Linux operating system, the instructions and procedures given there should be easily adaptable to other supported Unix-like platforms.

Each MySQL Cluster host computer must have the correct executable programs installed. A host running an SQL node must have installed on it a MySQL Server binary (`mysqld`). Management nodes require the management server daemon (`ndb_mgmd`); data nodes require the data node daemon (`ndbd`. It is not necessary to install the MySQL Server binary on management node hosts and data node hosts. It is recommended that you also install the management client (`ndb_mgm`) on the management server host.

Installation of MySQL Cluster on Linux can be done using precompiled binaries from Oracle (downloaded as a .tar.gz archive), with RPM packages (also available from Oracle), or from source code. All three of these installation methods are described in the section that follow.

Regardless of the method used, it is still necessary following installation of the MySQL Cluster binaries to create configuration files for all cluster nodes, before you can start the cluster. See Section 17.2.2, “Initial Configuration of MySQL Cluster”.

17.2.1.1 Installing a MySQL Cluster Binary Release on Linux

This section covers the steps necessary to install the correct executables for each type of Cluster node from precompiled binaries supplied by Oracle.

For setting up a cluster using precompiled binaries, the first step in the installation process for each cluster host is to download the latest MySQL 5.0 binary archive from the MySQL downloads page. We assume that you have placed this file in each machine's `/var/tmp` directory. (If you do require a custom binary, see Section 2.17.2, “Installing MySQL Using a Development Source Tree”.)

Note

After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of the nodes (see Section 17.2.2, “Initial Configuration of MySQL Cluster”).

SQL nodes. On each of the machines designated to host SQL nodes, perform the following steps as the system `root` user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql -s /bin/false mysql
   ```

   The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symbolic link named `mysql` to the `mysql` directory. Note that the actual file and directory names vary according to the MySQL Cluster version number.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> cd /var/tmp
shell> tar -C /usr/local -xzvf mysql-5.0.96-linux-i686-glibc23.tar.gz
shell> ln -s /usr/local/mysql-5.0.96-linux-i686-glibc23 /usr/local/mysql
```

3. Change location to the `mysql` directory and run the supplied script for creating the system databases:

```
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
```

4. Set the necessary permissions for the MySQL server and data directories:

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

```
shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

   (The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is `/etc/init.d`.)

   Here we use Red Hat's `chkconfig` for creating links to the startup scripts; use whatever means is appropriate for this purpose on your platform, such as `update-rc.d` on Debian.

Remember that the preceding steps must be repeated on each machine where an SQL node is to reside.

Data nodes. Installation of the data nodes does not require the `mysqld` binary. Only the MySQL Cluster data node executable `ndbd` is required. This binary can also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root` (that is, after using `sudo`, `su root`, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install the data node binaries on the data node hosts:

1. Change location to the `/var/tmp` directory, and extract the `ndbd` binary from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-5.0.96-linux-i686-glibc23.tar.gz
shell> cd mysql-5.0.96-linux-i686-glibc23
shell> cp bin/ndbd /usr/local/bin/ndbd
```

   (You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` has been copied to the executables directory.)

2. Change location to the directory into which you copied the binary, and make it executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndbd
```

The preceding steps should be repeated on each data node host.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

The data directory on each machine hosting a data node is `/usr/local/mysql/data`. This piece of information is essential when configuring the management node. (See Section 17.2.2, "Initial Configuration of MySQL Cluster".)

Management nodes. Installation of the management node does not require the `mysqld` binary. Only the MySQL Cluster management server (`ndb_mgmd`) is required; you most likely want to install the management client (`ndb_mgm`) as well. Both of these binaries also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root`, perform the following steps to install `ndb_mgmd` and `ndb_mgm` on the management node host:

1. Change location to the `/var/tmp` directory, and extract the `ndb_mgm` and `ndb_mgmd` from the archive into a suitable directory such as `/usr/local/bin`:

   ```
   shell> cd /var/tmp
   shell> tar -zxvf mysql-5.0.96-linux-i686-glibc23.tar.gz
   shell> cd mysql-5.0.96-linux-i686-glibc23
   shell> cp bin/ndb_mgm* /usr/local/bin
   ```

   (You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

   ```
   shell> cd /usr/local/bin
   shell> chmod +x ndb_mgm*
   ```

In Section 17.2.2, "Initial Configuration of MySQL Cluster", we create configuration files for all of the nodes in our example MySQL Cluster.

17.2.1.2 Installing MySQL Cluster from RPM

This section covers the steps necessary to install the correct executables for each type of MySQL Cluster node using RPM packages supplied by Oracle.

RPMs are available for both 32-bit and 64-bit Linux platforms. For a MySQL Cluster, three RPMs are required:

- The **Server** RPM (for example, `MySQL-server-5.0.96-0.glibc23.i386.rpm`), which supplies the core files needed to run a MySQL Server.

  If you do not have your own client application capable of administering a MySQL server, you should also obtain and install the **Client** RPM (such as `MySQL-client-5.0.96-0.sles10.i586.rpm`).

- The **NDB Cluster - Storage engine** RPM (for example, `MySQL-ndb-storage-5.0.96-0.glibc23.i386.rpm`), which supplies the MySQL Cluster data node binary (`ndbd`).

- The **NDB Cluster - Storage engine management RPM** (for example, `MySQL-ndb-management-5.0.96-0.glibc23.i386.rpm`), which provides the MySQL Cluster management server binary (`ndb_mgmd`).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In addition, you should also obtain the **NDB Cluster - Storage engine basic tools** RPM (for example, `MySQL-ndb-tools-5.0.96-0.glibc23.i386.rpm`), which supplies several useful applications for working with a MySQL Cluster. The most important of these is the MySQL Cluster management client (`ndb_mgm`). The **NDB Cluster - Storage engine extra tools** RPM (for example, `MySQL-ndb-extra-5.0.96-0.glibc23.i386.rpm`) contains some additional testing and monitoring programs, but is not required to install a MySQL Cluster. (For more information about these additional programs, see Section 17.4, "MySQL Cluster Programs".)

The MySQL version number in the RPM file names (shown here as `5.0.96`) can vary according to the version which you are actually using. *It is very important that all of the Cluster RPMs to be installed have the same MySQL version number*. The `glibc` version number (if present—shown here as `glibc23`), and architecture designation (shown here as `i386`) should be appropriate to the machine on which the RPM is to be installed.

Data nodes. On a computer that is to host a cluster data node it is necessary to install only the **NDB Cluster - Storage engine** RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL web site:

```
shell> rpm -Uhv MySQL-ndb-storage-5.0.96-0.glibc23.i386.rpm
```

The previous command installs the MySQL Cluster data node binary (`ndbd`) in the `/usr/sbin` directory.

SQL nodes. On each machine to be used for hosting a cluster SQL node, install the **Server** RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL web site:

```
shell> rpm -Uhv MySQL-server-5.0.96-0.glibc23.i386.rpm
```

This installs the MySQL server binary (`mysqld`) in the `/usr/sbin` directory, as well as all needed MySQL Server support files. It also installs the `mysql.server` and `mysqld_safe` startup scripts in `/usr/share/mysql` and `/usr/bin`, respectively. The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

Note

To administer the SQL node (MySQL server), you should also install the **Client** RPM, as shown here:

```
shell> rpm -Uhv MySQL-client-5.0.96-0.sles10.i586.rpm
```

This installs the `mysql` client program.

Management nodes. To install the MySQL Cluster management server, it is necessary only to use the **NDB Cluster - Storage engine management** RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the **Storage engine management** RPM downloaded from the MySQL web site):

```
shell> rpm -Uhv MySQL-ndb-management-5.0.96-0.glibc23.i386.rpm
```

This installs the management server binary (`ndb_mgmd`) to the `/usr/sbin` directory.

You should also install the `NDB` management client, which is supplied by the **Storage engine basic tools** RPM. Copy this RPM to the same computer as the management node, and then install it by running the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

following command as the system root user (again, replace the name shown for the RPM as necessary to match that of the **Storage engine basic tools** RPM downloaded from the MySQL web site):

```
shell> rpm -Uhv MySQL-ndb-tools-5.0.96-0.sles10.i586.rpm
```

The **Storage engine basic tools** RPM installs the MySQL Cluster management client (`ndb_mgm`) to the `/usr/bin` directory.

Note

You can also install the **Cluster storage engine extra tools** RPM, if you wish, as shown here:

```
shell> rpm -Uhv MySQL-ndb-extra-5.0.96-0.sles10.i586.rpm
```

You may find the extra tools useful; however the **Cluster storage engine extra tools** RPM is *not* required to install a working MySQL Cluster.

See Section 2.12, “Installing MySQL on Linux Using RPM Packages”, for general information about installing MySQL using RPMs supplied by Oracle.

After installing from RPM, you still need to configure the cluster as discussed in Section 17.2.2, “Initial Configuration of MySQL Cluster”.

17.2.1.3 Building MySQL Cluster from Source on Linux

This section provides information about compiling MySQL Cluster on Linux and other Unix-like platforms.To build MySQL Cluster, you need the MySQL 5.0 source archive available from http://dev.mysql.com/downloads/. Building MySQL Cluster from source is similar to building the standard MySQL Server, although it differs in a few key respects discussed here. For general information about building MySQL from source, see Section 2.17, “Installing MySQL from Source”.

In addition to any other `configure` options you wish to use, be sure to include `--with-ndbcluster`. This option causes the binaries for the management nodes, data nodes, and other MySQL Cluster programs to be built; it also causes `mysqld` to be compiled with `NDB` storage engine support.

After you have run `make && make install` (or your system's equivalent), the result is similar to what is obtained by unpacking a precompiled binary to the same location. However, the layout can differ. These differences are covered in the next few paragraphs.

Management nodes. When building from source and running the default `make install`, the management server binary (`ndb_mgmd`) is placed in `/usr/local/mysql/libexec`, while the management client binary (`ndb_mgm`) can be found in `/usr/local/mysql/bin`. Only `ndb_mgmd` is required to be present on a management node host; however, it is also a good idea to have `ndb_mgm` present on the same host machine. Neither of these executables requires a specific location on the host machine's file system.

Data nodes. The only executable required on a data node host is `ndbd` (`mysqld`, for example, does not have to be present on the host machine). By default when doing a source build, this file is placed in the directory `/usr/local/mysql/libexec`. For installing on multiple data node hosts, only `ndbd` need be copied to the other host machine or machines. (This assumes that all data node hosts use the same architecture and operating system; otherwise you may need to compile separately for each different platform.) `ndbd` need not be in any particular location on the host's file system, as long as the location is known.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SQL nodes. If you compile MySQL with clustering support, and perform the default installation (using `make install` as the system `root` user), `mysqld` is placed in `/usr/local/mysql/bin`. Follow the steps given in Section 2.17, “Installing MySQL from Source” to make `mysqld` ready for use. If you want to run multiple SQL nodes, you can use a copy of the same `mysqld` executable and its associated support files on several machines. The easiest way to do this is to copy the entire `/usr/local/mysql` directory and all directories and files contained within it to the other SQL node host or hosts, then repeat the steps from Section 2.17, “Installing MySQL from Source” on each machine. If you configure the build with a nondefault `--prefix`, you need to adjust the directory accordingly.

In Section 17.2.2, “Initial Configuration of MySQL Cluster”, we create configuration files for all of the nodes in our example MySQL Cluster.

17.2.2 Initial Configuration of MySQL Cluster

For our four-node, four-host MySQL Cluster, it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a *connection string* that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the `NDBCLUSTER` storage engine.

  For more information on connection strings, see Section 17.3.3.2, “MySQL Cluster Connection Strings”.

- The management node needs a `config.ini` file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the data nodes and SQL nodes. The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

Note

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine

[mysql_cluster]
# Options for MySQL Cluster processes:
ndb-connectstring=192.168.0.10  # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a `mysqld` process with the `ndbcluster` and `ndb-connectstring` parameters in the `[mysqld]` and `[mysql_cluster]` sections of the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. This is by design.

Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2    # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=18M   # How much memory to allocate for index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
                  # this example Cluster setup.

[tcp default]
# TCP/IP options:
portnumber=2202   # This the default; however, you can use any
                  # port that is free for all the hosts in the cluster
                  # Note: It is recommended that you do not specify the port
                  # number at all and simply allow the default value to be used
                  # instead

[ndb_mgmd]
# Management process options:
hostname=192.168.0.10           # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster  # Directory for MGM node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
hostname=192.168.0.30           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data files

[ndbd]
# Options for data node "B":
hostname=192.168.0.40           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data files

[mysqld]
# SQL node options:
hostname=192.168.0.20           # Hostname or IP address
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

Note

The `world` database can be downloaded from http://dev.mysql.com/doc/index-other.html.

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in Section 17.2.3, “Initial Startup of MySQL Cluster”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For more detailed information about the available MySQL Cluster configuration parameters and their uses, see Section 17.3.3, “MySQL Cluster Configuration Files”, and Section 17.3, “MySQL Cluster Configuration”. For configuration of MySQL Cluster as relates to making backups, see Section 17.5.3.3, “Configuration for MySQL Cluster Backups”.

Note

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. Beginning with MySQL 5.0.3, this restriction is lifted, and the cluster automatically allocates ports for data nodes from those that are already free.

17.2.3 Initial Startup of MySQL Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

Note

`ndb_mgmd` must be told where to find its configuration file, using the `-f` or `--config-file` option. (See Section 17.4.2, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”, for details.)

For additional options which can be used with `ndb_mgmd`, see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

2. On each of the data node hosts, run this command to start the `ndbd` process:

```
shell> ndbd
```

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the `ndb_mgm` management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @192.168.0.30  (Version: 5.0.96, Nodegroup: 0, Master)
id=3    @192.168.0.40  (Version: 5.0.96, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
id=1    @192.168.0.10  (Version: 5.0.96)

[mysqld(API)]   1 node(s)
id=4    @192.168.0.20  (Version: 5.0.96)
```

The SQL node is referenced here as `[mysqld(API)]`, which reflects the fact that the `mysqld` process is acting as a MySQL Cluster API node.

Note

The IP address shown for a given MySQL Cluster SQL or other API node in the output of `SHOW` is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.

You should now be ready to work with databases, tables, and data in MySQL Cluster. See Section 17.2.4, "MySQL Cluster Example with Tables and Data", for a brief discussion.

17.2.4 MySQL Cluster Example with Tables and Data

Working with database tables and data in MySQL Cluster is not much different from doing so in standard MySQL. There are two key points to keep in mind:

- For a table to be replicated in the cluster, it must use the `NDBCLUSTER` storage engine. To specify this, use the `ENGINE=NDBCLUSTER` or `ENGINE=NDB` option when creating the table:

  ```
  CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
  ```

  Alternatively, for an existing table that uses a different storage engine, use `ALTER TABLE` to change the table to use `NDBCLUSTER`:

  ```
  ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
  ```

- Every `NDBCLUSTER` table has a primary key. If no primary key is defined by the user when a table is created, the `NDBCLUSTER` storage engine automatically generates a hidden one. Such a key takes up space just as does any other table index. (It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of `mysqldump`, you can open the SQL script in a text editor and add the `ENGINE` option to any table creation statements, or replace any existing `ENGINE` (or `TYPE`) options. Suppose that you have the `world` sample database on another MySQL server that does not support MySQL Cluster, and you want to export the `City` table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file will contain this table creation statement (and the `INSERT` statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the `NDBCLUSTER` storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the `City` table as an example, modify the `ENGINE` option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of `TYPE=`*`engine_name`* or `ENGINE=`*`engine_name`* with `ENGINE=NDBCLUSTER`. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use `ALTER TABLE` to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named `world` on the SQL node of the cluster, you can then use the `mysql` command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address `192.168.0.20`).

To create a copy of the entire `world` database on the SQL node, use `mysqldump` on the noncluster server to export the database to a file named `world.sql`; for example, in the `/tmp` directory. Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

Note

`NDBCLUSTER` in MySQL 5.0 does not support autodiscovery of databases. (See Section 17.1.5, “Known Limitations of MySQL Cluster”.) This means that, once the `world` database and its tables have been created on one data node, you need to issue the `CREATE DATABASE world` statement (beginning with MySQL 5.0.2, you may use `CREATE SCHEMA world` instead), followed by `FLUSH TABLES` on each

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SQL node in the cluster. This causes the node to recognize the database and read its table definitions.

Running `SELECT` queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the `root` password at the `Enter password:` prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.96

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We simply use the MySQL server's `root` account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong `root` password. For more information, see Section 2.18.4, “Securing the Initial MySQL Accounts”.

It is worth taking into account that Cluster nodes do *not* make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the `root` account) effects only applications that access the SQL node, not interaction between nodes. See Section 17.5.10.2, “MySQL Cluster and MySQL Privileges”, for more information.

If you did not modify the `ENGINE` clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a `SELECT` query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Bombay    |   10500000 |
| Seoul     |    9981619 |
| São Paulo |    9968485 |
| Shanghai  |    9696300 |
| Jakarta   |    9604900 |
+-----------+------------+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access `NDB` tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the `SELECT` statement just shown by using the PHP 5.X `mysqli` extension running on a Web server elsewhere on the network:

```
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
          content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      printf("<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
              $row->Name, $row->Population);
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
    printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();

  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

17.2.5 Safe Shutdown and Restart of MySQL Cluster

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the `ndb_mgm` client from the shell. (See Section 17.4.20, "Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs", for more information about this option.) The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqladmin shutdown` and other means.

To restart the cluster, run these commands:

- On the management host (`192.168.0.10` in our example setup):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- On each of the data node hosts (`192.168.0.30` and `192.168.0.40`):

```
shell> ndbd
```

- Use the `ndb_mgm` client to verify that both data nodes have started successfully.
- On the SQL host (`192.168.0.20`):

```
shell> mysqld_safe &
```

In a production setting, it is usually not desirable to shut down the cluster completely. In many cases, even when making configuration changes, or performing upgrades to the cluster hardware or software (or both), which require shutting down individual host machines, it is possible to do so without shutting down the cluster as a whole by performing a *rolling restart* of the cluster. For more information about doing this, see Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster".

17.2.6 Upgrading and Downgrading MySQL Cluster

This section provides information about MySQL Cluster software and table file compatibility between MySQL 5.0 releases with regard to performing upgrades and downgrades as well as a compatibility matrix and notes. You are expected already to be familiar with installing and configuring a MySQL Cluster prior to attempting an upgrade or downgrade. See Section 17.3, "MySQL Cluster Configuration".

For information regarding the rolling restart procedure used to perform an online upgrade, see Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster".

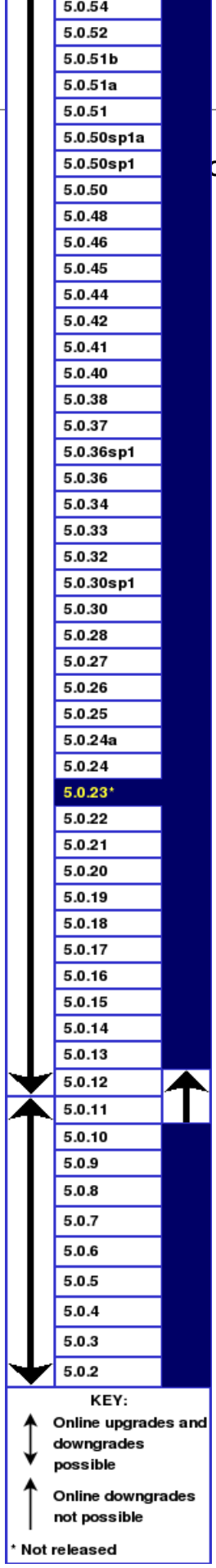
Important

Only compatibility between MySQL versions with regard to `NDBCLUSTER` is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the MySQL Cluster software*. See Section 2.19.1, "Upgrading MySQL".

The following table shows Cluster upgrade and downgrade compatibility between different releases of MySQL 5.0:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Cluster Upgrade and Downgrade Compatibility



This documentation is for an older version. If you're

Notes

- MySQL 5.0.2 was the first public release in this series.
- Direct upgrades or downgrades between MySQL Cluster 4.1 and 5.0 are not supported; you must dump all `NDBCLUSTER` tables using `mysqldump`, install the new version of the software, and then reload the tables from the dump.
- Online downgrades from MySQL Cluster 5.0.12 to 5.0.11 (or earlier) are not supported.
- You cannot restore with `ndb_restore` to a MySQL 5.0 Cluster using a backup made from a Cluster running MySQL 5.1. You must use `mysqldump` in such cases.
- There was no public release of MySQL 5.0.23.

17.3 MySQL Cluster Configuration

A MySQL server that is part of a MySQL Cluster differs in one chief respect from a normal (nonclustered) MySQL server, in that it employs the `NDB` storage engine. This engine is also referred to sometimes as `NDBCLUSTER`, although `NDB` is preferred.

To avoid unnecessary allocation of resources, the server is configured by default with the `NDB` storage engine disabled. To enable `NDB`, you must modify the server's `my.cnf` configuration file, or start the server with the `--ndbcluster` option.

This MySQL server is a part of the cluster, so it also must know how to access a management node to obtain the cluster configuration data. The default behavior is to look for the management node on `localhost`. However, should you need to specify that its location is elsewhere, this can be done in `my.cnf`, or with the `mysql` client. Before the `NDB` storage engine can be used, at least one management node must be operational, as well as any desired data nodes.

For more information about `--ndbcluster` and other `mysqld` options specific to MySQL Cluster, see mysqld Command Options for MySQL Cluster.

For information about installing MySQL Cluster, see Section 17.2, “MySQL Cluster Installation and Upgrades”.

17.3.1 Quick Test Setup of MySQL Cluster

To familiarize you with the basics, we will describe the simplest possible configuration for a functional MySQL Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system `root` user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `HostName` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 SQL nodes.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but SQL nodes are not required
# to do so.
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
# If you do not know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld  default]
[ndb_mgmd default]
[tcp default]

[ndb_mgmd]
HostName= myhost.example.com

[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[mysqld]
[mysqld]
[mysqld]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running `ndbd`:

```
shell> ndbd
```

For command-line options which can be used when starting `ndbd`, see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

By default, `ndbd` looks for the management server at `localhost` on port 1186.

Note

If you have installed MySQL from a binary tarball, you will need to specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the NDB storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.96

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an NDB table:

```
shell> mysql
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
*************************** 1. row ***************************
       Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the SHOW command from within the management client to obtain a report on the cluster's status:

```
ndb_mgm> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     1 node(s)
id=2    @127.0.0.1  (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @127.0.0.1  (Version: 3.5.3)

[mysqld(API)]   3 node(s)
id=3    @127.0.0.1  (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

At this point, you have successfully set up a working MySQL Cluster. You can now store data in the cluster by using any table created with `ENGINE=NDBCLUSTER` or its alias `ENGINE=NDB`.

17.3.2 Overview of MySQL Cluster Configuration Parameters, Options, and Variables

The next several sections provide summary tables of MySQL Cluster node configuration parameters used in the `config.ini` file to govern various aspects of node behavior, as well as of options and variables read by `mysqld` from a `my.cnf` file or from the command line when run as a MySQL Cluster process. Each of the node parameter tables lists the parameters for a given type (`ndbd`, `ndb_mgmd`, `mysqld`, `computer`, `tcp`, `shm`, or `sci`). All tables include the data type for the parameter, option, or variable, as well as its default, mimimum, and maximum values as applicable.

Considerations when restarting nodes. For node parameters, these tables also indicate what type of restart is required (node restart or system restart)—and whether the restart must be done with `--initial`—to change the value of a given configuration parameter. When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked as `node` online—that is, without shutting down the cluster—in this fashion. An initial node restart requires restarting each `ndbd` process with the `--initial` option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster file system after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted for them to read the updated configuration parameter values.

Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. Many of these can be increased online, using a rolling restart.

However, decreasing the values of such parameters—whether this is done using a node restart, node initial restart, or even a complete system restart of the cluster—is not to be undertaken lightly; it is recommended that you do so only after careful planning and testing. This is especially true with regard to those parameters that relate to memory usage and disk space, such as `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes`. In addition, it is the generally the case that configuration parameters relating to memory and disk usage can be raised using a simple node restart, but they require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

Note

`4294967039` often appears as a maximum value in these tables. This value is defined in the `NDBCLUSTER` sources as `MAX_INT_RNIL` and is equal to `0xFFFFFEFF`, or $2^{32} - 2^8 - 1$.

17.3.2.1 MySQL Cluster Data Node Configuration Parameters

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The summary table in this section provides information about parameters used in the `[ndbd]` or `[ndbd default]` sections of a `config.ini` file for configuring MySQL Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see Section 17.3.3.5, "Defining MySQL Cluster Data Nodes".

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- `N`—Node restart: The parameter can be updated using a rolling restart (see Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster").
- `S`—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- `I`—Initial restart: Data nodes must be restarted using the `--initial` option.

For more information about restart types, see Section 17.3.2, "Overview of MySQL Cluster Configuration Parameters, Options, and Variables".

Table 17.1 Data Node Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
`ArbitrationTimeout`	milliseconds 7500 10 / 4294967039 (0xFFFFFEFF)	N	all
`BackupDataBufferSize`	bytes 16M 512K / 4294967039 (0xFFFFFEFF)	N	all
`BackupDataDir`	path FileSystemPath ...	IN	all
`BackupLogBufferSize`	bytes 16M 2M / 4294967039 (0xFFFFFEFF)	N	all
`BackupMaxWriteSize`	bytes 1M	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	256K / 4294967039 (0xFFFFFEFF)		
BackupMemory	bytes 32M 0 / 4294967039 (0xFFFFFEFF)	N	all
BackupWriteSize	bytes 256K 32K / 4294967039 (0xFFFFFEFF)	N	all
BatchSizePerLocalScan	integer 256 1 / 992	N	all
DataDir	path	IN	all
DataMemory	bytes 80M 1M / 1024G	N	all
Diskless	true|false (1|0) false true, false	IS	all
ExecuteOnComputer	name [none] ...	S	all
FileSystemPath	path DataDir ...	IN	NDB 5.0.0
HeartbeatIntervalDbApi	milliseconds 1500 100 / 4294967039 (0xFFFFFEFF)	N	all
HeartbeatIntervalDbDb	milliseconds	N	all

This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	5000 10 / 4294967039 (0xFFFFFEFF)		
HostName	name or IP address localhost ...	N	all
Id	unsigned [none] 1 / 48	IS	all
IndexMemory	bytes 18M 1M / 1T	N	all
LockPagesInMainMemory	numeric 0 0 / 2	N	NDB 5.0.36
LogLevelCheckpoint	log level 0 0 / 15	N	all
LogLevelCongestion	levelr 0 0 / 15	N	NDB 5.0.0
LogLevelConnection	integer 0 0 / 15	N	all
LogLevelError	integer 0 0 / 15	N	all
LogLevelInfo	integer 0 0 / 15	N	all
LogLevelNodeRestart	integer 0	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	0 / 15		
LogLevelShutdown	integer 0 0 / 15	N	all
LogLevelStartup	integer 1 0 / 15	N	all
LogLevelStatistic	integer 0 0 / 15	N	all
LongMessageBuffer	bytes 64M 512K / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfAttributes	integer 1000 32 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfConcurrentIndexOperations	integer 8K 0 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfConcurrentOperations	integer 32K 32 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfConcurrentScans	integer 256 2 / 500	N	all
MaxNoOfConcurrentTransactions	integer 4096	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	32 / 4294967039 (0xFFFFFEFF)		
MaxNoOfFiredTriggers	integer 4000 0 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfLocalOperations	integer UNDEFINED 32 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfLocalScans	integer [see text] 32 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfOpenFiles	unsigned 0 20 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfOrderedIndexes	integer 128 0 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfSavedMessages	integer 25 0 / 4294967039 (0xFFFFFEFF)	N	all
MaxNoOfTables	integer 128 8 / 20320	N	NDB 5.0.0
MaxNoOfTriggers	integer 768 0 / 4294967039 (0xFFFFFEFF)	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
MaxNoOfUniqueHashIndexes	integer 64 0 / 4294967039 (0xFFFFFEFF)	N	all
NodeId	unsigned [none] 1 / 48	IS	NDB 5.0.15
NoOfDiskPagesToDiskAfterRestartACC	8K pages/100 milliseconds 20 1 / 4294967039 (0xFFFFFEFF)	N	all
NoOfDiskPagesToDiskAfterRestartTUP	8K pages/100 milliseconds 40 1 / 4294967039 (0xFFFFFEFF)	N	all
NoOfDiskPagesToDiskDuringRestartACC	8K pages/100 milliseconds 20 1 / 4294967039 (0xFFFFFEFF)	N	all
NoOfDiskPagesToDiskDuringRestartTUP	8K pages/100 milliseconds 40 1 / 4294967039 (0xFFFFFEFF)	N	all
NoOfFragmentLogFiles	integer 16 3 / 4294967039 (0xFFFFFEFF)	IN	all
NoOfReplicas	integer 2 1 / 4	IS	all
RedoBuffer	bytes 32M	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	1M / 4294967039 (0xFFFFFEFF)		
RestartOnErrorInsert	error code 2 0 / 4	N	all
ServerPort	unsigned [none] 1 / 64K	S	all
StartFailureTimeout	milliseconds 0 0 / 4294967039 (0xFFFFFEFF)	N	all
StartPartialTimeout	milliseconds 30000 0 / 4294967039 (0xFFFFFEFF)	N	all
StartPartitionedTimeout	milliseconds 60000 0 / 4294967039 (0xFFFFFEFF)	N	all
StopOnError	boolean 1 0, 1	N	all
StringMemory	% or bytes 25 0 / 4294967039 (0xFFFFFEFF)	S	all
TimeBetweenGlobalCheckpoints	milliseconds 2000 20 / 32000	N	all
TimeBetweenInactiveTransactionAbortCheck	milliseconds 1000 1000 / 4294967039 (0xFFFFFEFF)	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
TimeBetweenLocalCheckpoints	number of 4-byte words, as a base-2 logarithm 20 0 / 31	N	all
TimeBetweenWatchDogCheck	milliseconds 6000 70 / 4294967039 (0xFFFFFEFF)	N	all
TransactionBufferMemory	bytes 1M 1K / 4294967039 (0xFFFFFEFF)	N	all
TransactionDeadlockDetectionTimeout	milliseconds 1200 50 / 4294967039 (0xFFFFFEFF)	N	all
TransactionInactiveTimeout	milliseconds [see text] 0 / 4294967039 (0xFFFFFEFF)	N	all
UndoDataBuffer	unsigned 16M 1M / 4294967039 (0xFFFFFEFF)	N	all
UndoIndexBuffer	unsigned 2M 1M / 4294967039 (0xFFFFFEFF)	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

To add new data nodes to a MySQL Cluster, it is necessary to shut down the cluster completely, update the `config.ini` file, and then restart the cluster, starting all data node processes using the `--initial` option—that is, you must perform a system restart.

It is possible to add new data node groups to a running cluster online using MySQL Cluster NDB 7.0 or later (see Adding MySQL Cluster Data Nodes Online); however, we do not plan to implement this change in MySQL 5.0.

17.3.2.2 MySQL Cluster Management Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[ndb_mgmd]` or `[mgm]` sections of a `config.ini` file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see Section 17.3.3.4, "Defining a MySQL Cluster Management Server".

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- `N`—Node restart: The parameter can be updated using a rolling restart (see Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster").
- `S`—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- `I`—Initial restart: Data nodes must be restarted using the `--initial` option.

For more information about restart types, see Section 17.3.2, "Overview of MySQL Cluster Configuration Parameters, Options, and Variables".

Table 17.2 Management Node Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
`ArbitrationDelay`	milliseconds 0 0 / 4294967039 (0xFFFFFEFF)	N	all
`ArbitrationRank`	0-2 1 0 / 2	N	all
`DataDir`	path	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
ExecuteOnComputer	name [none] ...	S	all
HostName	name or IP address [none] ...	N	all
Id	unsigned [none] 1 / 255	IS	all
LogDestination	{CONSOLE|SYSLOG|FILE} [see text] ...	N	all
MaxNoOfSavedEvents	unsigned 100 0 / 4294967039 (0xFFFFFEFF)	N	all
NodeId	unsigned [none] 1 / 63	IS	NDB 5.0.15
PortNumber	unsigned 1186 0 / 64K	S	all
PortNumberStats	unsigned [none] 0 / 64K	N	all

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect. See Section 17.3.3.4, “Defining a MySQL Cluster Management Server”, for more information.

To add new management servers to a running MySQL Cluster, it is also necessary perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple

This documentation is for an older version. If you're
This documentation is for an older version. If you're

management nodes, see Section 17.1.5.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

17.3.2.3 MySQL Cluster SQL Node and API Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[mysqld]` and `[api]` sections of a `config.ini` file for configuring MySQL Cluster SQL nodes and API nodes. For detailed descriptions and other additional information about each of these parameters, see Section 17.3.3.6, “Defining SQL and Other API Nodes in a MySQL Cluster”.

Note

For a discussion of MySQL server options for MySQL Cluster, see mysqld Command Options for MySQL Cluster; for information about MySQL server system variables relating to MySQL Cluster, see MySQL Cluster System Variables.

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- `N`—Node restart: The parameter can be updated using a rolling restart (see Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”).
- `S`—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- `I`—Initial restart: Data nodes must be restarted using the `--initial` option.

For more information about restart types, see Section 17.3.2, “Overview of MySQL Cluster Configuration Parameters, Options, and Variables”.

Table 17.3 SQL Node / API Node Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
`ArbitrationDelay`	milliseconds 0 0 / 4294967039 (0xFFFFFEFF)	N	all
`ArbitrationRank`	0-2 0 0 / 2	N	all
`BatchByteSize`	bytes 16K 1024 / 1M	N	all
`BatchSize`	records 256	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	1 / 992		
ExecuteOnComputer	name [none] ...	S	all
HostName	name or IP address [none] ...	N	all
Id	unsigned [none] 1 / 255	IS	all
MaxScanBatchSize	bytes 256K 32K / 16M	N	all
NodeId	unsigned [none] 1 / 63	IS	NDB 5.0.15

Note

To add new SQL or API nodes to the configuration of a running MySQL Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

17.3.2.4 Other MySQL Cluster Configuration Parameters

The summary tables in this section provide information about parameters used in the `[computer]`, `[tcp]`, `[shm]`, and `[sci]` sections of a `config.ini` file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about individual parameters, see Section 17.3.3.8, “MySQL Cluster TCP/IP Connections”, Section 17.3.3.10, “MySQL Cluster Shared-Memory Connections”, or Section 17.3.3.11, “SCI Transport Connections in MySQL Cluster”, as appropriate.

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary tables as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- N—Node restart: The parameter can be updated using a rolling restart (see Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”).
- S—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- I—Initial restart: Data nodes must be restarted using the `--initial` option.

For more information about restart types, see Section 17.3.2, “Overview of MySQL Cluster Configuration Parameters, Options, and Variables”.

Table 17.4 Computer Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
HostName	name or IP address [none] ...	N	all
Id	string [none] ...	IS	all

Table 17.5 TCP Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
Checksum	boolean false true, false	N	all
Group	unsigned 55 0 / 200	N	all
NodeId1	numeric [none] ...	N	all
NodeId2	numeric [none] ...	N	all
NodeIdServer	numeric	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	[none] ...		
PortNumber	unsigned [none] 0 / 64K	S	all
Proxy	string [none] ...	N	all
ReceiveBufferMemory	bytes 2M 16K / 4294967039 (0xFFFFFEFF)	N	all
SendBufferMemory	unsigned 2M 256K / 4294967039 (0xFFFFFEFF)	N	all
SendSignalId	boolean [see text] true, false	N	all

Table 17.6 Shared Memory Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
Checksum	boolean true true, false	N	all
Group	unsigned 35 0 / 200	N	all
NodeId1	numeric	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	[none] ...		
NodeId2	numeric [none] ...	N	all
NodeIdServer	numeric [none] ...	N	all
PortNumber	unsigned [none] 0 / 64K	S	all
SendSignalId	boolean false true, false	N	all
ShmKey	unsigned [none] 0 / 4294967039 (0xFFFFFEFF)	N	all
ShmSize	bytes 1M 64K / 4294967039 (0xFFFFFEFF)	N	all
Signum	unsigned [none] 0 / 4294967039 (0xFFFFFEFF)	N	all

Table 17.7 SCI Configuration Parameters

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
Checksum	boolean	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	false true, false		
Group	unsigned 15 0 / 200	N	all
Host1SciId0	unsigned [none] 0 / 4294967039 (0xFFFFFEFF)	N	all
Host1SciId1	unsigned 0 0 / 4294967039 (0xFFFFFEFF)	N	all
Host2SciId0	unsigned [none] 0 / 4294967039 (0xFFFFFEFF)	N	all
Host2SciId1	unsigned 0 0 / 4294967039 (0xFFFFFEFF)	N	all
NodeId1	numeric [none] ...	N	all
NodeId2	numeric [none] ...	N	all
NodeIdServer	numeric [none] ...	N	all
PortNumber	unsigned [none] 0 / 64K	S	all
SendLimit	unsigned 8K	N	all

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Parameter Name	Type or Units Default Value Minimum/ Maximum or Permitted Values	Restart Type	In Version ... (and later)
	128 / 32K		
SendSignalId	boolean true true, false	N	all
SharedBufferSize	unsigned 10M 64K / 4294967039 (0xFFFFFEFF)	N	all

17.3.2.5 MySQL Cluster mysqld Option and Variable Reference

The following table provides a list of the command-line options, server and status variables applicable within `mysqld` when it is running as an SQL node in a MySQL Cluster. For a table showing *all* command-line options, server and status variables available for use with `mysqld`, see Section 5.1.1, "Server Option and Variable Reference".

Table 17.8 MySQL Server Options and Variables for MySQL Cluster: MySQL 5.0

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
Com_show_ndb_status		
No	No	Yes
No	Both	No
DESCRIPTION: Count of SHOW NDB STATUS statements		
Handler_discover		
No	No	Yes
No	Both	No
DESCRIPTION: Number of times that tables have been discovered		
have_ndbcluster		
No	Yes	No
No	Global	No
DESCRIPTION: Whether mysqld supports NDB Cluster tables (set by --ndbcluster option)		
ndb-connectstring		
Yes	No	No
Yes		No

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
DESCRIPTION: Point to the management server that distributes the cluster configuration		
ndb-mgmd-host		
Yes	No	No
Yes		No
DESCRIPTION: Set the host (and port, if desired) for connecting to management server		
ndb-nodeid		
Yes	No	Yes
Yes	Global	No
DESCRIPTION: MySQL Cluster node ID for this MySQL server		
ndb_autoincrement_prefetch_sz		
Yes	Yes	No
Yes	Both	Yes
DESCRIPTION: NDB auto-increment prefetch size		
ndb_cache_check_time		
Yes	Yes	No
Yes	Global	Yes
DESCRIPTION: Number of milliseconds between checks of cluster SQL nodes made by the MySQL query cache		
Ndb_cluster_node_id		
No	No	Yes
No	Both	No
DESCRIPTION: If the server is acting as a MySQL Cluster node, then the value of this variable its node ID in the cluster		
Ndb_config_from_host		
No	No	Yes
No	Both	No
DESCRIPTION: The host name or IP address of the Cluster management server. Formerly Ndb_connected_host		
Ndb_config_from_port		
No	No	Yes
No	Both	No
DESCRIPTION: The port for connecting to Cluster management server. Formerly Ndb_connected_port		
ndb_force_send		
Yes	Yes	No
Yes	Both	Yes
DESCRIPTION: Forces sending of buffers to NDB immediately, without waiting for other threads		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
ndb_index_stat_cache_entries		
Yes	Yes	No
Yes	Both	Yes
DESCRIPTION: Sets the granularity of the statistics by determining the number of starting and ending keys		
ndb_index_stat_enable		
Yes	Yes	No
Yes	Both	Yes
DESCRIPTION: Use NDB index statistics in query optimization		
ndb_index_stat_update_freq		
Yes	Yes	No
Yes	Both	Yes
DESCRIPTION: How often to query data nodes instead of the statistics cache		
ndb_optimized_node_selection		
Yes	Yes	No
Yes	Global	No
DESCRIPTION: Determines how an SQL node chooses a cluster data node to use as transaction coordinator		
ndb_report_thresh_binlog_epoch_slip		
Yes	No	No
Yes		No
DESCRIPTION: This is a threshold on the number of epochs to be behind before reporting binary log status		
ndb_report_thresh_binlog_mem_usage		
Yes	No	No
Yes		No
DESCRIPTION: This is a threshold on the percentage of free memory remaining before reporting binary log status		
ndb_use_exact_count		
No	Yes	No
No	Both	Yes
DESCRIPTION: Use exact row count when planning queries		
ndb_use_transactions		
Yes	Yes	No
Yes	Both	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option or Variable Name		
Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
Notes		
DESCRIPTION: Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query `ndbcluster`		
Yes	No	No
Yes		No
DESCRIPTION: Enable NDB Cluster (if this version of MySQL supports it) Disabled by `--skip-ndbcluster`		

17.3.3 MySQL Cluster Configuration Files

Configuring MySQL Cluster requires working with two files:

- `my.cnf`: Specifies options for all MySQL Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file, sometimes known as the *global configuration file*, is read only by the MySQL Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see Sections of the `config.ini` File.

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”.

17.3.3.1 MySQL Cluster Configuration: Basic Example

To support MySQL Cluster, you will need to update `my.cnf` as shown in the following example. You may also specify these parameters on the command line when invoking the executables.

Note

The options shown here should not be confused with those that are used in `config.ini` global configuration files. Global configuration options are discussed later in this section.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid in MySQL 5.0)

# enable ndbcluster storage engine, and provide connection string for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
# provide connection string for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connection string for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connection strings, see Section 17.3.3.2, “MySQL Cluster Connection Strings”.)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connection string for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Important

Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design*.

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional `NDB` variables that can be set in the `my.cnf` file, see MySQL Cluster System Variables.

The MySQL Cluster global configuration file is by convention named `config.ini` (but this is not required). It is read by `ndb_mgmd` at startup and can be placed anywhere. Its location and name are specified by using `--config-file=path_name` on the `ndb_mgmd` command line. If the configuration file is not specified, `ndb_mgmd` by default tries to read a file named `config.ini` located in the current working directory.

The global configuration file for MySQL Cluster uses INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (“`:`”) as well as the equal sign (“`=`”); however, the equal sign is preferred. Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. (*Exception*: The `NoOfReplicas` configuration parameter has no default value, and must always be specified explicitly in the `[ndbd default]` section.) To create a default value section, simply add the word `default` to the section name. For example, an `[ndbd]` section contains parameters that apply to a particular data node,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

whereas an `[ndbd default]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `[ndbd default]` section that contains a `DataMemory` line to specify the data memory size.

The global configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two SQL nodes is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first SQL node can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com

[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[ndbd]
HostName= ndbd_2.mysql.com

[ndbd]
HostName= ndbd_3.mysql.com

[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `[ndbd]` sections defining these nodes.

Note

Do not place comments on the same line as a section heading in the `config.ini` file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the config.ini File

There are six different sections that you can use in the `config.ini` configuration file, as described in the following list:

- `[computer]`: Defines cluster hosts. This is not required to configure a viable MySQL Cluster, but be may used as a convenience when setting up a large cluster. See Section 17.3.3.3, “Defining Computers in a MySQL Cluster”, for more information.
- `[ndbd]`: Defines a cluster data node (`ndbd` process). See Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”, for details.
- `[mysqld]`: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see Section 17.3.3.6, “Defining SQL and Other API Nodes in a MySQL Cluster”.
- `[mgm]` or `[ndb_mgmd]`: Defines a cluster management server (MGM) node. For information concerning the configuration of management nodes, see Section 17.3.3.4, “Defining a MySQL Cluster Management Server”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `[tcp]`: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default connection protocol. Normally, `[tcp]` or `[tcp default]` sections are not required to set up a MySQL Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See Section 17.3.3.8, "MySQL Cluster TCP/IP Connections", for information about available TCP/IP configuration parameters and how to use them. (You may also find Section 17.3.3.9, "MySQL Cluster TCP/IP Connections Using Direct Connections" to be of interest in some cases.)

- `[shm]`: Defines shared-memory connections between nodes. In MySQL 5.0, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see Section 17.3.3.10, "MySQL Cluster Shared-Memory Connections".

- `[sci]`:Defines *Scalable Coherent Interface* connections between cluster data nodes. Such connections require software which, while freely available, is not part of the MySQL Cluster distribution, as well as specialized hardware. See Section 17.3.3.11, "SCI Transport Connections in MySQL Cluster" for detailed information about SCI interconnects.

You can define `default` values for each section. All Cluster parameter names are case-insensitive, which differs from parameters specified in `my.cnf` or `my.ini` files.

17.3.3.2 MySQL Cluster Connection Strings

With the exception of the MySQL Cluster management server (`ndb_mgmd`), each node that is part of a MySQL Cluster requires a *connection string* that points to the management server's location. This connection string is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connection string is as follows:

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]

host-definition:
    host_name[:port_number]
```

`node_id` is an integer greater than or equal to 1 which identifies a node in `config.ini`. *`host_name`* is a string representing a valid Internet host name or IP address. *`port_number`* is an integer referring to a TCP/IP port number.

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):   "myhost1"
```

`localhost:1186` is used as the default connection string value if none is provided. If *`port_num`* is omitted from the connection string, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see http://www.iana.org/assignments/port-numbers for details).

By listing multiple host definitions, it is possible to designate several redundant management servers. A MySQL Cluster data or API node attempts to contact successive management servers on each host in the order specified, until a successful connection has been established.

There are a number of different ways to specify the connection string:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)

- It is also possible to set the connection string for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.

- For backward compatibility, two other options are available, using the same syntax:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

1. Set the NDB_CONNECTSTRING environment variable to contain the connection string.
2. Write the connection string for each executable into a text file named Ndb.cfg and place this file in the executable's startup directory.

However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connection string is to set it on the command line or in the my.cnf file for each executable.

The maximum length of a connection string is 1024 characters.

17.3.3.3 Defining Computers in a MySQL Cluster

The [computer] section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- Id

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	string	[none]	...	IS

This is a unique identifier, used to refer to the host computer elsewhere in the configuration file.

Important

The computer ID is *not* the same as the node ID used for a management, API, or data node. Unlike the case with node IDs, you cannot use NodeId in place of Id in the [computer] section of the config.ini file.

- HostName

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

This is the computer's hostname or IP address.

17.3.3.4 Defining a MySQL Cluster Management Server

The [ndb_mgmd] section is used to configure the behavior of the management server. [mgm] can be used as an alias; the two section names are equivalent. All parameters in the following list are optional and assume their default values if omitted.

Note

If neither the ExecuteOnComputer nor the HostName parameter is present, the default value localhost will be assumed for both.

- Id

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	1 - 63	IS

Each node in the cluster has a unique identity, which is represented by an integer value in the range 1 to 63 inclusive. This ID is used by all internal cluster messages for addressing the node.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In MySQL 5.0.15 and later, `NodeId` is a synonym for this parameter, and is the preferred form. In MySQL Cluster NDB 6.2 and later, `Id` is deprecated in favor of `NodeId` for identifying management nodes.

- `NodeId`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.15	unsigned	[none]	1 - 63	IS

Beginning with MySQL 5.0.15, `NodeId` is available as a synonym for `Id`.

In MySQL Cluster NDB 6.2 and later, `Id` is deprecated in favor of `NodeId` for identifying management nodes.

- `ExecuteOnComputer`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name	[none]	...	S

This refers to the `Id` set for one of the computers defined in a `[computer]` section of the `config.ini` file.

- `PortNumber`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	1186	0 - 64K	S

This is the port number on which the management server listens for configuration requests and management commands.

- `HostName`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `LogDestination`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	{CONSOLE|SYSLOG|FILE}	[see text]	...	N

This parameter specifies where to send cluster logging information. There are three options in this regard—`CONSOLE`, `SYSLOG`, and `FILE`—with `FILE` being the default:

- `CONSOLE` outputs the log to `stdout`:

This documentation is for an older version. If you're

```
CONSOLE
```

This documentation is for an older version. If you're

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

  Note

  Not every facility is necessarily supported by every operating system.

  ```
  SYSLOG:facility=syslog
  ```

- `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:

  - `filename`: The name of the log file.
  - `maxsize`: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending `.N` to the file name, where `N` is the next number not yet used with this name.
  - `maxfiles`: The maximum number of log files.

  ```
  FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
  ```

  The default value for the `FILE` parameter is `FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6`, where `node_id` is the ID of the node.

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- `ArbitrationRank`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	0-2	1	0 - 2	N

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. `ArbitrationRank` can take one of the following values:

- `0`: The node will never be used as an arbitrator.
- `1`: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
- `2`: Indicates a low-priority node which be used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

- `ArbitrationDelay`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	0	0 - 4294967039 (0xFFFFFEFF)	N

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- `DataDir`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	path	.	...	N

This specifies the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for `LogDestination` as discussed previously in this section.)

The default value for this parameter is the directory in which `ndb_mgmd` is located.

- `PortNumberStats`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	0 - 64K	N

This parameter specifies the port number used to obtain statistical information from a MySQL Cluster management server. It has no default value.

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect.

To add new management servers to a running MySQL Cluster, it is also necessary to perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 17.1.5.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

17.3.3.5 Defining MySQL Cluster Data Nodes

The `[ndbd]` and `[ndbd default]` sections are used to configure the behavior of the cluster's data nodes.

There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameters are:

- Either `ExecuteOnComputer` or `HostName`, which must be defined in the local `[ndbd]` section.
- The parameter `NoOfReplicas`, which must be defined in the `[ndbd default]` section, as it is common to all Cluster data nodes.

Most data node parameters are set in the `[ndbd default]` section. Only those parameters explicitly stated as being able to set local values are permitted to be changed in the `[ndbd]` section. Where present, `HostName`, `Id` and `ExecuteOnComputer` *must* be defined in the local `[ndbd]` section, and not

This documentation is for an older version. If you're
This documentation is for an older version. If you're

in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, `100K` means 100 × 1024 = 102400.) Parameter names and values are currently case-sensitive.

Identifying data nodes. The `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- `Id`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	1 - 48	IS

This is the node ID used as the address of the node for all cluster internal messages. For data nodes, this is an integer in the range 1 to 48 inclusive. Each node in the cluster must have a unique identifier.

In MySQL 5.0.15 and later, `NodeId` is a synonym for this parameter, and is the preferred form. In MySQL Cluster NDB 6.2 and later, `Id` is deprecated in favor of `NodeId` for identifying data nodes.

- `NodeId`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.15	unsigned	[none]	1 - 48	IS

Beginning with MySQL 5.0.15, `NodeId` is available as a synonym for `Id`.

In MySQL Cluster NDB 6.2 and later, `Id` is deprecated in favor of `NodeId` for identifying management nodes.

- `ExecuteOnComputer`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name	[none]	...	S

This refers to the `Id` set for one of the computers defined in a `[computer]` section.

- `HostName`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	localhost	...	N

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `ServerPort`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	1 - 64K	S

Each node in the cluster uses a port to connect to other nodes. By default, this port is allocated dynamically in such a way as to ensure that no two nodes on the same host computer receive the same port number, so it should normally not be necessary to specify a value for this parameter.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

However, if you need to be able to open specific ports in a firewall to permit communication between data nodes and API nodes (including SQL nodes), you can set this parameter to the number of the desired port in an `[ndbd]` section or (if you need to do this for multiple data nodes) the `[ndbd default]` section of the `config.ini` file, and then open the port having that number for incoming connections from SQL nodes, API nodes, or both.

Note

Connections from data nodes to management nodes is done using the `ndb_mgmd` management port (the management server's `PortNumber`; see Section 17.3.3.4, "Defining a MySQL Cluster Management Server") so outgoing connections to that port from any data nodes should always be permitted.

- `NoOfReplicas`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	[none]	1 - 4	IS

This global parameter can be set only in the `[ndbd default]` section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that `NoOfReplicas` is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to fail.

If no node IDs are provided, the order of the data nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's `SHOW` command.

There is no default value for `NoOfReplicas`; the recommended value is 2 for most common usage scenarios.

The maximum possible value is 4; *currently, only the values 1 and 2 are actually supported*.

Important

Setting `NoOfReplicas` to 1 means that there is only a single copy of all Cluster data; in this case, the loss of a single data node causes the cluster to fail because there are no additional copies of the data stored by that node.

The value for this parameter must divide evenly into the number of data nodes in the cluster. For example, if there are two data nodes, then `NoOfReplicas` must be equal to either 1 or 2, since 2/3 and 2/4 both yield fractional values; if there are four data nodes, then `NoOfReplicas` must be equal to 1, 2, or 4.

- `DataDir`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	path	.	...	IN

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

The default is the data node process working directory.

- `FileSystemPath`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	path	DataDir	...	IN

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs, and data files are placed. The default is the directory specified by `DataDir`.

Note

This directory must exist before the `ndbd` process is initiated.

The recommended directory hierarchy for MySQL Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's file system is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- `BackupDataDir`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	path	[see text]	...	IN

This parameter specifies the directory in which backups are placed.

Important

The string '`/BACKUP`' is always appended to this value. For example, if you set the value of `BackupDataDir` to `/var/lib/cluster-data`, then all backups are stored under `/var/lib/cluster-data/BACKUP`. This also means that the *effective* default backup location is the directory named `BACKUP` under the location specified by the `FileSystemPath` parameter.

Data memory, index memory, and string memory. `DataMemory` and `IndexMemory` are `[ndbd]` parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how `DataMemory` and `IndexMemory` are used, as they usually need to be updated to reflect actual usage by the cluster:

- `DataMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	80M	1M - 1024G	N

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by `DataMemory` is used to store both the actual records and indexes. Each record is currently of fixed size. (Even `VARCHAR` columns are stored as fixed-width columns.) There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored

This documentation is for an older version. If you're
This documentation is for an older version. If you're

in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

The maximum record size is currently 8052 bytes.

The memory space defined by `DataMemory` is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index. A common error among users is to assume that all indexes are stored in the memory allocated by `IndexMemory`, but this is not the case: Only primary key and unique hash indexes use this memory; ordered indexes use the memory allocated by `DataMemory`. However, creating a primary key or unique hash index also creates an ordered index on the same keys, unless you specify `USING HASH` in the index creation statement. This can be verified by running `ndb_desc -d db_name table_name` in the management client.

MySQL Cluster can use a maximum of 512 MB for hash indexes per partition, which means in some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. This can also pose a problem with data node restarts on nodes that are heavily loaded with data. You can force `NDB` to create extra partitions for MySQL Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient.

The memory space allocated by `DataMemory` consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`.

In addition, due to the way in which new pages are allocated when the capacity of the current page is exhausted, there is an additional overhead of approximately 18.75%. When more `DataMemory` is required, more than one new page is allocated, according to the following formula:

```
number of new pages = FLOOR(number of current pages × 0.1875) + 1
```

For example, if 15 pages are currently allocated to a given table and an insert to this table requires additional storage space, the number of new pages allocated to the table is `FLOOR(15 × 0.1875) + 1 = FLOOR(2.8125) + 1 = 2 + 1 = 3`. Now 15 + 3 = 18 memory pages are allocated to the table. When the last of these 18 pages becomes full, `FLOOR(18 × 0.1875) + 1 = FLOOR(3.3750) + 1 = 3 + 1 = 4` new pages are allocated, so the total number of pages allocated to the table is now 22.

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that `DataMemory` pages, once allocated to a given table, cannot be used by other tables.) Performing a node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The `DataMemory` memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the `DataMemory`. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Large transactions use more memory

The default value for `DataMemory` is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2−4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.

- `IndexMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	18M	1M - 1T	N

This parameter controls the amount of storage used for hash indexes in MySQL Cluster. Hash indexes are always used for primary key indexes, unique indexes, and unique constraints. Note that when defining a primary key and a unique index, two indexes will be created, one of which is a hash index used for all tuple accesses as well as lock handling. It is also used to enforce unique constraints.

The size of the hash index is 25 bytes per record, plus the size of the primary key. For primary keys larger than 32 bytes another 8 bytes is added.

The default value for `IndexMemory` is 18MB. The minimum is 1MB.

- `StringMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	% or bytes	0	0 - 4294967039 (0xFFFFFEFF)	S

This parameter determines how much memory is allocated for strings such as table names, and is specified in an `[ndbd]` or `[ndbd default]` section of the `config.ini` file. A value between `0` and `100` inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of `.FRM` files, `MaxNoOfTriggers`, maximum column name size, and maximum default column value. In general it is safe to assume that the maximum default value is approximately 5 MB for a MySQL Cluster having 1000 tables.

A value greater than `100` is interpreted as a number of bytes.

In MySQL 5.0, the default value is `100`—that is, 100 percent of the default maximum, or roughly 5 MB. It is possible to reduce this value safely, but it should never be less than 5 percent. If you encounter Error 773 `Out of string memory, please modify StringMemory config parameter: Permanent error: Schema error`, this means that means that you have set the `StringMemory` value too low. `25` (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions, as when there are hundreds or thousands of `NDB` tables with names whose lengths and columns whose number approach their permitted maximums.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns `a` and `b` consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with `b` as primary key and `a` as a column. This other table consumes an additional 29 bytes of index memory per record in the `example` table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in MySQL Cluster to handle uniqueness constraints.

The only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both `IndexMemory` and `DataMemory` is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two replicas, there will be two node groups. Thus, the total data memory available is 2 × `DataMemory` for each data node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

`DataMemory` and `IndexMemory` can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire MySQL Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction parameters. The next three `[ndbd]` parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

- `MaxNoOfConcurrentTransactions`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	4096	32 - 4294967039 (0xFFFFFEFF)	N

Each cluster data node requires a transaction record for each active transaction in the cluster. The task of coordinating transactions is distributed among all of the data nodes. The total number of transaction

This documentation is for an older version. If you're

records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual SQL nodes. Each such connection requires at least one transaction record, plus an additional transaction object per table accessed by that connection. This means that a reasonable minimum for this parameter is

```
MaxNoOfConcurrentTransactions =
    (maximum number of tables accessed in any single transaction + 1)
    * number of cluster SQL nodes
```

Suppose that there are 10 SQL nodes using the cluster. A single join involving 10 tables requires 11 transaction records; if there are 10 such joins in a transaction, then 10 * 11 = 110 transaction records are required for this transaction, per SQL node, or 110 * 10 = 1100 transaction records total. Each data node can be expected to handle TotalNoOfConcurrentTransactions / number of data nodes. For a MySQL Cluster having 4 data nodes, this would mean setting `MaxNoOfConcurrentTransactions` on each data node to 1100 / 4 = 275. In addition, you should provide for failure recovery by insuring that a single node group can accommodate all concurrent transactions; in other words, that each data node's `MaxNoOfConcurrentTransactions` is sufficient to cover a number of transaction equal to TotalNoOfConcurrentTransactions / number of node groups. If this cluster has a single node group, then `MaxNoOfConcurrentTransactions` should be set to 1100 (the same as the total number of concurrent transactions for the entire cluster).

In addition, each transaction involves at least one operation; for this reason, the value set for `MaxNoOfConcurrentTransactions` should always be no more than the value of `MaxNoOfConcurrentOperations`.

This parameter must be set to the same value for all cluster data nodes. This is due to the fact that, when a data node fails, the oldest surviving node re-creates the transaction state of all transactions that were ongoing in the failed node.

It is possible to change this value using a rolling restart, but the amount of traffic on the cluster must be such that no more transactions occur than the lower of the old and new levels while this is taking place.

The default value is 4096.

- `MaxNoOfConcurrentOperations`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	32K	32 - 4294967039 (0xFFFFFEFF)	N

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions which involve only a few operations and records, the default value for this parameter is usually sufficient. Performing large transactions involving many records usually requires that you increase its value.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle one million concurrent updates using transactions, you should set this value to 1000000 / 4 = 250000. To help provide resiliency against failures, it is suggested that you set

This documentation is for an older version. If you're
This documentation is for an older version. If you're

this parameter to a value that is high enough to permit an individual data node to handle the load for its node group. In other words, you should set the value equal to `total number of concurrent operations / number of node groups`. (In the case where there is a single node group, this is the same as the total number of concurrent operations for the entire cluster.)

Because each transaction always involves at least one operation, the value of `MaxNoOfConcurrentOperations` should always be greater than or equal to the value of `MaxNoOfConcurrentTransactions`.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` will always be used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- `MaxNoOfLocalOperations`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	UNDEFINED	32 - 4294967039 (0xFFFFFEFF)	N

By default, this parameter is calculated as 1.1 × `MaxNoOfConcurrentOperations`. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

Transaction temporary storage. The next set of `[ndbd]` parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- `MaxNoOfConcurrentIndexOperations`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	8K	0 - 4294967039 (0xFFFFFEFF)	N

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter `MaxNoOfConcurrentOperations`.

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- `MaxNoOfFiredTriggers`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	4000	0 - 4294967039 (0xFFFFFEFF)	N

The default value of `MaxNoOfFiredTriggers` is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- `TransactionBufferMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	1M	1K - 4294967039 (0xFFFFFEFF)	N

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for `TransactionBufferMemory` is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter `ZATTRBUF_FILESIZE` (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000 × 128 bytes (500KB). A similar buffer for key information, `ZDATABUF_FILESIZE` (also in `Dbtc.hpp`) contains 4000 × 16 = 62.5KB of buffer space. `Dbtc` is the module that handles transaction coordination.

Scans and buffering. There are additional `[ndbd]` parameters in the `Dblqh` module (in `ndb/src/kernel/blocks/Dblqh/Dblqh.hpp`) that affect reads and updates. These include `ZATTRINBUF_FILESIZE`, set by default to 10000 × 128 bytes (1250KB) and `ZDATABUF_FILE_SIZE`, set by default to 10000*16 bytes (roughly 156KB) of buffer space. To date, there have been neither any

This documentation is for an older version. If you're
This documentation is for an older version. If you're

reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- `MaxNoOfConcurrentScans`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	256	2 - 500	N

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

- `MaxNoOfLocalScans`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	[see text]	32 - 4294967039 (0xFFFFFEFF)	N

Specifies the number of local scan records if many scans are not fully parallelized. If the number of local scan records is not provided, it is calculated as the product of `MaxNoOfConcurrentScans` and the number of data nodes in the system, plus 2. The minimum value is 32.

- `BatchSizePerLocalScan`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	64	1 - 992	N

This parameter is used to calculate the number of lock records used to handle concurrent scan operations.

`BatchSizePerLocalScan` has a strong connection to the `BatchSize` defined in the SQL nodes.

- `LongMessageBuffer`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	1M	512K - 4294967039 (0xFFFFFEFF)	N

This is an internal buffer used for passing messages within individual nodes and between nodes. Although it is highly unlikely that this would need to be changed, it is configurable. By default, it is set to 1MB.

Logging and checkpointing. The following `[ndbd]` parameters control log and checkpoint behavior.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `NoOfFragmentLogFiles`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	8	3 - 4294967039 (0xFFFFFEFF)	IN

This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A `REDO` log record is not removed until three local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

How these parameters interact and proposals for how to configure them are discussed in Section 17.3.3.12, “Configuring MySQL Cluster Parameters for Local Checkpoints”.

The default parameter value is 8, which means 8 sets of 4 16MB files for a total of 512MB. In other words, REDO log space is always allocated in blocks of 64MB. In scenarios requiring a great many updates, the value for `NoOfFragmentLogFiles` may need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (`Out of log file space temporarily`). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

Important

This parameter cannot be changed “on the fly”; you must restart the node using `--initial`. If you wish to change this value for all data nodes in a running cluster, you can do so using a rolling node restart (using `--initial` when starting each data node).

- `MaxNoOfOpenFiles`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	40	20 - 4294967039 (0xFFFFFEFF)	N

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation requiring a change in this parameter should be reported as a bug*.

The default value is 40.

- `MaxNoOfSavedMessages`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	25	0 - 4294967039 (0xFFFFFEFF)	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This parameter sets the maximum number of trace files that are kept before overwriting old ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25 trace files.

Metadata objects. The next set of `[ndbd]` parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters. Note that these act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	1000	32 - 4294967039 (0xFFFFFEFF)	N

This parameter sets a suggested maximum number of attributes that can be defined in the cluster; like `MaxNoOfTables`, it is not intended to function as a hard upper limit.

A known issue in MySQL Cluster in MySQL 5.0 is that this parameter is occasionally treated as a hard limit for certain operations. This can lead to confusion when it is sometimes possible (or not possible, depending on the circumstances) to create more than `MaxNoOfAttributes` attributes. This is fixed in MySQL Cluster NDB 6.3 and later. (Bug #61684)

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes`, it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the execution of `ALTER TABLE` on a Cluster table, 3 times the number of attributes as in the original table are used, and a good practice is to permit double this amount. For example, if the MySQL Cluster table having the greatest number of attributes (*`greatest_number_of_attributes`*) has 100 attributes, a good starting point for the value of `MaxNoOfAttributes` would be `6 * greatest_number_of_attributes = 600`.

You should also estimate the average number of attributes per table and multiply this by the total number of MySQL Cluster tables. If this value is larger than the value obtained in the previous paragraph, you should use the larger value instead.

Assuming that you can create all desired tables without any problems, you should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of `MaxNoOfTables` and test it again.

- `MaxNoOfTables`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	128	8 - 1600	N
MySQL 5.0.0	integer	128	8 - 20320	N

A table object is allocated for each table and for each unique hash index in the cluster. This parameter sets a suggested maximum number of table objects for the cluster as a whole; like `MaxNoOfAttributes`, it is not intended to function as a hard upper limit.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A known issue in MySQL Cluster in MySQL 5.0 is that this parameter is occasionally treated as a hard limit for certain operations. This can lead to confusion when it is sometimes possible (or not possible, depending on the circumstances) to create more than `MaxNoOfTables` tables. This is fixed in MySQL Cluster NDB 6.3 and later. (Bug #61684)

For each attribute that has a `BLOB` data type an extra table is used to store most of the `BLOB` data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 20320. (This is a change from MySQL 4.1.) Each table object consumes approximately 20KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfOrderedIndexes`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	128	0 - 4294967039 (0xFFFFFEFF)	N

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index. `MaxNoOfOrderedIndexes` sets the total number of hash indexes that can be in use in the system at any one time.

The default value of this parameter is 128. Each hash index object consumes approximately 10KB of data per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfUniqueHashIndexes`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	64	0 - 4294967039 (0xFFFFFEFF)	N

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfTriggers`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	768	0 - 4294967039 (0xFFFFFEFF)	N

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfIndexes`

  This parameter is deprecated. You should use `MaxNoOfOrderedIndexes` and `MaxNoOfUniqueHashIndexes` instead.

  This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

  The default value of this parameter is 128.

Boolean parameters. The behavior of data nodes is also affected by a set of `[ndbd]` parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to `1` or `Y`, and as `FALSE` by setting them equal to `0` or `N`.

- `LockPagesInMainMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	true|false (1|0)	0	0 - 1	N
MySQL 5.0.0	true|false (1|0)	0	0 - 1	N
MySQL 5.0.36	numeric	0	0 - 2	N

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

Beginning with MySQL 5.0.36, this parameter takes one of the integer values `0`, `1`, or `2`, which act as follows:

- `0`: Disables locking. This is the default value.
- `1`: Performs the lock after allocating memory for the process.
- `2`: Performs the lock before memory for the process is allocated.

Previously, this parameter was a Boolean. `0` or `false` was the default setting, and disabled locking. `1` or `true` enabled locking of the process after its memory was allocated.

Important

Beginning with MySQL 5.0.36, it is no longer possible to use `true` or `false` for the value of this parameter; when upgrading from a previous version, you must change the value to `0`, `1`, or `2`.

This documentation is for an older version. If you're

Note

If the operating system is not configured to permit unprivileged users to lock pages, then the data node process making use of this parameter may have to be run as system root. (`LockPagesInMainMemory` uses the `mlockall` function. From Linux kernel 2.6.9, unprivileged users can lock memory as limited by `max locked memory`. For more information, see `ulimit -l` and http://linux.die.net/man/2/mlock).

Important

Beginning with `glibc` 2.10, `glibc` uses per-thread arenas to reduce lock contention on a shared pool, which consumes real memory. In general, a data node process does not need per-thread arenas, since it does not perform any memory allocation after startup. (This difference in allocators does not appear to affect performance significantly.)

The `glibc` behavior is intended to be configurable via the `MALLOC_ARENA_MAX` environment variable, but a bug in this this mechanism prior to `glibc` 2.16 meant that this variable could not be set to less than 8, so that the wasted memory could not be reclaimed. (Bug #15907219; see also http://sourceware.org/bugzilla/show_bug.cgi?id=13137 for more information concerning this issue.)

One possible workaround for this problem is to use the `LD_PRELOAD` environment variable to preload a `jemalloc` memory allocation library to take the place of that supplied with `glibc`.

- `StopOnError`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	1	0, 1	N

This parameter specifies whether a data node process should exit or perform an automatic restart when an error condition is encountered.

This parameter's default value is `1`; this means that, by default, an error causes the data node process to halt.

- `Diskless`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	true|false (1|0)	false	true, false	IS

It is possible to specify MySQL Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run `ndbd` on a diskless computer.

Important

This feature causes the *entire* cluster to operate in diskless mode.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` is disabled by default.

- `RestartOnErrorInsert`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	error code	2	0 - 4	N

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

Controlling timeouts, intervals, and disk paging. There are a number of `[ndbd]` parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- `TimeBetweenWatchDogCheck`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	6000	70 - 4294967039 (0xFFFFFEFF)	N

To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 4000 milliseconds (4 seconds).

- `StartPartialTimeout`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	30000	0 - 4294967039 (0xFFFFFEFF)	N

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- `StartPartitionedTimeout`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	60000	0 - 4294967039 (0xFFFFFEFF)	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed. If `StartPartitionedTimeout` is set to 0, the cluster waits indefinitely.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default timeout is 60000 milliseconds (60 seconds).

- `StartFailureTimeout`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	0	0 - 4294967039 (0xFFFFFEFF)	N

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as 10−15 minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- `HeartbeatIntervalDbDb`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	1500	10 - 4294967039 (0xFFFFFEFF)	N

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. After missing three heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is four times the heartbeat interval.

The default heartbeat interval is 1500 milliseconds (1.5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

See also Network communication and latency.

- `HeartbeatIntervalDbApi`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	1500	100 - 4294967039 (0xFFFFFEFF)	N

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for `HeartbeatIntervalDbDb`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

For more information, see Network communication and latency.

- `TimeBetweenLocalCheckpoints`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	number of 4-byte words, as a base-2 logarithm	20	0 - 31	N

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting `TimeBetweenLocalCheckpoints` to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- `TimeBetweenGlobalCheckpoints`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	2000	10 - 32000	N

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases—a complete crash of the cluster—is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- `TimeBetweenInactiveTransactionAbortCheck`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	1000	1000 - 4294967039 (0xFFFFFEFF)	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction will be checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- `TransactionInactiveTimeout`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	[see text]	0 - 4294967039 (0xFFFFFEFF)	N

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is `4G` (also the maximum). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a relatively small value. The unit is milliseconds.

- `TransactionDeadlockDetectionTimeout`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	1200	50 - 4294967039 (0xFFFFFEFF)	N

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection. In MySQL 5.0.20 and earlier versions, setting it too high could cause undesirable behavior in situations involving deadlocks and node failure. Beginning with MySQL 5.0.21, active transactions occurring during node failures are actively aborted by the MySQL Cluster Transaction Coordinator, and so high settings are no longer an issue with this parameter.

The default timeout value is 1200 milliseconds (1.2 seconds). The effective minimum value is 100 milliseconds; it is possible to set it as low as 50 milliseconds, but any such value is treated as 100 ms. (Bug #44099)

- `NoOfDiskPagesToDiskAfterRestartTUP`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	8K pages/100 milliseconds	40	1 - 4294967039 (0xFFFFFEFF)	N

When executing a local checkpoint, the algorithm flushes all data pages to disk. Merely doing so as quickly as possible without any moderation is likely to impose excessive loads on processors, networks, and disks. To control the write speed, this parameter specifies how many pages per

This documentation is for an older version. If you're
This documentation is for an older version. If you're

100 milliseconds are to be written. In this context, a “page” is defined as 8KB. This parameter is specified in units of 80KB per second, so setting `NoOfDiskPagesToDiskAfterRestartTUP` to a value of `20` entails writing 1.6MB in data pages to disk each second during a local checkpoint. This value includes the writing of UNDO log records for data pages. That is, this parameter handles the limitation of writes from data memory. UNDO log records for index pages are handled by the parameter `NoOfDiskPagesToDiskAfterRestartACC`. (See the entry for `IndexMemory` for information about index pages.)

In short, this parameter specifies how quickly to execute local checkpoints. It operates in conjunction with `NoOfFragmentLogFiles`, `DataMemory`, and `IndexMemory`.

For more information about the interaction between these parameters and possible strategies for choosing appropriate values for them, see Section 17.3.3.12, “Configuring MySQL Cluster Parameters for Local Checkpoints”.

The default value is 40 (3.2MB of data pages per second).

- `NoOfDiskPagesToDiskAfterRestartACC`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	8K pages/100 milliseconds	20	1 - 4294967039 (0xFFFFFEFF)	N

This parameter uses the same units as `NoOfDiskPagesToDiskAfterRestartTUP` and acts in a similar fashion, but limits the speed of writing index pages from index memory.

The default value of this parameter is 20 (1.6MB of index memory pages per second).

- `NoOfDiskPagesToDiskDuringRestartTUP`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	8K pages/100 milliseconds	40	1 - 4294967039 (0xFFFFFEFF)	N

This parameter is used in a fashion similar to `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, only it does so with regard to local checkpoints executed in the node when a node is restarting. A local checkpoint is always performed as part of all node restarts. During a node restart it is possible to write to disk at a higher speed than at other times, because fewer activities are being performed in the node.

This parameter covers pages written from data memory.

The default value is 40 (3.2MB per second).

- `NoOfDiskPagesToDiskDuringRestartACC`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	8K pages/100 milliseconds	20	1 - 4294967039 (0xFFFFFEFF)	N

Controls the number of index memory pages that can be written to disk during the local checkpoint phase of a node restart.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As with NoOfDiskPagesToDiskAfterRestartTUP and NoOfDiskPagesToDiskAfterRestartACC, values for this parameter are expressed in terms of 8KB pages written per 100 milliseconds (80KB/second).

The default value is 20 (1.6MB per second).

- ArbitrationTimeout

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	3000	10 - 4294967039 (0xFFFFFEFF)	N

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

The default value is 1000 milliseconds (1 second).

Buffering and logging. Several [ndbd] configuration parameters corresponding to former compile-time parameters were introduced in MySQL 4.1.5. These enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the NDB storage engine's file system abstraction layer.

- UndoIndexBuffer

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	2M	1M - 4294967039 (0xFFFFFEFF)	N

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The NDB storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 (Index UNDO buffers overloaded).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `UndoDataBuffer`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	16M	1M - 4294967039 (0xFFFFFEFF)	N

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 (`Data UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `RedoBuffer`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	8M	1M - 4294967039 (0xFFFFFEFF)	N

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

`RedoBuffer` sets the size of the buffer in which the REDO log is written, and is 8MB by default. The minimum value is 1MB.

If this buffer is too small, the NDB storage engine issues error code 1221 (`REDO log buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

Controlling log messages. In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that there will be no event reports made in that category.

This documentation is for an older version. If you're

By default, only the startup message is sent to `stdout`, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- `LogLevelStartup`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	1	0 - 15	N

The reporting level for events generated during startup of the process.

The default level is 1.

- `LogLevelShutdown`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	0	0 - 15	N

The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- `LogLevelStatistic`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	0	0 - 15	N

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- `LogLevelCheckpoint`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	log level	0	0 - 15	N

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- `LogLevelNodeRestart`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	0	0 - 15	N

The reporting level for events generated during node restart.

The default level is 0.

- `LogLevelConnection`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	0	0 - 15	N

The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- `LogLevelError`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	0	0 - 15	N

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- `LogLevelCongestion`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	levelr	0	0 - 15	N

The reporting level for events generated by congestion. These errors do not cause node failure but are still considered worth reporting.

The default level is 0.

- `LogLevelInfo`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	integer	0	0 - 15	N

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

Backup parameters. The `[ndbd]` parameters discussed in this section define memory buffers set aside for execution of online backups.

- `BackupDataBufferSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	2M	0 - 4294967039 (0xFFFFFEFF)	N
NDB 7.5.0	bytes	16M	2M - 4294967039 (0xFFFFFEFF)	N

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as `BackupWriteSize` (see below), the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup

This documentation is for an older version. If you're
This documentation is for an older version. If you're

process pauses the scan and waits until some disk writes have completed freeing up memory so that scanning may continue.

The default value is 2MB.

- `BackupLogBufferSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	2M	0 - 4294967039 (0xFFFFFEFF)	N

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must be large enough to handle the load caused by write activities while the backup is being made. See Section 17.5.3.3, “Configuration for MySQL Cluster Backups”.

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value is 2MB.

- `BackupMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	4M	0 - 4294967039 (0xFFFFFEFF)	N

This parameter is simply the sum of `BackupDataBufferSize` and `BackupLogBufferSize`.

The default value is 2MB + 2MB = 4MB.

Important

If `BackupDataBufferSize` and `BackupLogBufferSize` taken together exceed 4MB, then this parameter must be set explicitly in the `config.ini` file to their sum.

- `BackupWriteSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	32K	2K - 4294967039 (0xFFFFFEFF)	N

This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

The default value is 32KB.

This documentation is for an older version. If you're

- `BackupMaxWriteSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	256K	2K - 4294967039 (0xFFFFFEFF)	N

This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

The default value is 256KB.

Important

When specifying these parameters, the following relationships must hold true. Otherwise, the data node will be unable to start.

- `BackupDataBufferSize >= BackupWriteSize + 188KB`
- `BackupLogBufferSize >= BackupWriteSize + 16KB`
- `BackupMaxWriteSize >= BackupWriteSize`

Note

To add new data nodes to a MySQL Cluster, it is necessary to shut down the cluster completely, update the `config.ini` file, and then restart the cluster (that is, you must perform a system restart). All data node processes must be started with the `--initial` option.

Beginning with MySQL Cluster NDB 7.0, it is possible to add new data node groups to a running cluster online; however, we do not plan to implement this change in MySQL 5.0.

17.3.3.6 Defining SQL and Other API Nodes in a MySQL Cluster

The `[mysqld]` and `[api]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a `[mysqld]` section is used to indicate a MySQL server providing an SQL interface to the cluster, and an `[api]` section is used for applications other than `mysqld` processes accessing cluster data, but the two designations are actually synonomous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an `[api]` section.

Note

For a discussion of MySQL server options for MySQL Cluster, see mysqld Command Options for MySQL Cluster; for information about MySQL server system variables relating to MySQL Cluster, see MySQL Cluster System Variables.

- `Id`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	1 - 63	IS

The `Id` is an integer value used to identify the node in all cluster internal messages. It must be an integer in the range 1 to 63 inclusive, and must be unique among all node IDs within the cluster.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In MySQL 5.0.15 and later, `NodeId` is a synonym for this parameter, and is the preferred form. In MySQL Cluster NDB 6.2 and later, `Id` is deprecated in favor of `NodeId` for identifying SQL and other API nodes.

- `NodeId`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.15	unsigned	[none]	1 - 63	IS

Beginning with MySQL 5.0.15, `NodeId` is available as a synonym for `Id`.

In MySQL Cluster NDB 6.2 and later, `Id` is deprecated in favor of `NodeId` for identifying SQL and API nodes.

- `ExecuteOnComputer`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name	[none]	...	S

This refers to the `Id` set for one of the computers (hosts) defined in a `[computer]` section of the configuration file.

- `HostName`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname, either this parameter or `ExecuteOnComputer` is required.

If no `HostName` or `ExecuteOnComputer` is specified in a given `[mysql]` or `[api]` section of the `config.ini` file, then an SQL or API node may connect using the corresponding “slot” from any host which can establish a network connection to the management server host machine. *This differs from the default behavior for data nodes, where `localhost` is assumed for `HostName` unless otherwise specified*.

- `ArbitrationRank`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	0-2	0	0 - 2	N

This parameter defines which nodes can act as arbitrators. Both management nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

- `ArbitrationDelay`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	milliseconds	0	0 - 4294967039 (0xFFFFFEFF)	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- `BatchByteSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	32K	1024 - 1M	N

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (`BatchSize`) and in terms of bytes (`BatchByteSize`). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set.

This parameter is measured in bytes. The default value is 32K.

- `BatchSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	records	64	1 - 992	N

This parameter is measured in number of records and is by default set to 64. The maximum size is 992.

- `MaxScanBatchSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	256K	32K - 16M	N

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

You can obtain some information from a MySQL server running as a Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----------------------------+---------------+
| Variable_name               | Value         |
+-----------------------------+---------------+
| Ndb_cluster_node_id         | 5             |
| Ndb_config_from_host        | 192.168.0.112 |
| Ndb_config_from_port        | 1186          |
| Ndb_number_of_storage_nodes | 4             |
+-----------------------------+---------------+
4 rows in set (0.02 sec)
```

For information about these Cluster system status variables, see Section 5.1.6, “Server Status Variables”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

To add new SQL or API nodes to the configuration of a running MySQL Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

17.3.3.7 MySQL Server Options and Variables for MySQL Cluster

This section provides information about MySQL server options, server and status variables that are specific to MySQL Cluster. For general information on using these, and for other options and variables not specific to MySQL Cluster, see Section 5.1, “The MySQL Server”.

For MySQL Cluster configuration parameters used in the cluster confiuration file (usually named `config.ini`), see Section 17.3, “MySQL Cluster Configuration”.

mysqld Command Options for MySQL Cluster

This section provides descriptions of `mysqld` server options relating to MySQL Cluster. For information about `mysqld` options not specific to MySQL Cluster, and for general information about the use of options with `mysqld`, see Section 5.1.3, “Server Command Options”.

For information about command-line options used with other MySQL Cluster processes (`ndbd`, `ndb_mgmd`, and `ndb_mgm`), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”. For information about command-line options used with `NDB` utility programs (such as `ndb_desc`, `ndb_size.pl`, and `ndb_show_tables`), see Section 17.4, “MySQL Cluster Programs”.

- `--ndbcluster`

Table 17.9 Type and value information for ndbcluster

Command Line	**System Variable**	**Status Variable**
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`ndbcluster`		
Yes	No	No
Yes		No
MySQL 5.0	boolean	FALSE
DESCRIPTION: Enable NDB Cluster (if this version of MySQL supports it) Disabled by `--skip-ndbcluster`		

The `NDBCLUSTER` storage engine is necessary for using MySQL Cluster. If a `mysqld` binary includes support for the `NDBCLUSTER` storage engine, the engine is disabled by default. Use the `--ndbcluster` option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--ndb-connectstring=connection_string`

Table 17.10 Type and value information for ndb-connectstring

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`ndb-connectstring`		
Yes	No	No
Yes		No
MySQL 5.0	string	
DESCRIPTION: Point to the management server that distributes the cluster configuration		

When using the `NDBCLUSTER` storage engine, this option specifies the management server that distributes cluster configuration data. See Section 17.3.3.2, “MySQL Cluster Connection Strings”, for syntax.

- `--ndb-mgmd-host=host[:port]`

Table 17.11 Type and value information for ndb-mgmd-host

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`ndb-mgmd-host`		
Yes	No	No
Yes		No
MySQL 5.0	string	localhost:1186
DESCRIPTION: Set the host (and port, if desired) for connecting to management server		

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--ndb-nodeid=#`

Table 17.12 Type and value information for ndb-nodeid

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`ndb-nodeid`		
Yes	No	Yes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
Yes	Global	No
5.0.45	integer	/ 1 - 63
DESCRIPTION: MySQL Cluster node ID for this MySQL server		

Set this MySQL server's node ID in a MySQL Cluster. This can be used instead of specifying the node ID as part of the connection string or in the `config.ini` file, or permitting the cluster to determine an arbitrary node ID. If you use this option, then `--ndb-nodeid` must be specified *before* `--ndb-connectstring`. If `--ndb-nodeid` is used *and* a node ID is specified in the connection string, then the MySQL server will not be able to connect to the cluster. In addition, if `--nodeid` is used, then either a matching node ID must be found in a `[mysqld]` or `[api]` section of `config.ini`, or there must be an "open" `[mysqld]` or `[api]` section in the file (that is, a section without an `Id` parameter specified).

Regardless of how the node ID is determined, its is shown as the value of the global status variable `Ndb_cluster_node_id` in the output of `SHOW STATUS`, and as `cluster_node_id` in the `connection` row of the output of `SHOW ENGINE NDBCLUSTER STATUS`.

For more information about node IDs for MySQL Cluster SQL nodes, see Section 17.3.3.6, "Defining SQL and Other API Nodes in a MySQL Cluster".

- `--skip-ndbcluster`

Table 17.13 Type and value information for skip-ndbcluster

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`skip-ndbcluster`		
Yes	No	No
Yes		No
DESCRIPTION: Disable the NDB Cluster storage engine		

Disable the `NDBCLUSTER` storage engine. This is the default for binaries that were built with `NDBCLUSTER` storage engine support; the server allocates memory and other resources for this storage engine only if the `--ndbcluster` option is given explicitly. See Section 17.3.1, "Quick Test Setup of MySQL Cluster", for an example.

MySQL Cluster System Variables

This section provides detailed information about MySQL server system variables that are specific to MySQL Cluster and the `NDB` storage engine. For system variables not specific to MySQL Cluster, see Section 5.1.4, "Server System Variables". For general information on using system variables, see Section 5.1.5, "Using System Variables".

- `have_ndbcluster`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 17.14 Type and value information for have_ndbcluster

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
have_ndbcluster		
No	Yes	No
No	Global	No
MySQL 5.0	boolean	
DESCRIPTION: Whether mysqld supports NDB Cluster tables (set by --ndbcluster option)		

YES if mysqld supports NDBCLUSTER tables. DISABLED if --skip-ndbcluster is used.

This variable is deprecated in MySQL 5.1, and is removed in MySQL 5.6. Use SHOW ENGINES instead.

- ndb_autoincrement_prefetch_sz

Table 17.15 Type and value information for ndb_autoincrement_prefetch_sz

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_autoincrement_prefetch_sz		
Yes	Yes	No
Yes	Both	Yes
MySQL 5.0	integer	32 / 1 - 256
5.0.56	integer	1 / 1 - 256
DESCRIPTION: NDB auto-increment prefetch size		

Determines the probability of gaps in an autoincremented column. Set it to 1 to minimize this. Setting it to a high value for optimization—makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers will be used in a batch of inserts. Default value: 32. Minimum value: 1.

Beginning with MySQL 5.0.56, this variable affects the number of AUTO_INCREMENT IDs that are fetched between statements only. Within a statement, at least 32 IDs are now obtained at a time. In MySQL 5.0.56 and later, the default value is 1. (Bug #31956)

Important

This variable does not affect inserts performed using INSERT ... SELECT.

- ndb_cache_check_time

This documentation is for an older version. If you're

Table 17.16 Type and value information for ndb_cache_check_time

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_cache_check_time		
Yes	Yes	No
Yes	Global	Yes
MySQL 5.0	integer	0 / -
DESCRIPTION: Number of milliseconds between checks of cluster SQL nodes made by the MySQL query cache		

The number of milliseconds that elapse between checks of MySQL Cluster SQL nodes by the MySQL query cache. Setting this to 0 (the default and minimum value) means that the query cache checks for validation on every query.

The recommended maximum value for this variable is 1000, which means that the check is performed once per second. A larger value means that the check is performed and possibly invalidated due to updates on different SQL nodes less often. It is generally not desirable to set this to a value greater than 2000.

- ndb_force_send

Table 17.17 Type and value information for ndb_force_send

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_force_send		
Yes	Yes	No
Yes	Both	Yes
MySQL 5.0	boolean	TRUE
DESCRIPTION: Forces sending of buffers to NDB immediately, without waiting for other threads		

Forces sending of buffers to NDB immediately, without waiting for other threads. Defaults to ON.

- ndb_index_stat_cache_entries

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 17.18 Type and value information for ndb_index_stat_cache_entries

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_index_stat_cache_entries		
Yes	Yes	No
Yes	Both	Yes
MySQL 5.0	integer	32 / 0 - 4294967295
DESCRIPTION: Sets the granularity of the statistics by determining the number of starting and ending keys		

Sets the granularity of the statistics by determining the number of starting and ending keys to store in the statistics memory cache. Zero means no caching takes place; in this case, the data nodes are always queried directly. Default value: `32`.

- `ndb_index_stat_enable`

Table 17.19 Type and value information for ndb_index_stat_enable

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_index_stat_enable		
Yes	Yes	No
Yes	Both	Yes
MySQL 5.0	boolean	OFF
DESCRIPTION: Use NDB index statistics in query optimization		

Use `NDB` index statistics in query optimization. Defaults to `ON`.

- `ndb_index_stat_update_freq`

Table 17.20 Type and value information for ndb_index_stat_update_freq

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_index_stat_update_freq		
Yes	Yes	No
Yes	Both	Yes
MySQL 5.0	integer	20 / 0 - 4294967295

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command Line	System Variable	Status Variable
Option File	Scope	Dynamic
From Version	Type	Default, Range
Notes		
DESCRIPTION: How often to query data nodes instead of the statistics cache		

How often to query data nodes instead of the statistics cache. For example, a value of `20` (the default) means to direct every 20th query to the data nodes.

- `ndb_optimized_node_selection`

Table 17.21 Type and value information for ndb_optimized_node_selection

Command Line	System Variable	Status Variable
Option File	Scope	Dynamic
From Version	Type	Default, Range
Notes		
`ndb_optimized_node_selection`		
Yes	Yes	No
Yes	Global	No
MySQL 5.0	boolean	ON
DESCRIPTION: Determines how an SQL node chooses a cluster data node to use as transaction coordinator		

Causes an SQL node to use the “closest” data node as transaction coordinator. For this purpose, a data node having a shared memory connection with the SQL node is considered to be “closest” to the SQL node; the next closest (in order of decreasing proximity) are: TCP connection to `localhost`; SCI connection; TCP connection from a host other than `localhost`.

This option is enabled by default. Set to `0` or `OFF` to disable it, in which case the SQL node uses each data node in the cluster in succession. When this option is disabled each SQL thread attempts to use a given data node 8 times before proceeding to the next one.

- `ndb_report_thresh_binlog_epoch_slip`

Table 17.22 Type and value information for ndb_report_thresh_binlog_epoch_slip

Command Line	System Variable	Status Variable
Option File	Scope	Dynamic
From Version	Type	Default, Range
Notes		
`ndb_report_thresh_binlog_epoch_slip`		
Yes	No	No
Yes		No
MySQL 5.0	integer	3 / 0 - 256

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
DESCRIPTION: This is a threshold on the number of epochs to be behind before reporting binary log status		

This is a threshold on the number of epochs to be behind before reporting binary log status. For example, a value of `3` (the default) means that if the difference between which epoch has been received from the storage nodes and which epoch has been applied to the binary log is 3 or more, a status message will be sent to the cluster log.

- `ndb_report_thresh_binlog_mem_usage`

Table 17.23 Type and value information for ndb_report_thresh_binlog_mem_usage

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`ndb_report_thresh_binlog_mem_usage`		
Yes	No	No
Yes		No
MySQL 5.0	integer	10 / 0 - 10
DESCRIPTION: This is a threshold on the percentage of free memory remaining before reporting binary log status		

This is a threshold on the percentage of free memory remaining before reporting binary log status. For example, a value of `10` (the default) means that if the amount of available memory for receiving binary log data from the data nodes falls below 10%, a status message will be sent to the cluster log.

- `ndb_use_exact_count`

Table 17.24 Type and value information for ndb_use_exact_count

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
`ndb_use_exact_count`		
No	Yes	No
No	Both	Yes
MySQL 5.0	boolean	ON
DESCRIPTION: Use exact row count when planning queries		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query. The default value is ON. For faster queries overall, disable this feature by setting the value of ndb_use_exact_count to OFF.

- ndb_use_transactions

Table 17.25 Type and value information for ndb_use_transactions

Command Line	System Variable	Status Variable
Option File	**Scope**	**Dynamic**
From Version	**Type**	**Default, Range**
Notes		
ndb_use_transactions		
Yes	Yes	No
Yes	Both	Yes
MySQL 5.0	boolean	ON
DESCRIPTION: Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query		

You can disable NDB transaction support by setting this variable's values to OFF (not recommended). The default is ON.

MySQL Cluster Status Variables

This section provides detailed information about MySQL server status variables that relate to MySQL Cluster and the NDB storage engine. For status variables not specific to MySQL Cluster, and for general information on using status variables, see Section 5.1.6, “Server Status Variables”.

- Handler_discover

  An SQL node can ask the NDB storage engine if it knows about a table with a given name. This is called discovery. Handler_discover indicates the number of times that tables have been discovered using this mechanism.

- Ndb_cluster_node_id

  If the server is acting as a MySQL Cluster node, then the value of this variable its node ID in the cluster.

  If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- Ndb_config_from_host

  If the server is part of a MySQL Cluster, the value of this variable is the host name or IP address of the Cluster management server from which it gets its configuration data.

  If the server is not part of a MySQL Cluster, then the value of this variable is an empty string.

  Prior to MySQL 5.0.23, this variable was named Ndb_connected_host.

- Ndb_config_from_port

  If the server is part of a MySQL Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

Prior to MySQL 5.0.23, this variable was named `Ndb_connected_port`.

- `Ndb_number_of_data_nodes`

  If the server is part of a MySQL Cluster, the value of this variable is the number of data nodes in the cluster.

  If the server is not part of a MySQL Cluster, then the value of this variable is 0.

  Prior to MySQL 5.0.29, this variable was named `Ndb_number_of_storage_nodes`.

17.3.3.8 MySQL Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for all connections between nodes in a MySQL Cluster. Normally it is not necessary to define TCP/IP connections; MySQL Cluster automatically sets up such connections for all data nodes, management nodes, and SQL or API nodes.

Note

For an exception to this rule, see Section 17.3.3.9, "MySQL Cluster TCP/IP Connections Using Direct Connections".

To override the default connection parameters, it is necessary to define a connection using one or more `[tcp]` sections in the `config.ini` file. Each `[tcp]` section explicitly defines a TCP/IP connection between two MySQL Cluster nodes, and must contain at a minimum the parameters `NodeId1` and `NodeId2`, as well as any connection parameters to override.

It is also possible to change the default values for these parameters by setting them in the `[tcp default]` section.

Important

Any `[tcp]` sections in the `config.ini` file should be listed *last*, following all other sections in the file. However, this is not required for a `[tcp default]` section. This requirement is a known issue with the way in which the `config.ini` file is read by the MySQL Cluster management server.

Connection parameters which can be set in `[tcp]` and `[tcp default]` sections of the `config.ini` file are listed here:

- `NodeId1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	numeric	[none]	...	N

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique `Id` values for each of these nodes as described in Section 17.3.3.6, "Defining SQL and Other API Nodes in a MySQL Cluster".

- `NodeId2`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	numeric	[none]	...	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique `Id` values for each of these nodes as described in Section 17.3.3.6, "Defining SQL and Other API Nodes in a MySQL Cluster".

- `HostName1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `SendBufferMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	256K	64K - 4294967039 (0xFFFFFEFF)	N

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer.

The default size of the send buffer is 256 KB; 2MB is recommended in most situations in which it is necessary to set this parameter. The minimum size is 64 KB; the theoretical maximum is 4 GB.

- `SendSignalId`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	[see text]	true, false	N

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `Checksum`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	false	true, false	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This parameter is a boolean parameter (enabled by setting it to `Y` or `1`, disabled by setting it to `N` or `0`). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- `PortNumber` (*OBSOLETE*)

  This formerly specified the port number to be used for listening for connections from other nodes. This parameter should no longer be used.

- `ReceiveBufferMemory`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	64K	16K - 4294967039 (0xFFFFFEFF)	N

  Specifies the size of the buffer used when receiving data from the TCP/IP socket.

  The default value of this parameter from its of 64 KB; 1M is recommended in most situations where the size of the receive buffer needs to be set. The minimum possible value is 16K; theoretical maximum is 4G.

17.3.3.9 MySQL Cluster TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the `[tcp]` section of the cluster `config.ini` file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the `172.23.72.*` subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the `1.1.0.*` address range as shown:

```
# Management Server
[ndb_mgmd]
Id=1
HostName=172.23.72.20

# SQL Node
[mysqld]
Id=2
HostName=172.23.72.21

# Data Nodes
[ndbd]
Id=3
HostName=172.23.72.22

[ndbd]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[tcp]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `HostName1` and `HostName2` parameters are used only when specifying direct TCP connections.

The use of direct TCP connections between data nodes can improve the cluster's overall efficiency by enabling the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency. It is important to note that to take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

17.3.3.10 MySQL Cluster Shared-Memory Connections

MySQL Cluster attempts to use the shared memory transporter and configure it automatically where possible. (In very early versions of MySQL Cluster, shared memory segments functioned only when the server binary was built using `--with-ndb-shm`.) `[shm]` sections in the `config.ini` file explicitly define shared-memory connections between nodes in the cluster. When explicitly defining shared memory as the connection method, it is necessary to define at least `NodeId1`, `NodeId2` and `ShmKey`. All other parameters have default values that should work well in most cases.

Important

SHM functionality is considered experimental only. It is not officially supported in any current MySQL Cluster release, and testing results indicate that SHM performance is not appreciably greater than when using TCP/IP for the transporter.

For these reasons, you must determine for yourself or by using our free resources (forums, mailing lists) whether SHM can be made to work correctly in your specific case.

- `NodeId1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	numeric	[none]	...	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `NodeId2`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	numeric	[none]	...	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `HostName1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `ShmKey`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	0 - 4294967039 (0xFFFFFEFF)	N

When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value.

- `ShmSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	bytes	1M	64K - 4294967039 (0xFFFFFEFF)	N

Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 1MB.

- `SendSignalId`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	false	true, false	N

To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `Y` causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `Checksum`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	true	true, false	N

This parameter is a boolean (`Y`/`N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- `SigNum`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	0 - 4294967039 (0xFFFFFEFF)	N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When using the shared memory transporter, a process sends an operating system signal to the other process when there is new data available in the shared memory. Should that signal conflict with with an existing signal, this parameter can be used to change it. This is a possibility when using SHM due to the fact that different operating systems use different signal numbers.

The default value of `SigNum` is 0; therefore, it must be set to avoid errors in the cluster log when using the shared memory transporter. Typically, this parameter is set to 10 in the `[shm default]` section of the `config.ini` file.

17.3.3.11 SCI Transport Connections in MySQL Cluster

`[sci]` sections in the `config.ini` file explicitly define SCI (Scalable Coherent Interface) connections between cluster nodes. Using SCI transporters in MySQL Cluster is supported only when the MySQL binaries are built using `--with-ndb-sci=/your/path/to/SCI`. The *`path`* should point to a directory that contains at a minimum `lib` and `include` directories containing SISCI libraries and header files. (See Section 17.3.4, "Using High-Speed Interconnects with MySQL Cluster" for more information about SCI.)

In addition, SCI requires specialized hardware.

It is strongly recommended to use SCI Transporters only for communication between `ndbd` processes. Note also that using SCI Transporters means that the `ndbd` processes never sleep. For this reason, SCI Transporters should be used only on machines having at least two CPUs dedicated for use by `ndbd` processes. There should be at least one CPU per `ndbd` process, with at least one CPU left in reserve to handle operating system activities.

- `NodeId1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	numeric	[none]	...	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `NodeId2`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	numeric	[none]	...	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `Host1SciId0`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	0 - 4294967039 (0xFFFFFEFF)	N

This identifies the SCI node ID on the first Cluster node (identified by `NodeId1`).

- `Host1SciId1`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	0	0 - 4294967039 (0xFFFFFEFF)	N

It is possible to set up SCI Transporters for failover between two SCI cards which then should use separate networks between the nodes. This identifies the node ID and the second SCI card to be used on the first node.

- `Host2SciId0`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	[none]	0 - 4294967039 (0xFFFFFEFF)	N

This identifies the SCI node ID on the second Cluster node (identified by `NodeId2`).

- `Host2SciId1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	0	0 - 4294967039 (0xFFFFFEFF)	N

When using two SCI cards to provide failover, this parameter identifies the second SCI card to be used on the second node.

- `HostName1`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SCI connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	name or IP address	[none]	...	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SCI connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `SharedBufferSize`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	10M	64K - 4294967039 (0xFFFFFEFF)	N

Each SCI transporter has a shared memory segment used for communication between the two nodes. Setting the size of this segment to the default value of 1MB should be sufficient for most applications.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Using a smaller value can lead to problems when performing many parallel inserts; if the shared buffer is too small, this can also result in a crash of the `ndbd` process.

- `SendLimit`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	unsigned	8K	128 - 32K	N

A small buffer in front of the SCI media stores messages before transmitting them over the SCI network. By default, this is set to 8KB. Our benchmarks show that performance is best at 64KB but 16KB reaches within a few percent of this, and there was little if any advantage to increasing it beyond 8KB.

- `SendSignalId`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	true	true, false	N

To trace a distributed message it is necessary to identify each message uniquely. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `Checksum`

Effective Version	Type/Units	Default	Range/Values	Restart Type
MySQL 5.0.0	boolean	false	true, false	N

This parameter is a boolean value, and is disabled by default. When `Checksum` is enabled, checksums are calculated for all messages before they are placed in the send buffer. This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

17.3.3.12 Configuring MySQL Cluster Parameters for Local Checkpoints

The parameters discussed in Logging and Checkpointing and in Data Memory, Index Memory, and String Memory that are used to configure local checkpoints for a MySQL Cluster do not exist in isolation, but rather are very much interdepedent on each other. In this section, we illustrate how these parameters—including `DataMemory`, `IndexMemory`, `NoOfDiskPagesToDiskAfterRestartTUP`, `NoOfDiskPagesToDiskAfterRestartACC`, and `NoOfFragmentLogFiles`—relate to one another in a working Cluster.

In this example, we assume that our application performs the following numbers of types of operations per hour:

- 50000 selects
- 15000 inserts
- 15000 updates
- 15000 deletes

We also make the following assumptions about the data used in the application:

- We are working with a single table having 40 columns.
- Each column can hold up to 32 bytes of data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- A typical `UPDATE` run by the application affects the values of 5 columns.
- No `NULL` values are inserted by the application.

A good starting point is to determine the amount of time that should elapse between local checkpoints (LCPs). It is worth noting that, in the event of a system restart, it takes 40-60 percent of this interval to execute the REDO log—for example, if the time between LCPs is 5 minutes (300 seconds), then it should take 2 to 3 minutes (120 to 180 seconds) for the REDO log to be read.

The maximum amount of data per node can be assumed to be the size of the `DataMemory` parameter. In this example, we assume that this is 2 GB. The `NoOfDiskPagesToDiskAfterRestartTUP` parameter represents the amount of data to be checkpointed per unit time—however, this parameter is actually expressed as the number of 8K memory pages to be checkpointed per 100 milliseconds. 2 GB per 300 seconds is approximately 6.8 MB per second, or 700 KB per 100 milliseconds, which works out to roughly 85 pages per 100 milliseconds.

Similarly, we can calculate `NoOfDiskPagesToDiskAfterRestartACC` in terms of the time for local checkpoints and the amount of memory required for indexes—that is, the `IndexMemory`. Assuming that we permit 512 MB for indexes, this works out to approximately 20 8-KB pages per 100 milliseconds for this parameter.

Next, we need to determine the number of REDO log files required—that is, fragment log files—the corresponding parameter being `NoOfFragmentLogFiles`. We need to make sure that there are sufficient REDO log files for keeping records for at least 3 local checkpoints. In a production setting, there are always uncertainties—for instance, we cannot be sure that disks always operate at top speed or with maximum throughput. For this reason, it is best to err on the side of caution, so we double our requirement and calculate a number of fragment log files which should be enough to keep records covering 6 local checkpoints.

It is also important to remember that the disk also handles writes to the REDO log and UNDO log, so if you find that the amount of data being written to disk as determined by the values of `NoOfDiskPagesToDiskAfterRestartACC` and `NoOfDiskPagesToDiskAfterRestartTUP` is approaching the amount of disk bandwidth available, you may wish to increase the time between local checkpoints.

Given 5 minutes (300 seconds) per local checkpoint, this means that we need to support writing log records at maximum speed for 6 * 300 = 1800 seconds. The size of a REDO log record is 72 bytes plus 4 bytes per updated column value plus the maximum size of the updated column, and there is one REDO log record for each table record updated in a transaction, on each node where the data reside. Using the numbers of operations set out previously in this section, we derive the following:

- 50000 select operations per hour yields 0 log records (and thus 0 bytes), since `SELECT` statements are not recorded in the REDO log.
- 15000 `DELETE` statements per hour is approximately 5 delete operations per second. (Since we wish to be conservative in our estimate, we round up here and in the following calculations.) No columns are updated by deletes, so these statements consume only 5 operations * 72 bytes per operation = 360 bytes per second.
- 15000 `UPDATE` statements per hour is roughly the same as 5 updates per second. Each update uses 72 bytes, plus 4 bytes per column * 5 columns updated, plus 32 bytes per column * 5 columns—this works out to 72 + 20 + 160 = 252 bytes per operation, and multiplying this by 5 operation per second yields 1260 bytes per second.
- 15000 `INSERT` statements per hour is equivalent to 5 insert operations per second. Each insert requires REDO log space of 72 bytes, plus 4 bytes per record * 40 columns, plus 32 bytes per column * 40

This documentation is for an older version. If you're
This documentation is for an older version. If you're

columns, which is 72 + 160 + 1280 = 1512 bytes per operation. This times 5 operations per second yields 7560 bytes per second.

So the total number of REDO log bytes being written per second is approximately 0 + 360 + 1260 + 7560 = 9180 bytes. Multiplied by 1800 seconds, this yields 16524000 bytes required for REDO logging, or approximately 15.75 MB. The unit used for `NoOfFragmentLogFiles` represents a set of 4 16-MB log files—that is, 64 MB. Thus, the minimum value (3) for this parameter is sufficient for the scenario envisioned in this example, since 3 times 64 = 192 MB, or about 12 times what is required; the default value of 8 (or 512 MB) is more than ample in this case.

A copy of each altered table record is kept in the UNDO log. In the scenario discussed above, the UNDO log would not require any more space than what is provided by the default seetings. However, given the size of disks, it is sensible to allocate at least 1 GB for it.

17.3.4 Using High-Speed Interconnects with MySQL Cluster

Even before design of `NDBCLUSTER` began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, `NDBCLUSTER` was designed from the very beginning to permit the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The MySQL Cluster codebase includes support for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in Section 17.3.3.8, “MySQL Cluster TCP/IP Connections”.
- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for MySQL Cluster. See Section 17.3.3.9, “MySQL Cluster TCP/IP Connections Using Direct Connections”, for details.
- *Shared memory (SHM)*. For more information about SHM, see Section 17.3.3.10, “MySQL Cluster Shared-Memory Connections”.

  Note

  SHM is considered experimental only, and is not officially supported.

- *Scalable Coherent Interface (SCI)*, as described in the next section of this chapter, Section 17.3.3.11, “SCI Transport Connections in MySQL Cluster”.

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with MySQL Cluster.

We are working to make sure that communication with the `ndbd` process is made in “chunks” that are as large as possible because this benefits all types of data transmission.

For users who desire it, it is also possible to use cluster interconnects to enhance performance even further. There are two ways to achieve this: Either a custom transporter can be designed to handle this case, or you can use socket implementations that bypass the TCP/IP stack to one extent or another. We have experimented with both of these techniques using the SCI (Scalable Coherent Interface) technology developed by Dolphin Interconnect Solutions.

17.3.4.1 Configuring MySQL Cluster to use SCI Sockets

This documentation is for an older version. If you're
This documentation is for an older version. If you're

It is possible employing Scalable Coherent Interface (SCI) technology to achieve a significant increase in connection speeds and throughput between MySQL Cluster data and SQL nodes. To use SCI, it is necessary to obtain and install Dolphin SCI network cards and to use the drivers and other software supplied by Dolphin. You can get information on obtaining these, from Dolphin Interconnect Solutions. SCI SuperSocket or SCI Transporter support is available for 32-bit and 64-bit Linux, Solaris, and other platforms. See the Dolphin documentation referenced later in this section for more detailed information regarding platforms supported for SCI.

Note

Prior to MySQL 5.0.66, there were issues with building MySQL Cluster with SCI support (see Bug #25470), but these have been resolved due to work contributed by Dolphin. SCI Sockets are now correctly supported for MySQL Cluster hosts running recent versions of Linux using the `-max` builds, and versions of MySQL Cluster with SCI Transporter support can be built using either of `compile-amd64-max-sci` or `compile-pentium64-max-sci`. Both of these build scripts can be found in the `BUILD` directory of the MySQL Cluster source trees; it should not be difficult to adapt them for other platforms. Generally, all that is necessary is to compile MySQL Cluster with SCI Transporter support is to configure the MySQL Cluster build using `--with-ndb-sci=/opt/DIS`.

Once you have acquired the required Dolphin hardware and software, you can obtain detailed information on how to adapt a MySQL Cluster configured for normal TCP/IP communication to use SCI from the from the Dolphin SCI online documentation.

17.3.4.2 MySQL Cluster Interconnects and Performance

The `ndbd` process has a number of simple constructs which are used to access the data in a MySQL Cluster. We have created a very simple benchmark to check the performance of each of these and the effects which various interconnects have on their performance.

There are four access methods:

- **Primary key access.** This is access of a record through its primary key. In the simplest case, only one record is accessed at a time, which means that the full cost of setting up a number of TCP/IP messages and a number of costs for context switching are borne by this single request. In the case where multiple primary key accesses are sent in one batch, those accesses share the cost of setting up the necessary TCP/IP messages and context switches. If the TCP/IP messages are for different destinations, additional TCP/IP messages need to be set up.

- **Unique key access.** Unique key accesses are similar to primary key accesses, except that a unique key access is executed as a read on an index table followed by a primary key access on the table. However, only one request is sent from the MySQL Server, and the read of the index table is handled by `ndbd`. Such requests also benefit from batching.

- **Full table scan.** When no indexes exist for a lookup on a table, a full table scan is performed. This is sent as a single request to the `ndbd` process, which then divides the table scan into a set of parallel scans on all cluster `ndbd` processes. In future versions of MySQL Cluster, an SQL node will be able to filter some of these scans.

- **Range scan using ordered index**

  When an ordered index is used, it performs a scan in the same manner as the full table scan, except that it scans only those records which are in the range used by the query transmitted by the MySQL server

This documentation is for an older version. If you're
This documentation is for an older version. If you're

(SQL node). All partitions are scanned in parallel when all bound index attributes include all attributes in the partitioning key.

With benchmarks developed internally by MySQL for testing simple and batched primary and unique key accesses, we have found that using SCI sockets improves performance by approximately 100% over TCP/IP, except in rare instances when communication performance is not an issue. This can occur when scan filters make up most of processing time or when very large batches of primary key accesses are achieved. In that case, the CPU processing in the `ndbd` processes becomes a fairly large part of the overhead.

Using the SCI transporter instead of SCI Sockets is only of interest in communicating between `ndbd` processes. Using the SCI transporter is also only of interest if a CPU can be dedicated to the `ndbd` process because the SCI transporter ensures that this process will never go to sleep. It is also important to ensure that the `ndbd` process priority is set in such a way that the process does not lose priority due to running for an extended period of time, as can be done by locking processes to CPUs in Linux 2.6. If such a configuration is possible, the `ndbd` process will benefit by 10−70% as compared with using SCI sockets. (The larger figures will be seen when performing updates and probably on parallel scan operations as well.)

There are several other optimized socket implementations for computer clusters, including Myrinet, Gigabit Ethernet, Infiniband and the VIA interface. However, we have tested MySQL Cluster so far only with SCI sockets. See Section 17.3.4.1, “Configuring MySQL Cluster to use SCI Sockets”, for information on how to set up SCI sockets using ordinary TCP/IP for MySQL Cluster.

17.4 MySQL Cluster Programs

Using and managing a MySQL Cluster requires several specialized programs, which we describe in this chapter. We discuss the purposes of these programs in a MySQL Cluster, how to use the programs, and what startup options are available for each of them.

These programs include the MySQL Cluster data, management, and SQL node processes (`ndbd`, `ndb_mgmd`, and `mysqld`) and the management client (`ndb_mgm`).

For information about using `mysqld` as a MySQL Cluster process, see Section 17.5.4, “MySQL Server Usage for MySQL Cluster”.

Other `NDB` utility, diagnostic, and example programs are included with the MySQL Cluster distribution. These include `ndb_restore`, `ndb_show_tables`, and `ndb_config`. These programs are also covered in this section.

The final portion of this section contains tables of options that are common to all the various MySQL Cluster programs.

17.4.1 ndbd — The MySQL Cluster Data Node Daemon

`ndbd` is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In a MySQL Cluster, a set of `ndbd` processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

The following table includes command options specific to the MySQL Cluster data node program `ndbd`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndbd`), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Table 17.26 This table describes command-line options for the ndbd program

Format	Description	Added or Removed
--initial	Perform initial start of ndbd, including cleaning the file system. Consult the documentation before using this option	All MySQL 5.0 based releases
--nostart, -n	Don't start ndbd immediately; ndbd waits for command to start from ndb_mgmd	All MySQL 5.0 based releases
--daemon, -d	Start ndbd as daemon (default); override with --nodaemon	All MySQL 5.0 based releases
--nodaemon	Do not start ndbd as daemon; provided for testing purposes	All MySQL 5.0 based releases
--foreground	Run ndbd in foreground, provided for debugging purposes (implies --nodaemon)	All MySQL 5.0 based releases
--nowait-nodes=list	Do not wait for these data nodes to start (takes comma-separated list of node IDs). Also requires --ndb-nodeid to be used.	ADDED: 5.0.21
--initial-start	Perform partial initial start (requires --nowait-nodes)	ADDED: 5.0.21
--bind-address=name	Local bind address	ADDED: 5.0.29

- `--bind-address`

Introduced	5.0.29	
Command-Line Format	`--bind-address=name`	
Permitted Values	**Type**	`string`
	Default	

Causes `ndbd` to bind to a specific network interface (host name or IP address). This option has no default value.

This option was added in MySQL 5.0.29.

- `--daemon`, `-d`

Command-Line Format	`--daemon`	
Permitted Values	**Type**	`boolean`
	Default	`TRUE`

Instructs `ndbd` to execute as a daemon process. This is the default behavior. `--nodaemon` can be used to prevent the process from running as a daemon.

- `--initial`

Command-Line Format	`--initial`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	Type	boolean
	Default	FALSE

Instructs ndbd to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of ndbd. It also re-creates recovery log files. Note that on some operating systems this process can take a substantial amount of time.

An --initial start is to be used *only* when starting the ndbd process under very special circumstances; this is because this option causes all files to be removed from the Cluster file system and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of ndbd.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

Use of this option prevents the StartPartialTimeout and StartPartitionedTimeout configuration parameters from having any effect.

Important

This option does *not* affect any backup files that have already been created by the affected node.

This option also has no effect on recovery of data by a data node that is just starting (or restarting) from data nodes that are already running. This recovery of data occurs automatically, and requires no user intervention in a MySQL Cluster that is running normally.

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- --initial-start

Introduced	5.0.21	
Command-Line Format	--initial-start	
Permitted Values	Type	boolean
	Default	FALSE

This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as --nowait-nodes.

Suppose that you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5—that is, omitting node 3:

```
shell> ndbd --ndb-nodeid=2 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=4 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=5 --nowait-nodes=3 --initial-start
```

This option was added in MySQL 5.0.21.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--nowait-nodes=node_id_1[, node_id_2[, ...]]`

Introduced	5.0.21	
Command-Line Format	`--nowait-nodes=list`	
Permitted Values	Type	`string`
	Default	

This option takes a list of data nodes which for which the cluster will not wait for before starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each `ndbd` process with `--nowait-nodes=3,5`. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does *not* wait `StartPartitionedTimeout` milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one `ndbd`—say, for example, that the host machine for node 3 has suffered a hardware failure—then start nodes 2, 4, and 5 with `--nowait-nodes=3`. Then the cluster will start as soon as nodes 2, 4, and 5 connect and will not wait for node 3 to start.

When using this option, you must also specify the node ID for the data node being started with the `--ndb-nodeid` option.

This option was added in MySQL 5.0.21.

- `--nodaemon`

Command-Line Format	`--nodaemon`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

Instructs `ndbd` not to start as a daemon process. This is useful when `ndbd` is being debugged and you want output to be redirected to the screen.

- `--foreground`

Command-Line Format	`--foreground`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

Causes `ndbd` to execute as a foreground process, primarily for debugging purposes. This option implies the `--nodaemon` option.

- `--nostart`, `-n`

Command-Line Format	`--nostart`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

Instructs `ndbd` not to start automatically. When this option is used, `ndbd` connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This can be accomplished by issuing the proper START command in the management client (see Section 17.5.2, “Commands in the MySQL Cluster Management Client”).

ndbd generates a set of log files which are placed in the directory specified by DataDir in the config.ini configuration file.

These log files are listed below. *node_id* is the node's unique identifier. Note that *node_id* represents the node's unique identifier. For example, ndb_2_error.log is the error log generated by the data node whose node ID is 2.

- ndb_*node_id*_error.log is a file containing records of all crashes which the referenced ndbd process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

  Listings of possible ndbd exit codes and messages generated when a data node process shuts down prematurely can be found in ndbd Error Messages.

  Important

  The last entry in the error log file is not necessarily the newest one (nor is it likely to be). Entries in the error log are *not* listed in chronological order; rather, they correspond to the order of the trace files as determined in the ndb_*node_id*_trace.log.next file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.

- ndb_*node_id*_trace.log.*trace_id* is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the MySQL Cluster development team.

  It is possible to configure the number of these trace files that will be created before old files are overwritten. *trace_id* is a number which is incremented for each successive trace file.

- ndb_*node_id*_trace.log.next is the file that keeps track of the next trace file number to be assigned.

- ndb_*node_id*_out.log is a file containing any data output by the ndbd process. This file is created only if ndbd is started as a daemon, which is the default behavior.

- ndb_*node_id*.pid is a file containing the process ID of the ndbd process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.

- ndb_*node_id*_signal.log is a file used only in debug versions of ndbd, where it is possible to trace all incoming, outgoing, and internal messages with their data in the ndbd process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the .pid file remains in effect even after the process has terminated.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To start ndbd, it may also be necessary to specify the host name of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See Section 17.3.3.2, “MySQL Cluster Connection Strings”, for additional information about this issue. Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”, describes other command-line options which can be used with ndbd. For information about data node configuration parameters, see Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”.

When ndbd starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the ndbd process if it is configured to do so. Thus, if you attempt to kill ndbd using the Unix kill command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an ndbd process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent actions. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the ndbd process. In a multi-processor system performing a large number of operations (including updates), the ndbd process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several ndbd processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 5.0 in a production setting. See Section 17.1.5, “Known Limitations of MySQL Cluster”.

17.4.2 ndb_mgmd — The MySQL Cluster Management Server Daemon

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

The following table includes options that are specific to the MySQL Cluster management server program ndb_mgmd. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_mgmd), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.27 This table describes command-line options for the ndb_mgmd program

Format	Description	Added or Removed
--config-file=file, -f, -c	Specify the cluster configuration file; in NDB-6.4.0 and later, needs --reload or --initial to override configuration cache if present	All MySQL 5.0 based releases
--print-full-config, -P	Print full configuration and exit	ADDED: 5.0.10
--daemon, -d	Run ndb_mgmd in daemon mode (default)	All MySQL 5.0 based releases
--nodaemon	Do not run ndb_mgmd as a daemon	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
--interactive	Run ndb_mgmd in interactive mode (not officially supported in production; for testing purposes only)	All MySQL 5.0 based releases
--no-nodeid-checks	Do not provide any node id checks	All MySQL 5.0 based releases
--mycnf	Read cluster configuration data from the my.cnf file	All MySQL 5.0 based releases

- --no-nodeid-checks

Command-Line Format	--no-nodeid-checks	
Permitted Values	Type	boolean
	Default	FALSE

Do not perform any checks of node IDs.

- --config-file=*filename*, -f *filename*

Command-Line Format	--config-file=file	
Permitted Values	Type	file name
	Default	./config.ini

Instructs the management server as to which file it should use for its configuration file. By default, the management server looks for a file named config.ini in the same directory as the ndb_mgmd executable; otherwise the file name and location must be specified explicitly.

This option also can be given as -c *file_name*, but this shortcut is obsolete and should *not* be used in new installations.

- --mycnf

Command-Line Format	--mycnf	
Permitted Values	Type	boolean
	Default	FALSE

Read configuration data from the my.cnf file.

- --daemon, -d

Command-Line Format	--daemon	
Permitted Values	Type	boolean
	Default	TRUE

Instructs ndb_mgmd to start as a daemon process. This is the default behavior.

- --interactive

Command-Line Format	--interactive

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	**Type**	boolean
	Default	FALSE

Starts ndb_mgmd in interactive mode; that is, an ndb_mgm client session is started as soon as the management server is running. This option does not start any other MySQL Cluster nodes.

- --nodaemon

Command-Line Format	--nodaemon	
Permitted Values	**Type**	boolean
	Default	FALSE

Instructs ndb_mgmd not to start as a daemon process.

- --print-full-config, -P

Introduced	5.0.10	
Command-Line Format	--print-full-config	
Permitted Values	**Type**	boolean
	Default	FALSE

Shows extended information regarding the configuration of the cluster. With this option on the command line the ndb_mgmd process prints information about the cluster setup including an extensive list of the cluster configuration sections as well as parameters and their values. Normally used together with the --config-file (-f) option.

It is not strictly necessary to specify a connection string when starting the management server. However, if you are using more than one management server, a connection string should be provided and each node in the cluster should specify its node ID explicitly.

See Section 17.3.3.2, “MySQL Cluster Connection Strings”, for information about using connection strings. Section 17.4.2, “ndb_mgmd — The MySQL Cluster Management Server Daemon”, describes other options for ndb_mgmd.

The following files are created or used by ndb_mgmd in its starting directory, and are placed in the DataDir as specified in the config.ini configuration file. In the list that follows, *node_id* is the unique node identifier.

- config.ini is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. Section 17.3, “MySQL Cluster Configuration”, discusses how to set up this file.

- ndb_*node_id*_cluster.log is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in Section 17.5, “Management of MySQL Cluster”.

  By default, when the size of the cluster log reaches one million bytes, the file is renamed to ndb_*node_id*_cluster.log.*seq_id*, where *seq_id* is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number 4.) You can change the size and number of files, and other characteristics of the cluster log, using the LogDestination configuration parameter.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- ndb_*node_id*_out.log is the file used for stdout and stderr when running the management server as a daemon.
- ndb_*node_id*.pid is the process ID file used when running the management server as a daemon.

17.4.3 ndb_mgm — The MySQL Cluster Management Client

The ndb_mgm management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by ndb_mgm.

To start the management client, it is necessary to supply the host name and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default host name and port number are localhost and 1186, respectively.

The following table includes options that are specific to the MySQL Cluster management client program ndb_mgm. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_mgm), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.28 This table describes command-line options for the ndb_mgm program

Format	Description	Added or Removed
--try-reconnect=#, -t	Set the number of times to retry a connection before giving up; synonym for --connect-retries	All MySQL 5.0 based releases
--execute=name, -e	Execute command and exit	All MySQL 5.0 based releases

- --execute=command, -e command

Command-Line Format	--execute=name

This option can be used to send a command to the MySQL Cluster management client from the system shell. For example, either of the following is equivalent to executing SHOW in the management client:

```
shell> ndb_mgm -e "SHOW"

shell> ndb_mgm --execute="SHOW"
```

This is analogous to how the --execute or -e option works with the mysql command-line client. See Section 4.2.4, “Using Options on the Command Line”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

If the management client command to be passed using this option contains any space characters, then the command *must* be enclosed in quotation marks. Either single or double quotation marks may be used. If the management client command contains no space characters, the quotation marks are optional.

- `--try-reconnect=number`

Command-Line Format	--try-reconnect=#	
Permitted Values	Type	integer
	Default	3
	Min Value	0
	Max Value	4294967295

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *number* before giving up and reporting an error instead.

Additional information about using ndb_mgm can be found in Section 17.5.2, “Commands in the MySQL Cluster Management Client”.

17.4.4 ndb_config — Extract MySQL Cluster Configuration Information

This tool extracts configuration information for data nodes, SQL nodes, and API nodes from a cluster management node (and possibly its config.ini file).

The following table includes options that are specific to ndb_config. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_config), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.29 This table describes command-line options for the ndb_config program

Format	Description	Added or Removed
--nodes	Print node information ([ndbd] or [ndbd default] section of cluster configuration file) only. Cannot be used with --system or --connections.	All MySQL 5.0 based releases
--query=string, -q	One or more query options (attributes)	All MySQL 5.0 based releases
--host=name	Specify host	All MySQL 5.0 based releases
--type=name	Specify node type	All MySQL 5.0 based releases
--nodeid, --id	Get configuration of node with this ID	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
--fields=string, -f	Field separator	All MySQL 5.0 based releases
--rows=string, -r	Row separator	All MySQL 5.0 based releases
--config-file=file_name	Set the path to config.ini file	All MySQL 5.0 based releases
--mycnf	Read configuration data from my.cnf file	All MySQL 5.0 based releases
-c	Short form for --ndb-connectstring	ADDED: 5.0.33

Table 17.30 ndb_config Command Line Options

Format	Description	Introduced
--config-file	Set the path to config.ini file	
--fields	Field separator	
--host	Specify host	
--mycnf	Read configuration data from my.cnf file	
--id, --nodeid	Get configuration of node with this ID	
--nodes	Print node information ([ndbd] or [ndbd default] section of cluster configuration file) only. Cannot be used with --system or --connections.	
-c	Short form for --ndb-connectstring	5.0.33
--query	One or more query options (attributes)	
--rows	Row separator	
--type	Specify node type	

- --usage, --help, or -?

Command-Line Format	--help
	--usage

Causes ndb_config to print a list of available options, and then exit.

- --version, -V

Command-Line Format	--version

Causes ndb_config to print a version information string, and then exit.

- --ndb-connectstring=*connection_string*

Command-Line Format	--ndb-connectstring=connectstring	
	--connect-string=connectstring	
Permitted Values	Type	string
	Default	localhost:1186

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Specifies the connection string to use in connecting to the management server. The format for the connection string is the same as described in Section 17.3.3.2, “MySQL Cluster Connection Strings”, and defaults to `localhost:1186`.

The use of `-c` as a short version for this option is supported for `ndb_config` beginning with MySQL 5.0.29.

- `--config-file=path-to-file`

Command-Line Format	`--config-file=file_name`	
Permitted Values	Type	`file name`
	Default	

Gives the path to the management server's configuration file (`config.ini`). This may be a relative or absolute path. If the management node resides on a different host from the one on which `ndb_config` is invoked, then an absolute path must be used.

- `--mycnf`

Command-Line Format	`--mycnf`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

Read configuration data from the `my.cnf` file.

- `--query=query-options`, `-q query-options`

Command-Line Format	`--query=string`	
Permitted Values	Type	`string`
	Default	

This is a comma-delimited list of *query options*—that is, a list of one or more node attributes to be returned. These include `id` (node ID), type (node type—that is, `ndbd`, `mysqld`, or `ndb_mgmd`), and any configuration parameters whose values are to be obtained.

For example, `--query=id,type,indexmemory,datamemory` returns the node ID, node type, `DataMemory`, and `IndexMemory` for each node.

Note

If a given parameter is not applicable to a certain type of node, than an empty string is returned for the corresponding value. See the examples later in this section for more information.

- `--host=hostname`

Command-Line Format	`--host=name`	
Permitted Values	Type	`string`
	Default	

Specifies the host name of the node for which configuration information is to be obtained.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

While the hostname `localhost` usually resolves to the IP address `127.0.0.1`, this may not necessarily be true for all operating platforms and configurations. This means that it is possible, when `localhost` is used in `config.ini`, for `ndb_config --host=localhost` to fail if `ndb_config` is run on a different host where `localhost` resolves to a different address (for example, on some versions of SUSE Linux, this is `127.0.0.2`). In general, for best results, you should use numeric IP addresses for all MySQL Clustewr configuration values relating to hosts, or verify that all MySQL Cluster hosts handle `localhost` in the same fashion.

- `--id=node_id`

  `--nodeid=node_id`

Command-Line Format	`--ndb-nodeid=#`	
Permitted Values	Type	`numeric`
	Default	`0`

  Either of these options can be used to specify the node ID of the node for which configuration information is to be obtained. `--nodeid` is the preferred form.

- `--nodes`

Command-Line Format	`--nodes`	
Permitted Values	Type	`boolean`
	Default	`FALSE`

  In MySQL 5.0, this option has no affect.

- `--connections`

  This option (not listed in the table) is present but has no affect in MySQL 5.0. It is reserved for use in a future version of MySQL Cluster.

- `--type=node_type`

Command-Line Format	`--type=name`	
Permitted Values	Type	`enumeration`
	Default	`[none]`
	Valid Values	`ndbd`
		`mysqld`
		`ndb_mgmd`

  Filters results so that only configuration values applying to nodes of the specified *`node_type`* (`ndbd`, `mysqld`, or `ndb_mgmd`) are returned.

- `--fields=delimiter`, `-f delimiter`

Command-Line Format	`--fields=string`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Permitted Values	Type	string
	Default	

Specifies a *delimiter* string used to separate the fields in the result. The default is ",” (the comma character).

Note

If the *delimiter* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

- --rows=*separator*, -r *separator*

Command-Line Format	--rows=string	
Permitted Values	Type	string
	Default	

Specifies a *separator* string used to separate the rows in the result. The default is a space character.

Note

If the *separator* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

Examples

1. To obtain the node ID and type of each node in the cluster:

```
shell> ./ndb_config --query=id,type --fields=':' --rows='\n'
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqld
7:mysqld
8:mysqld
9:mysqld
```

   In this example, we used the --fields options to separate the ID and type of each node with a colon character (:), and the --rows options to place the values for each node on a new line in the output.

2. To produce a connection string that can be used by data, SQL, and API nodes to connect to the management server:

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini \
--query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
192.168.0.179:1186
```

3. This invocation of ndb_config checks only data nodes (using the --type option), and shows the values for each node's ID and host name, and its DataMemory, IndexMemory, and DataDir parameters:

```
shell> ./ndb_config --type=ndbd --query=id,host,datamemory,indexmemory,datadir -f ' : ' -r '\n'
1 : 192.168.0.193 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
2 : 192.168.0.112 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
3 : 192.168.0.176 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
4 : 192.168.0.119 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively.

4. To exclude results from any host except one in particular, use the `--host` option:

```
shell> ./ndb_config --host=192.168.0.176 -f : -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--id` or `--nodeid` option.

17.4.5 ndb_cpcd — Automate Testing for NDB Development

This utility is found in the `libexec` directory. It is part of an internal automated test framework used in testing and debugging MySQL Cluster. Because it can control processes on remote systems, it is not advisable to use `ndb_cpcd` in a production cluster.

The source files for `ndb_cpcd` may be found in the directory `storage/ndb/src/cw/cpcd`, in the MySQL source tree.

17.4.6 ndb_delete_all — Delete All Rows from an NDB Table

`ndb_delete_all` deletes all rows from the given `NDB` table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE TABLE`.

Usage

```
ndb_delete_all -c connection_string tbl_name -d db_name
```

This deletes all rows from the table named *`tbl_name`* in the database named *`db_name`*. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

The following table includes options that are specific to `ndb_delete_all`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_delete_all`), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.31 This table describes command-line options for the ndb_delete_all program

Format	Description	Added or Removed
`--database=dbname`, `-d`	Name of the database in which the table is found	All MySQL 5.0 based releases
`--transactional`, `-t`	Perform the delete in a single transaction (may run out of operations)	All MySQL 5.0 based releases
`--tupscan`	Run tup scan	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
--diskscan	Run disk scan	All MySQL 5.0 based releases

- --transactional, -t

  Use of this option causes the delete operation to be performed as a single transaction.

Warning

With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.

17.4.7 ndb_desc — Describe NDB Tables

ndb_desc provides a detailed description of one or more NDB tables.

Usage

```
ndb_desc -c connection_string tbl_name -d db_name [options]
```

Additional options that can be used with ndb_desc are listed later in this section.

Sample Output

MySQL table creation and population statements:

```
USE test;

CREATE TABLE fish (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20),

    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) ENGINE=NDBCLUSTER;

INSERT INTO fish VALUES
    ('','guppy'), ('','tuna'), ('','shark'),
    ('','manta ray'), ('','grouper'), ('','puffer');
```

Output from ndb_desc:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 16777221
Fragment type: 5
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 268
Row Checksum: 1
Row GCI: 1
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
name Varchar(20;latin1_swedish_ci) NULL AT=SHORT_VAR ST=MEMORY
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
uk(name) - OrderedIndex
PRIMARY(id) - OrderedIndex
uk$unique(name) - UniqueHashIndex

-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory  Frag varsized memory
2          2          2             65536              327680
1          2          2             65536              327680
3          2          2             65536              327680

NDBT_ProgramExit: 0 - OK
```

Information about multiple tables can be obtained in a single invocation of ndb_desc by using their names, separated by spaces. All of the tables must be in the same database.

The Version column in the output contains the table's schema object version. For information about interpreting this value, see NDB Schema Object Versions.

The following table includes options that are specific to ndb_desc. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_desc), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.32 This table describes command-line options for the ndb_desc program

Format	Description	Added or Removed
--database=dbname, -d	Name of database containing table	All MySQL 5.0 based releases
--extra-partition-info, -p	Display information about partitions	All MySQL 5.0 based releases
--retries=#, -r	Number of times to retry the connection (once per second)	All MySQL 5.0 based releases
--unqualified, -u	Use unqualified table names	All MySQL 5.0 based releases

- --database=*db_name*, -d

  Specify the database in which the table should be found.

- --extra-partition-info, -p

  Print additional information about the table's partitions.

- --retries=#, -r

  Try to connect this many times before giving up. One connect attempt is made per second.

- --unqualified, -u

  Use unqualified table names.

17.4.8 ndb_drop_index — Drop Index from an NDB Table

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ndb_drop_index drops the specified index from an NDB table. *It is recommended that you use this utility only as an example for writing NDB API applications*—see the Warning later in this section for details.

Usage

```
ndb_drop_index -c connection_string table_name index -d db_name
```

The statement shown above drops the index named *index* from the *table* in the *database*.

The following table includes options that are specific to ndb_drop_index. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_drop_index), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.33 This table describes command-line options for the ndb_drop_index program

Format	Description	Added or Removed
--database=dbname, -d	Name of the database in which the table is found	All MySQL 5.0 based releases

Warning

Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *******
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.0.96

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+------------------+
| Tables_in_ctest1 |
+------------------+
| a                |
| bt1              |
| bt2              |
| dogs             |
| employees        |
| fish             |
+------------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statement`DROP TABLE` or the `ndb_drop_table` utility (see Section 17.4.9, “`ndb_drop_table` — Drop an NDB Table”) to drop the table.

17.4.9 `ndb_drop_table` — Drop an NDB Table

`ndb_drop_table` drops the specified `NDB` table. (If you try to use this on a table created with a storage engine other than `NDB`, the attempt fails with the error `723: No such table exists`.) This operation is extremely fast; in some cases, it can be an order of magnitude faster than using a MySQL `DROP TABLE` statement on an `NDB` table.

Usage

```
ndb_drop_table -c connection_string tbl_name -d db_name
```

The following table includes options that are specific to `ndb_drop_table`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_drop_table`), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.34 This table describes command-line options for the ndb_drop_table program

Format	Description	Added or Removed
`--database=dbname`, `-d`	Name of the database in which the table is found	All MySQL 5.0 based releases

17.4.10 `ndb_error_reporter` — NDB Error-Reporting Utility

`ndb_error_reporter` creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in MySQL Cluster*.

The following table includes command options specific to the MySQL Cluster program `ndb_error_reporter`. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including `ndb_error_reporter`), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.35 This table describes command-line options for the ndb_error_reporter program

Format	Description	Added or Removed
`--fs`	Include file system data in error report; can use a large amount of disk space	All MySQL 5.0 based releases

Usage

```
ndb_error_reporter path/to/config-file [username] [--fs]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (`config.ini`). Optionally, you can supply the name of a user that is able to access the cluster's data nodes using SSH, to copy the data node log files. ndb_error_reporter then includes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

all of these files in archive that is created in the same directory in which it is run. The archive is named ndb_error_report_*YYYYMMDDHHMMSS*.tar.bz2, where *YYYYMMDDHHMMSS* is a datetime string.

If the --fs is used, then the data node file systems are also copied to the management host and included in the archive that is produced by this script. As data node file systems can be extremely large even after being compressed, we ask that you please do *not* send archives created using this option to Oracle unless you are specifically requested to do so.

Command-Line Format	--fs	
Permitted Values	Type	boolean
	Default	FALSE

17.4.11 ndb_print_backup_file — Print NDB Backup File Contents

ndb_print_backup_file obtains diagnostic information from a cluster backup file.

Usage

```
ndb_print_backup_file file_name
```

file_name is the name of a cluster backup file. This can be any of the files (.Data, .ctl, or .log file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory BACKUP-#, where # is the sequence number for the backup. For more information about cluster backup files and their contents, see Section 17.5.3.1, “MySQL Cluster Backup Concepts”.

Like ndb_print_schema_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

17.4.12 ndb_print_schema_file — Print NDB Schema File Contents

ndb_print_schema_file obtains diagnostic information from a cluster schema file.

Usage

```
ndb_print_schema_file file_name
```

file_name is the name of a cluster schema file. For more information about cluster schema files, see MySQL Cluster Data Node File System Directory Files.

Like ndb_print_backup_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_schema_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Additional Options

None.

17.4.13 ndb_print_sys_file — Print NDB System File Contents

ndb_print_sys_file obtains diagnostic information from a MySQL Cluster system file.

Usage

```
ndb_print_sys_file file_name
```

file_name is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (DataDir); the path under this directory to system files matches the pattern ndb_#_fs/D#/DBDIH/P#.sysfile. In each case, the # represents a number (not necessarily the same number). For more information, see MySQL Cluster Data Node File System Directory Files.

Like ndb_print_backup_file and ndb_print_schema_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

17.4.14 ndb_restore — Restore a MySQL Cluster Backup

The cluster restoration program is implemented as a separate command-line utility ndb_restore, which can normally be found in the MySQL bin directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

ndb_restore must be executed once for each of the backup files that were created by the START BACKUP command used to create the backup (see Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”). This is equal to the number of data nodes in the cluster at the time that the backup was created.

Note

Before using ndb_restore, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See Section 17.5.8, “MySQL Cluster Single User Mode”, for more information.

The following table includes options that are specific to the MySQL Cluster native backup restoration program ndb_restore. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_restore), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.36 This table describes command-line options for the ndb_restore program

Format	Description	Added or Removed
--connect,	Alias for --connectstring.	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
-c --nodeid=#, -n	Restore backup files to node with this ID	All MySQL 5.0 based releases
--backupid=#, -b	Restore from the backup with the given ID	All MySQL 5.0 based releases
--restore_data, -r	Restore table data and logs into NDB Cluster using the NDB API	All MySQL 5.0 based releases
--restore_meta, -m	Restore metadata to NDB Cluster using the NDB API	All MySQL 5.0 based releases
--parallelism=#, -p	Number of parallel transactions to use while restoring data	All MySQL 5.0 based releases
--print	Print metadata, data and log to stdout (equivalent to --print_meta --print_data --print_log)	All MySQL 5.0 based releases
--print_meta	Print metadata to stdout	All MySQL 5.0 based releases
--print_data	Print data to stdout	All MySQL 5.0 based releases
--print_log	Print to stdout	All MySQL 5.0 based releases
--backup_path=dir_name	Path to backup files directory	ADDED: 5.0.38
--dont_ignore_systab_0, -f	Do not ignore system table during restore. Experimental only; not for production use	All MySQL 5.0 based releases
--fields-enclosed-by=char	Fields are enclosed with the indicated character	ADDED: 5.0.40
--fields-terminated-by=char	Fields are terminated by the indicated character	ADDED: 5.0.40
--fields-optionally-enclosed-by	Fields are optionally enclosed with the indicated character	ADDED: 5.0.40
--lines-terminated-by=char	Lines are terminated by the indicated character	ADDED: 5.0.40
--hex	Print binary types in hexadecimal format	ADDED: 5.0.40
--tab=dir_name, -T dir_name	Creates a tab-separated .txt file for each table in the given path	ADDED: 5.0.40
--append	Append data to a tab-delimited file	ADDED: 5.0.40
--verbose=#	Level of verbosity in output	All MySQL 5.0 based releases

Typical options for this utility are shown here:

```
ndb_restore [-c connection_string] -n node_id -b backup_id \
      [-m] -r --backup_path=/path/to/backup/files
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Normally, when restoring from a MySQL Cluster backup, ndb_restore requires at a minimum the --nodeid (short form: -n), --backupid (short form: -b), and --backup_path options.

The -c option is used to specify a connection string which tells ndb_restore where to locate the cluster management server. (See Section 17.3.3.2, “MySQL Cluster Connection Strings”, for information on connection strings.) If this option is not used, then ndb_restore attempts to connect to a management server on localhost:1186. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one [api] or [mysqld] section that can be used by it in the cluster config.ini file. It is a good idea to keep at least one empty [api] or [mysqld] section in config.ini that is not being used for a MySQL server or other application for this reason (see Section 17.3.3.6, “Defining SQL and Other API Nodes in a MySQL Cluster”).

You can verify that ndb_restore is connected to the cluster by using the SHOW command in the ndb_mgm management client. You can also accomplish this from a system shell, as shown here:

```
shell> ndb_mgm -e "SHOW"
```

The --nodeid or -n is used to specify the node ID of the data node on which the backup should be restored.

The first time you run the ndb_restore restoration program, you also need to restore the metadata. In other words, you must re-create the database tables—this can be done by running it with the --restore_meta (-m) option. Restoring the metdata need be done only on a single data node; this is sufficient to restore it to the entire cluster. Note that the cluster should have an empty database when starting to restore a backup. (In other words, you should start ndbd with --initial prior to performing the restore.)

The -b option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the Backup *backup_id* completed message displayed upon completion of a backup. (See Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”.)

Important

When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups will at best result in restoring the cluster to an inconsistent state, and may fail altogether.

The path to the backup directory is required; this is supplied to ndb_restore using the --backup_path option, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's DataDir is /var/lib/mysql-cluster, then the backup directory is /var/lib/mysql-cluster/BACKUP, and the backup files for the backup with the ID 3 can be found in /var/lib/mysql-cluster/BACKUP/BACKUP-3. The path may be absolute or relative to the directory in which the ndb_restore executable is located, and may be optionally prefixed with backup_path=.

Note

Previous to MySQL 5.0.38, the path to the backup directory was specified as shown here, with backup_path= being optional:

```
[backup_path=]/path/to/backup/files
```

Beginning with MySQL 5.0.38, this syntax changed to --backup_path=*/path/to/backup/files*, to conform more closely with options used by other MySQL programs; --backup_id is required, and there is no short form for this option.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID 12, created in a cluster with two database nodes having the node IDs 2 and 3, is to be restored to a cluster with four nodes. Then ndb_restore must be run twice—once for each database node in the cluster where the backup was taken. However, ndb_restore cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See Section 17.2.6, “Upgrading and Downgrading MySQL Cluster”, for more information.

Important

It is not possible to restore a backup made from a newer version of MySQL Cluster using an older version of ndb_restore. You can restore a backup made from a newer version of MySQL to an older cluster, but you must use a copy of ndb_restore from the newer MySQL Cluster version to do so.

For example, to restore a cluster backup taken from a cluster running MySQL 5.0.45 to a cluster running MySQL Cluster 5.0.41, you must use a copy of ndb_restore from the 5.0.45 distribution.

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an [api] or [mysqld] section in the cluster config.ini file available for each concurrent ndb_restore process. However, the data files must always be applied before the logs.

--dont_ignore_systab_0

Normally, when restoring table data and metadata, ndb_restore ignores the copy of the NDB system table that is present in the backup. --dont_ignore_systab_0 causes the system table to be restored. *This option is intended for experimental and development use only, and is not recommended in a production environment*.

--parallelism=#, -p

Determines the maximum number of parallel transactions that ndb_restore tries to use. By default, this is 128; the minimum is 1, and the maximum is 1024.

--restore_data

This option causes ndb_restore to output NDB table data and logs.

--restore_meta

This option causes ndb_restore to print NDB table metadata. Generally, you need only use this option when restoring the first data node of a cluster; additional data nodes can obtain the metadata from the first one.

--print_meta

This option causes ndb_restore to print all metadata to stdout.

--print_log

The --print_log option causes ndb_restore to output its log to stdout.

--print

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Causes ndb_restore to print all data, metadata, and logs to stdout. Equivalent to using the --print_data, --print_meta, and --print_log options together.

Note

Use of --print or any of the --print_* options is in effect performing a dry run. Including one or more of these options causes any output to be redirected to stdout; in such cases, ndb_restore makes no attempt to restore data or metadata to a MySQL Cluster.

--print_data

This option causes ndb_restore to direct its output to stdout.

TEXT and BLOB column values are always truncated to the first 256 bytes in the output; this cannot currrently be overridden when using --print_data.

Beginning with MySQL 5.0.40, several additional options are available for use with the --print_data option in generating data dumps, either to stdout, or to a file. These are similar to some of the options used with mysqldump, and are shown in the following list:

- --tab, -T

Introduced	5.0.40	
Command-Line Format	--tab=dir_name	
Permitted Values	**Type**	directory name

This option causes --print_data to create dump files, one per table, each named *tbl_name*.txt. It requires as its argument the path to the directory where the files should be saved; use . for the current directory.

- --fields-enclosed-by=*string*

Introduced	5.0.40	
Command-Line Format	--fields-enclosed-by=char	
Permitted Values	**Type**	string
	Default	

Each column values are enclosed by the string passed to this option (regardless of data type; see next item).

- --fields-optionally-enclosed-by=*string*

Introduced	5.0.40	
Command-Line Format	--fields-optionally-enclosed-by	
Permitted Values	**Type**	string
	Default	

The string passed to this option is used to enclose column values containing character data (such as CHAR, VARCHAR, BINARY, TEXT, or ENUM).

- --fields-terminated-by=*string*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Introduced	5.0.40	
Command-Line Format	--fields-terminated-by=char	
Permitted Values	Type	string
	Default	\t (tab)

The string passed to this option is used to separate column values. The default value is a tab character (\t).

- --hex

Introduced	5.0.40
Command-Line Format	--hex

If this option is used, all binary values are output in hexadecimal format.

- --fields-terminated-by=*string*

Introduced	5.0.40	
Command-Line Format	--fields-terminated-by=char	
Permitted Values	Type	string
	Default	\t (tab)

This option specifies the string used to end each line of output. The default is a linefeed character (\n).

- --append

Introduced	5.0.40
Command-Line Format	--append

When used with the --tab and --print_data options, this causes the data to be appended to any existing files having the same names.

Note

If a table has no explicit primary key, then the output generated when using the --print_data option includes the table's hidden primary key.

--verbose=#

Sets the level for the verbosity of the output. The minimum is 0; the maximum is 255. The default value is 1.

Beginning with MySQL 5.0.40, it is possible to restore selected databases, or to restore selected tables from a given database using the syntax shown here:

```
ndb_restore other_options db_name,[db_name[,...] | tbl_name[,tbl_name][,...]]
```

In other words, you can specify either of the following to be restored:

- All tables from one or more databases
- One or more tables from a single database

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Error reporting.
ndb_restore reports both temporary and permanent errors. In the case of temporary errors, it may able to recover from them. Beginning with MySQL 5.0.29, it reports `Restore successful, but encountered temporary error, please look at configuration` in such cases.

17.4.15 ndb_select_all — Print Rows from an NDB Table

ndb_select_all prints all rows from an NDB table to stdout.

Usage

```
ndb_select_all -c connection_string tbl_name -d db_name [> file_name]
```

The following table includes options that are specific to the MySQL Cluster native backup restoration program ndb_select_all. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_select_all), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.37 This table describes command-line options for the ndb_select_all program

Format	Description	Added or Removed
--database=dbname, -d	Name of the database in which the table is found	All MySQL 5.0 based releases
--parallelism=#, -p	Degree of parallelism	All MySQL 5.0 based releases
--lock=#, -l	Lock type	All MySQL 5.0 based releases
--order=index, -o	Sort resultset according to index whose name is supplied	All MySQL 5.0 based releases
--descending, -z	Sort resultset in descending order (requires order flag)	All MySQL 5.0 based releases
--header, -h	Print header (set to 0|FALSE to disable headers in output)	All MySQL 5.0 based releases
--useHexFormat, -x	Output numbers in hexadecimal format	All MySQL 5.0 based releases
--delimiter=char, -D	Set a column delimiter	All MySQL 5.0 based releases
--disk	Print disk references (useful only for Disk Data tables having nonindexed columns)	All MySQL 5.0 based releases
--rowid	Print rowid	All MySQL 5.0 based releases
--gci	Include GCI in output	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
--gci64	Include GCI and row epoch in output	All MySQL 5.0 based releases
--tup, -t	Scan in tup order	All MySQL 5.0 based releases
--nodata	Do not print table column data	All MySQL 5.0 based releases

- --database=*dbname*, -d *dbname*

  Name of the database in which the table is found. The default value is TEST_DB.

- parallelism=#, -p #

  Specifies the degree of parallelism.

- --lock=*lock_type*, -l *lock_type*

  Employs a lock when reading the table. Possible values for *lock_type* are:

  - 0: Read lock
  - 1: Read lock with hold
  - 2: Exclusive read lock

  There is no default value for this option.

- --order=*index_name*, -o *index_name*

  Orders the output according to the index named *index_name*. Note that this is the name of an index, not of a column, and that the index must have been explicitly named when created.

- --descending, -z

  Sorts the output in descending order. This option can be used only in conjunction with the -o (--order) option.

- --header=FALSE

  Excludes column headers from the output.

- --useHexFormat -x

  Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.

- --delimiter=*character*, -D *character*

  Causes the *character* to be used as a column delimiter. Only table data columns are separated by this delimiter.

  The default delimiter is the tab character.

- --rowid

  Adds a ROWID column providing information about the fragments in which rows are stored.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `--gci`

  Adds a `GCI` column to the output showing the global checkpoint at which each row was last updated. See Section 17.1, “MySQL Cluster Overview”, and Section 17.5.6.2, “MySQL Cluster Log Events”, for more information about checkpoints.

- `--gci64`

  Adds a `ROW$GCI64` column to the output showing the global checkpoint at which each row was last updated, as well as the number of the epoch in which this update occurred.

- `--tupscan`, `-t`

  Scan the table in the order of the tuples.

- `--nodata`

  Causes any table data to be omitted.

Sample Output

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
+----+-----------+
| id | name      |
+----+-----------+
|  3 | shark     |
|  6 | puffer    |
|  2 | tuna      |
|  4 | manta ray |
|  5 | grouper   |
|  1 | guppy     |
+----+-----------+
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of `ndb_select_all`:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

Note that all string values are enclosed by square brackets (“`[...]`”) in the output of `ndb_select_all`. For a further example, consider the table created and populated as shown here:

```
CREATE TABLE dogs (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(25) NOT NULL,
    breed VARCHAR(50) NOT NULL,
    PRIMARY KEY pk (id),
    KEY ix (name)
)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
    ('', 'Lassie', 'collie'),
    ('', 'Scooby-Doo', 'Great Dane'),
    ('', 'Rin-Tin-Tin', 'Alsatian'),
    ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional ndb_select_all options:

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci
GCI     id name           breed
834461  2  [Scooby-Doo]   [Great Dane]
834878  4  [Rosscoe]      [Mutt]
834463  3  [Rin-Tin-Tin]  [Alsatian]
835657  1  [Lassie]       [Collie]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

17.4.16 ndb_select_count — Print Row Counts for NDB Tables

ndb_select_count prints the number of rows in one or more NDB tables. With a single table, the result is equivalent to that obtained by using the MySQL statement SELECT COUNT(*) FROM *tbl_name*.

Usage

```
ndb_select_count [-c connection_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

The following table includes options that are specific to the MySQL Cluster native backup restoration program ndb_select_count. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_select_count), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.38 This table describes command-line options for the ndb_select_count program

Format	Description	Added or Removed
--database=dbname, -d	Name of the database in which the table is found	All MySQL 5.0 based releases
--parallelism=#, -p	Degree of parallelism	All MySQL 5.0 based releases
--lock=#, -l	Lock type	All MySQL 5.0 based releases

You can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

Sample Output

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
NDBT_ProgramExit: 0 - OK
```

17.4.17 ndb_show_tables — Display List of NDB Tables

ndb_show_tables displays a list of all NDB database objects in the cluster. By default, this includes not only both user-created tables and NDB system tables, but NDB-specific indexes, and internal triggers, as well.

The following table includes options that are specific to the MySQL Cluster native backup restoration program ndb_show_tables. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_show_tables), see Section 17.4.20, "Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs".

Table 17.39 This table describes command-line options for the ndb_show_tables program

Format	Description	Added or Removed
--database=string, -d	Specifies the database in which the table is found	All MySQL 5.0 based releases
--loops=#, -l	Number of times to repeat output	All MySQL 5.0 based releases
--type=#, -t	Limit output to objects of this type	All MySQL 5.0 based releases
--unqualified, -u	Do not qualify table names	All MySQL 5.0 based releases
--parsable, -p	Return output suitable for MySQL LOAD DATA INFILE statement	All MySQL 5.0 based releases
--show-temp-status	Show table temporary flag	All MySQL 5.0 based releases

Usage

```
ndb_show_tables [-c connection_string]
```

- --database, -d

  Specifies the name of the database in which the tables are found.

- --loops, -l

  Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- --parsable, -p

  Using this option causes the output to be in a format suitable for use with LOAD DATA INFILE.

- --show-temp-status

  If specified, this causes temporary tables to be displayed.

- --type, -t

This documentation is for an older version. If you're

Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

- 1: System table
- 2: User-created table
- 3: Unique hash index

Any other value causes all NDB database objects to be listed (the default).

- --unqualified, -u

  If specified, this causes unqualified object names to be displayed.

Note

Only user-created MySQL Cluster tables may be accessed from MySQL; system tables such as SYSTAB_0 are not visible to mysqld. However, you can examine the contents of system tables using NDB API applications such as ndb_select_all (see Section 17.4.15, "ndb_select_all — Print Rows from an NDB Table").

17.4.18 ndb_size.pl — NDBCLUSTER Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the NDBCLUSTER storage engine. Unlike the other utilities discussed in this section, it does not require access to a MySQL Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

Requirements

- A running MySQL server. The server instance does not have to provide support for MySQL Cluster.
- A working installation of Perl.
- The DBI and HTML::Template modules, both of which can be obtained from CPAN if they are not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for one or both of these libraries.)
- The ndb_size.tmpl template file, which you should be able to find in the share/mysql directory of your MySQL installation. This file should be copied or moved into the same directory as ndb_size.pl—if it is not there already—before running the script.
- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using GRANT USAGE ON *db_name*.*—where *db_name* is the name of the database to be examined—is sufficient for this purpose.

ndb_size.pl and ndb_size.tmpl can also be found in the MySQL sources in storage/ndb/tools.

Usage

```
perl ndb_size.pl db_name hostname username password > file_name.html
```

The command shown connects to the MySQL server at *hostname* using the account of the user *username* having the password *password*, analyzes all of the tables in database *db_name*, and

This documentation is for an older version. If you're

generates a report in HTML format which is directed to the file *file_name*.html. (Without the redirection, the output is sent to stdout.)

The output from this script includes the following information:

- Minimum values for the DataMemory, IndexMemory, MaxNoOfTables, MaxNoOfAttributes, MaxNoOfOrderedIndexes, MaxNoOfUniqueHashIndexes, and MaxNoOfTriggers configuration parameters required to accommodate the tables analyzed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The IndexMemory and DataMemory required per table and table row.

17.4.19 ndb_waiter — Wait for MySQL Cluster to Reach a Given Status

ndb_waiter repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the --timeout limit is exceeded, then exits. By default, it waits for the cluster to achieve STARTED status, in which all nodes have started and connected to the cluster. This can be overridden using the --no-contact and --not-started options (see Additional Options).

The node states reported by this utility are as follows:

- NO_CONTACT: The node cannot be contacted.
- UNKNOWN: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a START or RESTART command from the management server, but has not yet acted on it.
- NOT_STARTED: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's RESTART command.
- STARTING: The node's ndbd process has started, but the node has not yet joined the cluster.
- STARTED: The node is operational, and has joined the cluster.
- SHUTTING_DOWN: The node is shutting down.
- SINGLE USER MODE: This is shown for all cluster data nodes when the cluster is in single user mode.

The following table includes options that are specific to the MySQL Cluster native backup restoration program ndb_waiter. Additional descriptions follow the table. For options common to most MySQL Cluster programs (including ndb_waiter), see Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 17.40 This table describes command-line options for the ndb_waiter program

Format	Description	Added or Removed
--no-contact, -n	Wait for cluster to reach NO CONTACT state	All MySQL 5.0 based releases
--not-started	Wait for cluster to reach NOT STARTED state	All MySQL 5.0 based releases
--single-user	Wait for cluster to enter single user mode	All MySQL 5.0 based releases
--timeout=#,	Wait this many seconds, then exit whether or not cluster has	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
-t	reached desired state; default is 2 minutes (120 seconds)	

Usage

```
ndb_waiter [-c connection_string]
```

Additional Options

- --no-contact, -n

  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NO_CONTACT status before exiting.

- --not-started

  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NOT_STARTED status before exiting.

- --timeout=*seconds*, -t *seconds*

  Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

- --single-user

  The program waits for the cluster to enter single user mode.

Sample Output. Shown here is the output from ndb_waiter when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by "...") are omitted.

```
shell> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED

NDBT_ProgramExit: 0 - OK
```

Note

If no connection string is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

17.4.20 Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs

All MySQL Cluster programs accept the options described in this section, with the following exceptions:

- `mysqld`
- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

Users of earlier MySQL Cluster versions should note that some of these options have been changed to make them consistent with one another as well as with `mysqld`. You can use the `--help` option with any MySQL Cluster program—with the exception of `ndb_print_backup_file`, `ndb_print_schema_file`, and `ndb_print_sys_file`—to view a list of the options which the program supports.

The options in the following table are common to all MySQL Cluster executables (except those noted previously in this section).

Table 17.41 This table describes command-line options common to all MySQL Cluster programs

Format	Description	Added or Removed
`--help`,	Display help message and exit	All MySQL 5.0 based releases

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Format	Description	Added or Removed
`--usage`, `-?`		
`--character-sets-dir=dir_name`	Directory where character sets are installed	All MySQL 5.0 based releases
`--core-file`	Write core on errors (defaults to TRUE in debug builds)	All MySQL 5.0 based releases
`--debug=options`	Enable output from debug calls. Can be used only for versions compiled with debugging enabled	All MySQL 5.0 based releases
`--ndb-connectstring=connectstring`, `--connect-string=connectstring`, `-c`	Set connection string for connecting to ndb_mgmd. Syntax: [nodeid=<id>;] [host=]<hostname>[:<port>]. Overrides entries specified in NDB_CONNECTSTRING or my.cnf.	All MySQL 5.0 based releases
`--ndb-mgmd-host=host[:port]`	Set the host (and port, if desired) for connecting to management server	All MySQL 5.0 based releases
`--ndb-nodeid=#`	Set node id for this node	All MySQL 5.0 based releases
`--ndb-optimized-node-selection`	Select nodes for transactions in a more optimal way	All MySQL 5.0 based releases
`--ndb-shm`	Allow for optimization using shared memory connections where available (was EXPERIMENTAL, later REMOVED)	All MySQL 5.0 based releases
`--version`, `-V`	Output version information and exit	All MySQL 5.0 based releases

For options specific to individual MySQL Cluster programs, see Section 17.4, “MySQL Cluster Programs”.

See mysqld Command Options for MySQL Cluster, for `mysqld` options relating to MySQL Cluster.

- `--help --usage`, `-?`

Command-Line Format	`--help`
	`--usage`

Prints a short list with descriptions of the available command options.

- `--ndb-connectstring=`*`connection_string`*, `--connect-string=`*`connection_string`*, `-c` *`connection_string`*

Command-Line Format	`--ndb-connectstring=connectstring`	
	`--connect-string=connectstring`	
Permitted Values	**Type**	`string`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

	Default	`localhost:1186`

This option takes a MySQL Cluster connection string that specifies the management server for the application to connect to, as shown here:

```
shell> ndbd --ndb-connectstring="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see Section 17.3.3.2, “MySQL Cluster Connection Strings”.

- `--ndb-mgmd-host=host[:port]`

Command-Line Format	`--ndb-mgmd-host=host[:port]`	
Permitted Values	**Type**	`string`
	Default	`localhost:1186`

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--character-sets-dir=name`

Command-Line Format	`--character-sets-dir=dir_name`	
Permitted Values	**Type**	`directory name`
	Default	

Tells the program where to find character set information.

- `--connect-string=connection_string`, `-c connection_string`

Command-Line Format	`--ndb-connectstring=connectstring`	
	`--connect-string=connectstring`	
Permitted Values	**Type**	`string`
	Default	`localhost:1186`

`connection_string` sets the connection string to the management server as a command option.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see Section 17.3.3.2, “MySQL Cluster Connection Strings”.

- `--core-file`

Command-Line Format	`--core-file`	
Permitted Values	**Type**	`boolean`
	Default	`FALSE`

Write a core file if the program dies. The name and location of the core file are system-dependent. (For MySQL Cluster programs nodes running on Linux, the default location is the program's working directory—for a data node, this is the node's `DataDir`.) For some systems, there may be restrictions or limitations; for example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation for detailed information.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If MySQL Cluster was built using the `--debug` option for `configure`, then `--core-file` is enabled by default. For regular builds, `--core-file` is disabled by default.

- `--debug[=options]`

Command-Line Format	--debug=options	
Permitted Values	Type	string
	Default	d:t:O,/tmp/ndb_restore.trace

This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the `mysqld` process.

- `--ndb-nodeid=#`

Command-Line Format	--ndb-nodeid=#	
Permitted Values	Type	numeric
	Default	0

Sets this node's MySQL Cluster node ID. *The range of permitted values depends on the type of the node (data, management, or API) and the version of the MySQL Cluster software which is running on it*. See Section 17.1.5.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”, for more information.

- `--ndb-optimized-node-selection`

Command-Line Format	--ndb-optimized-node-selection	
Permitted Values	Type	boolean
	Default	TRUE

Optimize selection of nodes for transactions. Enabled by default.

- `--version`, `-V`

Command-Line Format	--version

Prints the MySQL Cluster version number of the executable. The version number is relevant because not all versions can be used together, and the MySQL Cluster startup process verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of MySQL Cluster.

See Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”), for more information.

- `--ndb-shm`

Command-Line Format	--ndb-shm	
Permitted Values	Type	boolean
	Default	FALSE

In MySQL 5.0, this experimental option allows an application to use shared memory for optimization when this is available. It is not intended for production use. `--ndb-shm` is deprecated in MySQL 5.1, and is removed from all MySQL Cluster programs in MySQL Cluster NDB 7.0 and later.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

17.5 Management of MySQL Cluster

Managing a MySQL Cluster involves a number of tasks, the first of which is to configure and start MySQL Cluster. This is covered in Section 17.3, “MySQL Cluster Configuration”, and Section 17.4, “MySQL Cluster Programs”.

The next few sections cover the management of a running MySQL Cluster.

For information about security issues relating to management and deployment of a MySQL Cluster, see Section 17.5.10, “MySQL Cluster Security Issues”.

There are essentially two methods of actively managing a running MySQL Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log`; this is usually found in the management server's `DataDir` directory, but this location can be overridden using the `LogDestination` option—see Section 17.3.3.4, “Defining a MySQL Cluster Management Server”, for details. (Recall that *`node_id`* represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

In addition, some aspects of the cluster's operation can be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement. See Section 13.7.5.12, “SHOW ENGINE Syntax”, for more information.

17.5.1 Summary of MySQL Cluster Start Phases

This section provides a simplified outline of the steps involved when MySQL Cluster data nodes are started. More complete information can be found in MySQL Cluster Start Phases.

These phases are the same as those reported in the output from the *`node_id`* `STATUS` command in the management client. (See Section 17.5.2, “Commands in the MySQL Cluster Management Client”, for more information about this command.)

Start types. There are several different startup types and modes, as shown here:

- **Initial Start.** The cluster starts with a clean file system on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial` option.

  Note

  Disk Data files are not removed when restarting a node using `--initial`.

- **System Restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.

- **Node Restart.** This is the online restart of a cluster node while the cluster itself is running.

- **Initial Node Restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean file system.

Setup and initialization (Phase -1). Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

Normally, in the event of a problem with the node, the node disconnects from the management server, the socket used for the connection is closed, and the reserved node ID is freed. However, if a node is disconnected abruptly—for example, due to a hardware failure in one of the cluster hosts, or because of network issues—the normal closing of the socket by the operating system may not take place. In this case, the node ID continues to be reserved and not released until a TCP timeout occurs 10 or so minutes later.

To take care of this problem, you can use `PURGE STALE SESSIONS`. Running this statement forces all reserved node IDs to be checked; any that are not being used by nodes actually connected to the cluster are then freed.

Beginning with MySQL 5.1.11, timeout handling of node ID assignments is implemented. This performs the ID usage checks automatically after approximately 20 seconds, so that `PURGE STALE SESSIONS` should no longer be necessary in a normal Cluster start.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start (see NDB Kernel Blocks). The cluster file system is cleared, if the cluster was started with the `--initial` option.
- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. Cluster connections are set up, inter-block communications are established, and Cluster heartbeats are started. In the case of a node restart, API node connections are also checked.

  Note

  When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible cause of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

  For a system restart:

  - Read schema or schemas.
  - Read data from the local checkpoint.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Apply all redo information until the latest restorable global checkpoint has been reached.

For a node restart, find the tail of the redo log.

- **Phase 5.** Most of the database-related portion of a data node start is performed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.
- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as `Started`. It is now possible for API nodes (including SQL nodes) to connect to the cluster. connect.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by `DBDIH`).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (OBSOLETE).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as *`SUMA` handover phase*.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

17.5.2 Commands in the MySQL Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client `ndb_mgm`. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in Section 17.5.6, “Event Reports Generated in MySQL Cluster”; commands for creating backups and restoring from them are provided in Section 17.5.3, “Online Backup of MySQL Cluster”.

The management client has the following basic commands. In the listing that follows, *`node_id`* denotes either a database node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

-
-
-
-
-
-
-
-
-
-

This documentation is for an older version. If you're

17.5.3 Online Backup of MySQL Cluster

This documentation is for an older version. If you're

(For information about the creation of backups with `mysqldump`, see Section 4.5.4, “`mysqldump` — A Database Backup Program”.) Restoration of MySQL Cluster backups is done using the `ndb_restore` utility provided with the MySQL Cluster distribution; for information about `ndb_restore` and its use in restoring MySQL Cluster backups, see Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”.

17.5.3.1 MySQL Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.ctl`

  A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-`*`backup_id`*`-0.`*`node_id`*`.data`

  A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.log`

  A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing above, *`backup_id`* stands for the backup identifier and *`node_id`* is the unique identifier for the node creating the file.

17.5.3.2 Using The MySQL Cluster Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See Section 17.5.3.3, “Configuration for MySQL Cluster Backups”.)

The `START BACKUP` command is used to create a backup:

```
START BACKUP [backup_id] [wait_option]

wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT
```

Successive backups are automatically identified sequentially, so the *`backup_id`*, an integer greater than or equal to 1, is optional; if it is omitted, the next available value is used. If an existing *`backup_id`* value is used, the backup fails with the error `Backup failed: file already exists`. If used, the *`backup_id`* must follow `START BACKUP` immediately, before any other options are used.

The maximum supported value for *`backup_id`* in MySQL 5.0 is 2147483648 (2^{31}). (Bug #43042)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

If you start a backup using `ndb_mgm -e "START BACKUP"`, the *`backup_id`* is required.

The *`wait_option`* can be used to determine when control is returned to the management client after a `START BACKUP` command is issued, as shown in the following list:

-
-
-

`WAIT COMPLETED` is the default.

The procedure for creating a backup consists of the following steps:

1. Start the management client (`ndb_mgm`), if it not running already.
2. Execute the **START BACKUP** command. This produces several lines of output indicating the progress of the backup, as shown here:

   ```
   ndb_mgm> START BACKUP
   Waiting for completed, this may take several minutes
   Node 2: Backup 1 started from node 1
   Node 2: Backup 1 started from node 1 completed
    StartGCP: 177 StopGCP: 180
    #Records: 7362 #LogRecords: 0
    Data: 453648 bytes Log: 0 bytes
   ndb_mgm>
   ```

3.
4. The management client indicates with a message like this one that the backup has started:

   ```
   Backup backup_id started from node node_id completed
   ```

   As is the case for the notification that the backup has started, *`backup_id`* is the unique identifier for this particular backup, and *`node_id`* is the node ID of the management server that is coordinating the backup with the data nodes. This output is accompanied by additional information including relevant global checkpoints, the number of records backed up, and the size of the data, as shown here:

   ```
   Node 2: Backup 1 started from node 1 completed
    StartGCP: 177 StopGCP: 180
    #Records: 7362 #LogRecords: 0
    Data: 453648 bytes Log: 0 bytes
   ```

It is also possible to perform a backup from the system shell by invoking `ndb_mgm` with the `-e` or `--execute` option, as shown in this example:

```
shell> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED"
```

When using `START BACKUP` in this way, you must specify the backup ID.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` configuration parameter as discussed in Identifying Data Nodes. The backup files created for a backup with a given *`backup_id`* are stored in a subdirectory named `BACKUP-`*`backup_id`* in the backup directory.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To abort a backup that is already in progress:

1. Start the management client.
2. Execute this command:

```
ndb_mgm> ABORT BACKUP backup_id
```

   The number *`backup_id`* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup backup_id started from node management_node_id`).

3. The management client will acknowledge the abort request with `Abort of backup backup_id ordered`.

Note

At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.

4. After the backup has been aborted, the management client will report this fact in a manner similar to what is shown here:

```
Node 1: Backup 3 started from 5 has been aborted.
  Error: 1321 - Backup aborted by user request: Permanent error: User defined error
Node 3: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
```

   In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is 3, and the management node to which the cluster management client is connected has the node ID 5. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)

Note

There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

   The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster file system.

It is also possible to abort a backup in progress from a system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

Note

If there is no backup having the ID *`backup_id`* running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

17.5.3.3 Configuration for MySQL Cluster Backups

Five configuration parameters are essential for backup:

- `BackupDataBufferSize`

  The amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize`

  The amount of memory used to buffer log records before these are written to disk.

- `BackupMemory`

  The total memory allocated in a database node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize`

  The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

- `BackupMaxWriteSize`

  The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in Backup Parameters.

17.5.3.4 MySQL Cluster Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.

Important

If you have set `BackupDataBufferSize` and `BackupLogBufferSize` and their sum is greater than 4MB, then you must also set `BackupMemory` as well. See `BackupMemory`.

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

`NDB` does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring a MySQL Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get nonrepeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

17.5.4 MySQL Server Usage for MySQL Cluster

`mysqld` is the traditional MySQL server process. To be used with MySQL Cluster, `mysqld` needs to be built with support for the `NDBCLUSTER` storage engine, as it is in the precompiled binaries available from http://dev.mysql.com/downloads/. If you build MySQL from source, you must invoke `configure` with the `--with-ndbcluster` option to enable `NDB Cluster` storage engine support.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For information about other MySQL server options and variables relevant to MySQL Cluster in addition to those discussed in this section, see Section 17.3.3.7, “MySQL Server Options and Variables for MySQL Cluster”.

If the `mysqld` binary has been built with Cluster support, the `NDBCLUSTER` storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` as a startup option on the command line when starting `mysqld`.
- Insert a line containing `NDBCLUSTER` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the `NDBCLUSTER` storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (`mysql`). You should see the value `YES` as the `Support` value in the row for `NDBCLUSTER`. If you see `NO` in this row or if there is no such row displayed in the output, you are not running an `NDB`-enabled version of MySQL. If you see `DISABLED` in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The host name or IP address for the management server (MGM node)
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The `mysqld` parameter `ndb-connectstring` is used to specify the connection string either on the command line when starting `mysqld` or in `my.cnf`. The connection string contains the host name or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 17.3.3.2, “MySQL Cluster Connection Strings”, for more information on connection strings.

Given this information, the MySQL server will be a full participant in the cluster. (We often refer to a `mysqld` process running in this manner as an SQL node.) It will be fully aware of all cluster data nodes as well as their status, and will establish connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` privilege, then the first row of the output is as shown here:

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db:
Command: Daemon
   Time: 1
  State: Waiting for event from ndbcluster
   Info: NULL
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Important

To participate in a MySQL Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster` and `--ndb-connectstring` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster` option, or if it is unable to contact the cluster, it is not possible to work with `NDB` tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as `NDB` tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in a MySQL Cluster, you must restart the server *without* the `--ndbcluster` option.

17.5.5 Performing a Rolling Restart of a MySQL Cluster

This section discusses how to perform a *rolling restart* of a MySQL Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable. These are described in the next few paragraphs.

Configuration change.
To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.

MySQL Cluster software upgrade or downgrade. To upgrade the cluster to a newer version of the MySQL Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of MySQL Cluster).

Change on node host. To make changes in the hardware or operating system on which one or more MySQL Cluster node processes are running.

System reset (cluster reset).
To reset the cluster because it has reached an undesirable state. In such cases it is often desirable to reload the data and metadata of one or more data nodes. This can be done any of three ways:

- Start each data node process (`ndbd`) with the `--initial` option, which forces the data node to clear its file system and to reload all MySQL Cluster data and metadata from the other data nodes.
- Create a backup using the `ndb_mgm` client `BACKUP` command prior to performing the restart. Following the upgrade, restore the node or nodes using `ndb_restore`.

  See Section 17.5.3, “Online Backup of MySQL Cluster”, and Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”, for more information.
- Use `mysqldump` to create a backup prior to the upgrade; afterward, restore the dump using `LOAD DATA INFILE`.

Resource Recovery.
To free memory previously allocated to a table by successive `INSERT` and `DELETE` operations, for re-use by other MySQL Cluster tables.

The process for performing a rolling restart may be generalized as follows:

1. Stop all cluster management nodes (`ndb_mgmd` processes), reconfigure them, then restart them.

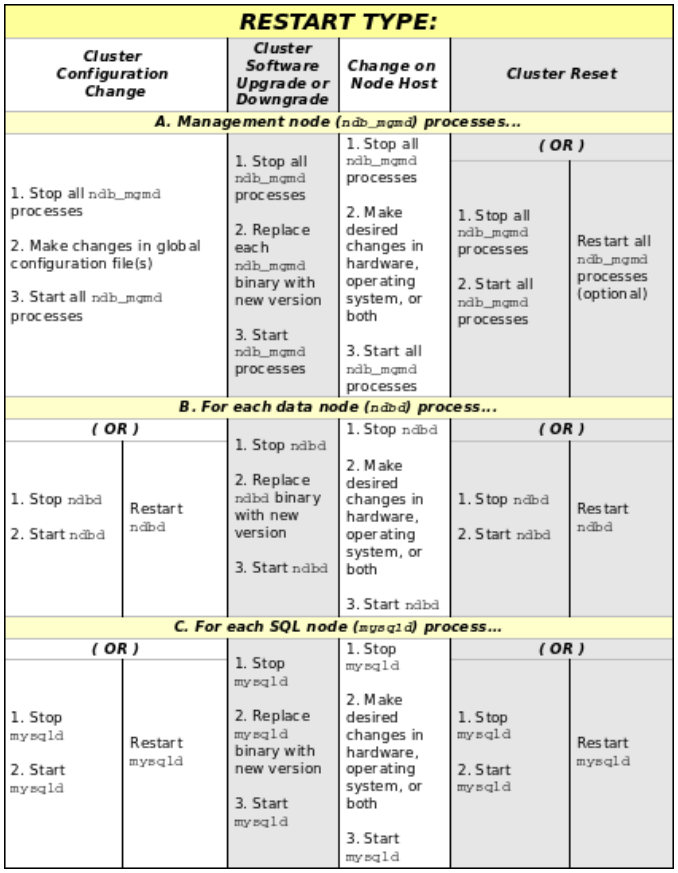
This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. Stop, reconfigure, then restart each cluster data node (`ndbd` process) in turn.

3. Stop, reconfigure, then restart each cluster SQL node (`mysqld` process) in turn.

The specifics for implementing a given rolling upgrade depend upon the changes being made. A more detailed view of the process is presented here:

Figure 17.6 MySQL Cluster Rolling Restarts By Type



In the previous diagram, the **Stop** and **Start** steps indicate that the process must be stopped completely using a shell command (such as `kill` on most Unix systems) or the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndb_mgmd` executable as appropriate.

Restart indicates that the process may be restarted using the `ndb_mgm` management client `RESTART` command (see Section 17.5.2, “Commands in the MySQL Cluster Management Client”).

Important

When performing an upgrade or downgrade of the MySQL Cluster software, you must upgrade or downgrade the management nodes first, then the data nodes, and finally the SQL nodes. Doing so in any other order may leave the cluster in an unusable state.

17.5.6 Event Reports Generated in MySQL Cluster

In this section, we discuss the types of event logs provided by MySQL Cluster, and the types of events that are logged.

MySQL Cluster provides two types of event log:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

  By default, the cluster log is saved to a file named `ndb_node_id_cluster.log`, (where *`node_id`* is the node ID of the management server) in the management server's `DataDir`.

  Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` and `LogDestination` configuration parameters. See Section 17.3.3.4, “Defining a MySQL Cluster Management Server”, for more information about these parameters.

- *Node logs* are local to each node.

  Output generated by node event logging is written to the file `ndb_node_id_out.log` (where *`node_id`* is the node's node ID) in the node's `DataDir`. Node event logs are generated for both management nodes and data nodes.

  Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.

- *Priority*: This is represented by one of the numbers from 0 to 15 inclusive, where 0 indicates “most important” and 15 “least important.”

- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

```
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9: API version 5.1.15
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Each line in the cluster log contains the following information:

- A timestamp in *YYYY-MM-DD HH:MM:SS* format.
- The type of node which is performing the logging. In the cluster log, this is always [MgmSrvr].
- The severity of the event.
- The ID of the node reporting the event.
- A description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status information.

17.5.6.1 MySQL Cluster Logging Management Commands

ndb_mgm supports a number of management commands related to the cluster log. In the listing that follows, *node_id* denotes either a database node ID or the keyword ALL, which indicates that the command should be applied to all of the cluster's data nodes.

- CLUSTERLOG ON

  Turns the cluster log on.

- CLUSTERLOG OFF

  Turns the cluster log off.

- CLUSTERLOG INFO

  Provides information about cluster log settings.

- *node_id* CLUSTERLOG *category=threshold*

  Logs *category* events with priority less than or equal to *threshold* in the cluster log.

- CLUSTERLOG FILTER *severity_level*

  Toggles cluster logging of events of the specified *severity_level*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note that events are reported per data node, and that the threshold can be set to different values on different nodes.

Category	Default threshold (All data nodes)
STARTUP	**7**
SHUTDOWN	**7**
STATISTICS	**7**
CHECKPOINT	**7**
NODERESTART	**7**
CONNECTION	**7**
ERROR	**15**
INFO	**7**

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The STATISTICS category can provide a great deal of useful data. See Section 17.5.6.3, “Using CLUSTERLOG STATISTICS in the MySQL Cluster Management Client”, for more information.

Thresholds are used to filter events within each category. For example, a STARTUP event with a priority of 3 is not logged unless the threshold for STARTUP is set to 3 or higher. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

Note

These correspond to Unix syslog levels, except for LOG_EMERG and LOG_NOTICE, which are not used or mapped.

Severity Level Value	Severity	Description
1	ALERT	A condition that should be corrected immediately, such as a corrupted system database
2	CRITICAL	Critical conditions, such as device errors or insufficient resources
3	ERROR	Conditions that should be corrected, such as configuration errors
4	WARNING	Conditions that are not errors, but that might require special handling
5	INFO	Informational messages
6	DEBUG	Debugging messages used for NDBCLUSTER development

Event severity levels can be turned on or off (using CLUSTERLOG FILTER—see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

Important

Cluster log levels are set on a per ndb_mgmd, per subscriber basis. This means that, in a MySQL Cluster with multiple management servers, using a CLUSTERLOG command in an instance of ndb_mgm connected to one management server affects only logs generated by that management server but not by any of the others. This also means that, should one of the management servers be restarted, only logs generated by that management server are affected by the resetting of log levels caused by the restart.

17.5.6.2 MySQL Cluster Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint”, respectively.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`CONNECTION` Events

These events are associated with connections between Cluster nodes.

Event	Priority	Severity Level	Description
data nodes connected	8	`INFO`	Data nodes connected
data nodes disconnected	8	`INFO`	Data nodes disconnected
Communication closed	8	`INFO`	SQL node or data node connection closed
Communication opened	8	`INFO`	SQL node or data node connection opened

`CHECKPOINT` Events

The logging messages shown here are associated with checkpoints.

Event	Priority	Severity Level	Description
LCP stopped in calc keep GCI	0	`ALERT`	LCP stopped
Local checkpoint fragment completed	11	`INFO`	LCP on a fragment has been completed
Global checkpoint completed	10	`INFO`	GCP finished
Global checkpoint started	9	`INFO`	Start of GCP: REDO log is written to disk
Local checkpoint completed	8	`INFO`	LCP completed normally
Local checkpoint started	7	`INFO`	Start of LCP: data written to disk
Report undo log blocked	7	`INFO`	UNDO logging blocked; buffer near overflow

`STARTUP` Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Event	Priority	Severity Level	Description
Internal start signal received STTORRY	15	`INFO`	Blocks received after completion of restart
Undo records executed	15	`INFO`	
New REDO log started	10	`INFO`	GCI keep *X*, newest restorable GCI *Y*
New log started	10	`INFO`	Log part *X*, start MB *Y*, stop MB *Z*
Node has been refused for inclusion in the cluster	8	`INFO`	Node cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
data node neighbors	8	`INFO`	Shows neighboring data nodes
data node start phase *X* completed	4	`INFO`	A data node start phase has been completed
Node has been successfully included into the cluster	3	`INFO`	Displays the node, managing node, and dynamic ID
data node start phases initiated	1	`INFO`	NDB Cluster nodes starting

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Event	Priority	Severity Level	Description
data node all start phases completed	1	INFO	NDB Cluster nodes started
data node shutdown initiated	1	INFO	Shutdown of data node has commenced
data node shutdown aborted	1	INFO	Unable to shut down data node normally

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Event	Priority	Severity Level	Description
Node failure phase completed	8	ALERT	Reports completion of node failure phases
Node has failed, node state was *X*	8	ALERT	Reports that a node has failed
Report arbitrator results	2	ALERT	There are eight different possible results for arbitration attempts: • Arbitration check failed—less than 1/2 nodes left • Arbitration check succeeded—node group majority • Arbitration check failed—missing node group • Network partitioning—arbitration required • Arbitration succeeded—affirmative response from node *X* • Arbitration failed - negative response from node *X* • Network partitioning - no arbitrator available • Network partitioning - no arbitrator configured
Completed copying a fragment	10	INFO	
Completed copying of dictionary information	8	INFO	
Completed copying distribution information	8	INFO	
Starting to copy fragments	8	INFO	
Completed copying all fragments	8	INFO	
GCP takeover started	7	INFO	
GCP takeover completed	7	INFO	
LCP takeover started	7	INFO	
LCP takeover completed (state = *X*)	7	INFO	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Event	Priority	Severity Level	Description
Report whether an arbitrator is found or not	6	INFO	There are seven different possible outcomes when seeking an arbitrator: • Management server restarts arbitration thread [state=*X*] • Prepare arbitrator node *X* [ticket=*Y*] • Receive arbitrator node *X* [ticket=*Y*] • Started arbitrator node *X* [ticket=*Y*] • Lost arbitrator node *X* - process failure [state=*Y*] • Lost arbitrator node *X* - process exit [state=*Y*] • Lost arbitrator node *X* <error msg> [state=*Y*]

STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Event	Priority	Severity Level	Description
Report job scheduling statistics	9	INFO	Mean internal job scheduling statistics
Sent number of bytes	9	INFO	Mean number of bytes sent to node *X*
Received # of bytes	9	INFO	Mean number of bytes received from node *X*
Report transaction statistics	8	INFO	Numbers of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
Report operations	8	INFO	Number of operations
Report table create	7	INFO	
Memory usage	5	INFO	Data and index memory usage (80%, 90%, and 100%)

ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Event	Priority	Severity	Description
Dead due to missed heartbeat	8	ALERT	Node *X* declared “dead” due to missed heartbeat
Transporter errors	2	ERROR	
Transporter warnings	8	WARNING	
Missed heartbeats	8	WARNING	Node *X* missed heartbeat #*Y*

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Event	Priority	Severity	Description
General warning events	2	WARNING	

INFO Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Event	Priority	Severity	Description
Sent heartbeat	12	INFO	Heartbeat sent to node *X*
Create log bytes	11	INFO	Log part, log file, MB
General information events	2	INFO	

Note

Sent heartbeat events are available only if MySQL Cluster was compiled with VM_TRACE enabled.

17.5.6.3 Using CLUSTERLOG STATISTICS in the MySQL Cluster Management Client

The NDB management client's CLUSTERLOG STATISTICS command can provide a number of useful statistics in its output. Counters providing information about the state of the cluster are updated at 5-second reporting intervals by the transaction coordinator (TC) and the local query handler (LQH), and written to the cluster log.

Transaction coordinator statistics. Each transaction has one transaction coordinator, which is chosen by one of the following methods:

- In a round-robin fashion
- By communication proximity

Note

You can determine which TC selection method is used for transactions started from a given SQL node using the ndb_optimized_node_selection system variable. For more information, see MySQL Cluster System Variables.

All operations within the same transaction use the same transaction coordinator, which reports the following statistics:

- **Trans count.** This is the number transactions started in the last interval using this TC as the transaction coordinator. Any of these transactions may have committed, have been aborted, or remain uncommitted at the end of the reporting interval.

  Note

  Transactions do not migrate between TCs.

- **Commit count.** This is the number of transactions using this TC as the transaction coordinator that were committed in the last reporting interval. Because some transactions committed in this reporting interval may have started in a previous reporting interval, it is possible for Commit count to be greater than Trans count.
- **Read count.** This is the number of primary key read operations using this TC as the transaction coordinator that were started in the last reporting interval, including simple reads. This count also

This documentation is for an older version. If you're
This documentation is for an older version. If you're

includes reads performed as part of unique index operations. A unique index read operation generates 2 primary key read operations—1 for the hidden unique index table, and 1 for the table on which the read takes place.

- **Simple read count.** This is the number of simple read operations using this TC as the transaction coordinator that were started in the last reporting interval. This is a subset of `Read count`. Because the value of `Simple read count` is incremented at a different point in time from `Read count`, it can lag behind `Read count` slightly, so it is conceivable that `Simple read count` is not equal to `Read count` for a given reporting interval, even if all reads made during that time were in fact simple reads.

- **Write count.** This is the number of primary key write operations using this TC as the transaction coordinator that were started in the last reporting interval. This includes all inserts, updates, writes and deletes, as well as writes performed as part of unique index operations.

  Note

  A unique index update operation can generate multiple PK read and write operations on the index table and on the base table.

- **AttrInfoCount.** This is the number of 32-bit data words received in the last reporting interval for primary key operations using this TC as the transaction coordinator. For reads, this is proportional to the number of columns requested. For inserts and updates, this is proportional to the number of columns written, and the size of their data. For delete operations, this is usually zero. Unique index operations generate multiple PK operations and so increase this count. However, data words sent to describe the PK operation itself, and the key information sent, are *not* counted here. Attribute information sent to describe columns to read for scans, or to describe ScanFilters, is also not counted in `AttrInfoCount`.

- **Concurrent Operations.** This is the number of primary key or scan operations using this TC as the transaction coordinator that were started during the last reporting interval but that were not completed. Operations increment this counter when they are started and decrement it when they are completed; this occurs after the transaction commits. Dirty reads and writes—as well as failed operations—decrement this counter. The maximum value that `Concurrent Operations` can have is the maximum number of operations that a TC block can support; currently, this is `(2 * MaxNoOfConcurrentOperations) + 16 + MaxNoOfConcurrentTransactions`. (For more information about these configuration parameters, see the *Transaction Parameters* section of Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”.)

- **Abort count.** This is the number of transactions using this TC as the transaction coordinator that were aborted during the last reporting interval. Because some transactions that were aborted in the last reporting interval may have started in a previous reporting interval, `Abort count` can sometimes be greater than `Trans count`.

- **Scans.** This is the number of table scans using this TC as the transaction coordinator that were started during the last reporting interval. This does not include range scans (that is, ordered index scans).

- **Range scans.** This is the number of ordered index scans using this TC as the transaction coordinator that were started in the last reporting interval.

Local query handler statistics (Operations). There is 1 cluster event per local query handler block (that is, 1 per data node process). Operations are recorded in the LQH where the data they are operating on resides.

Note

A single transaction may operate on data stored in multiple LQH blocks.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The `Operations` statistic provides the number of local operations performed by this LQH block in the last reporting interval, and includes all types of read and write operations (insert, update, write, and delete operations). This also includes operations used to replicate writes—for example, in a 2-replica cluster, the write to the primary replica is recorded in the primary LQH, and the write to the backup will be recorded in the backup LQH. Unique key operations may result in multiple local operations; however, this does *not* include local operations generated as a result of a table scan or ordered index scan, which are not counted.

Process scheduler statistics. In addition to the statistics reported by the transaction coordinator and local query handler, each `ndbd` process has a scheduler which also provides useful metrics relating to the performance of a MySQL Cluster. This scheduler runs in an infinite loop; during each loop the scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

Process scheduler statistics include the following:

- **Mean Loop Counter.** This is the number of loops executed in the third step from the preceding list. This statistic increases in size as the utilization of the TCP/IP buffer improves. You can use this to monitor changes in performance as you add new data node processes.
- **Mean send size and Mean receive size.** These statistics enable you to gauge the efficiency of, respectively writes and reads between nodes. The values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum value is 64K.

To cause all cluster log statistics to be logged, you can use the following command in the `NDB` management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

Note

Setting the threshold for `STATISTICS` to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity in the MySQL Cluster.

For more information about MySQL Cluster management client commands relating to logging and reporting, see Section 17.5.6.1, “MySQL Cluster Logging Management Commands”.

17.5.7 MySQL Cluster Log Messages

This section contains information about the messages written to the cluster log in response to different cluster log events. It provides additional, more specific information on `NDB` transporter errors.

17.5.7.1 MySQL Cluster: Messages in the Cluster Log

The following table lists the most common `NDB` cluster log messages. For information about the cluster log, log events, and event types, see Section 17.5.6, “Event Reports Generated in MySQL Cluster”. These

This documentation is for an older version. If you're
This documentation is for an older version. If you're

log messages also correspond to log event types in the MGM API; see The Ndb_logevent_type Type, for related information of interest to Cluster API developers.

Log Message	Description	Event Name	Event Type	Priority	Severity
Node *mgm_node_id*: Node *data_node_id* Connected	The data node having node ID *node_id* has connected to the management server (node *mgm_node_id*).	Connected	Connection	8	INFO
Node *mgm_node_id*: Node *data_node_id* Disconnected	The data node having node ID *data_node_id* has disconnected from the management server (node *mgm_node_id*).	Disconnected	Connection	8	ALERT
Node *data_node_id*: Communication to Node *api_node_id* closed	The API node or SQL node having node ID *api_node_id* is no longer communicating with data node *data_node_id*.	CommunicationClosed	Connection	8	INFO
Node *data_node_id*: Communication to Node *api_node_id* opened	The API node or SQL node having node ID *api_node_id* is now communicating with data node *data_node_id*.	CommunicationOpened	Connection	8	INFO
Node *mgm_node_id*: Node *api_node_id*: API *version*	The API node having node ID *api_node_id* has connected to management node *mgm_node_id* using NDB API version *version* (generally the same as the MySQL version number).	ConnectedApiVersion	Connection	8	INFO
Node *node_id*: Global checkpoint *gci* started	A global checkpoint with the ID *gci* has been started; node *node_id* is the master responsible	GlobalCheckpointStarted	Checkpoint	9	INFO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
	for this global checkpoint.				
Node *node_id*: Global checkpoint *gci* completed	The global checkpoint having the ID *gci* has been completed; node *node_id* was the master responsible for this global checkpoint.	GlobalCheckpointCompleted	Checkpoint	10	INFO
Node *node_id*: Local checkpoint *lcp* started. Keep GCI = *current_gci* oldest restorable GCI = *old_gci*	The local checkpoint having sequence ID *lcp* has been started on node *node_id*. The most recent GCI that can be used has the index *current_gci*, and the oldest GCI from which the cluster can be restored has the index *old_gci*.	LocalCheckpointStarted	Checkpoint	7	INFO
Node *node_id*: Local checkpoint *lcp* completed	The local checkpoint having sequence ID *lcp* on node *node_id* has been completed.	LocalCheckpointCompleted	Checkpoint	8	INFO
Node *node_id*: Local Checkpoint stopped in CALCULATED_KEEP_GCI	The node was unable to determine the most recent usable GCI.	LCPStoppedInCalcKeepGci	Checkpoint	0	ALERT
Node *node_id*: Table ID = *table_id*, fragment *ID* = *fragment_id* has completed LCP on Node *node_id* maxGciStarted: *started_gci* maxGciCompleted: *completed_gci*	A table fragment has been checkpointed to disk on node *node_id*. The GCI in progress has the index *started_gci*, and the most recent GCI to have been completed has the index *completed_gci*.	LCPFragmentCompleted	Checkpoint	11	INFO
Node *node_id*: ACC Blocked *num_1* and TUP	Undo logging is blocked because the log	UndoLogBlocked	Checkpoint	7	INFO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
`Blocked num_2 times last second`	buffer is close to overflowing.				
`Node node_id: Start initiated version`	Data node *node_id*, running NDB version *version*, is beginning its startup process.	`NDBStartStarted`	`StartUp`	1	`INFO`
`Node node_id: Started version`	Data node *node_id*, running NDB version *version*, has started successfully.	`NDBStartCompleted`	`StartUp`	1	`INFO`
`Node node_id: STTORRY received after restart finished`	The node has received a signal indicating that a cluster restart has completed.	`STTORRYRecieved`	`StartUp`	15	`INFO`
`Node node_id: Start phase phase completed (type)`	The node has completed start phase *phase* of a *type* start. For a listing of start phases, see Section 17.5.1, "Summary of MySQL Cluster Start Phases". (*type* is one of `initial`, `system`, `node`, `initial node`, or `<Unknown>`.)	`StartPhaseCompleted`	`StartUp`	4	`INFO`
`Node node_id: CM_REGCONF president = president_id, own Node = own_id, our dynamic id = dynamic_id`	Node *president_id* has been selected as "president". *own_id* and *dynamic_id* should always be the same as the ID (*node_id*) of the reporting node.	`CM_REGCONF`	`StartUp`	3	`INFO`
`Node node_id: CM_REGREF from Node president_id to our Node`	The reporting node (ID *node_id*) was unable to accept node *president_id* as president. The	`CM_REGREF`	`StartUp`	8	`INFO`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
node_id. Cause = *cause*	*cause* of the problem is given as one of Busy, Election with wait = false, Not president, Election without selecting new candidate, or No such cause.				
Node *node_id*: We are Node *own_id* with dynamic ID *dynamic_id*, our left neighbor is Node *id_1*, our right is Node *id_2*	The node has discovered its neighboring nodes in the cluster (node *id_1* and node *id_2*). *node_id*, *own_id*, and *dynamic_id* should always be the same; if they are not, this indicates a serious misconfiguration of the cluster nodes.	FIND_NEIGHBOURS	StartUp	8	INFO
Node *node_id*: *type* shutdown initiated	The node has received a shutdown signal. The *type* of shutdown is either Cluster or Node.	NDBStopStarted	StartUp	1	INFO
Node *node_id*: Node shutdown completed [, *action*] [Initiated by signal *signal*.]	The node has been shut down. This report may include an *action*, which if present is one of restarting, no start, or initial. The report may also include a reference to an NDB Protocol *signal*; for possible signals, refer to Operations and Signals.	NDBStopCompleted	StartUp	1	INFO
Node *node_id*: Forced node shutdown completed [, action].	The node has been forcibly shut down. The *action* (one of restarting, no start,	NDBStopForced	StartUp	1	ALERT

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
[Occured during startphase *start_phase*.] [Initiated by *signal*.] [Caused by error *error_code*: '*error_message*(*error_classification*).*error_status*'. [(extra info *extra_code*)]]	or initial) subsequently being taken, if any, is also reported. If the shutdown occurred while the node was starting, the report includes the *start_phase* during which the node failed. If this was a result of a *signal* sent to the node, this information is also provided (see Operations and Signals, for more information). If the error causing the failure is known, this is also included; for more information about NDB error messages and classifications, see MySQL Cluster API Errors.				
Node *node_id*: Node shutdown aborted	The node shutdown process was aborted by the user.	NDBStopAborted	StartUp	1	INFO
Node *node_id*: StartLog: [GCI Keep: *keep_pos* LastCompleted: *last_pos* NewestRestorable *restore_pos*]	This reports global checkpoints referenced during a node start. The redo log prior to *keep_pos* is dropped. *last_pos* is the last global checkpoint in which data node the participated; *restore_pos* is the global checkpoint which is actually used to restore all data nodes.	StartREDOLog	StartUp	4	INFO
startup_message [*Listed separately; see below.*]	There are a number of possible startup messages that	StartReport	StartUp	4	INFO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
	can be logged under different circumstances. These are listed separately; see Section 17.5.7.2, "MySQL Cluster Log Startup Messages".				
`Node node_id: Node restart completed copy of dictionary information`	Copying of data dictionary information to the restarted node has been completed.	`NR_CopyDict`	`NodeRestart`	8	`INFO`
`Node node_id: Node restart completed copy of distribution information`	Copying of data distribution information to the restarted node has been completed.	`NR_CopyDistr`	`NodeRestart`	8	`INFO`
`Node node_id: Node restart starting to copy the fragments to Node node_id`	Copy of fragments to starting data node *node_id* has begun	`NR_CopyFragsStarted`	`NodeRestart`	8	`INFO`
`Node node_id: Table ID = table_id, fragment ID = fragment_id have been copied to Node node_id`	Fragment *fragment_id* from table *table_id* has been copied to data node *node_id*	`NR_CopyFragDone`	`NodeRestart`	10	`INFO`
`Node node_id: Node restart completed copying the fragments to Node node_id`	Copying of all table fragments to restarting data node *node_id* has been completed	`NR_CopyFragsCompleted`	`NodeRestart`	8	`INFO`
`Node node_id: Node node1_id completed failure of Node node2_id`	Data node *node1_id* has detected the failure of data node *node2_id*	`NodeFailCompleted`	`NodeRestart`	8	`ALERT`
`All nodes completed failure of Node node_id`	All (remaining) data nodes have detected the failure of data node *node_id*	`NodeFailCompleted`	`NodeRestart`	8	`ALERT`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
`Node failure of node_idblock completed`	The failure of data node *node_id* has been detected in the *block*NDB kernel block, where block is 1 of DBTC, DBDICT, DBDIH, or DBLQH; for more information, see NDB Kernel Blocks	NodeFailCompleted	NodeRestart	8	ALERT
`Node mgm_node_id: Node data_node_id has failed. The Node state at failure was state_code`	A data node has failed. Its state at the time of failure is described by an arbitration state code *state_code*: possible state code values can be found in the file include/kernel/signaldata/ArbitSignalData.hpp.	NODE_FAILREP	NodeRestart	8	ALERT
`President restarts arbitration thread [state=state_code]` or `Prepare arbitrator node node_id [ticket=ticket_id]` or `Receive arbitrator node node_id [ticket=ticket_id]` or `Started arbitrator node node_id [ticket=ticket_id]` or `Lost arbitrator node node_id - process failure [state=state_code]` or `Lost arbitrator node node_id - process exit [state=state_code]` or `Lost arbitrator`	This is a report on the current state and progress of arbitration in the cluster. *node_id* is the node ID of the management node or SQL node selected as the arbitrator. *state_code* is an arbitration state code, as found in include/kernel/signaldata/ArbitSignalData.hpp. When an error has occurred, an *error_message*, also defined in ArbitSignalData.hpp, is provided. *ticket_id* is a unique identifier handed out by the arbitrator when it is selected to all the nodes that participated	ArbitState	NodeRestart	6	INFO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
node *node_id* - *error_message* [state=*state_code*]	in its selection; this is used to ensure that each node requesting arbitration was one of the nodes that took part in the selection process.				
Arbitration check lost - less than 1/2 nodes left or Arbitration check won - all node groups and more than 1/2 nodes left or Arbitration check won - node group majority or Arbitration check lost - missing node group or Network partitioning - arbitration required or Arbitration won - positive reply from node *node_id* or Arbitration lost - negative reply from node *node_id* or Network partitioning - no arbitrator available or Network partitioning - no arbitrator configured or Arbitration failure - *error_message* [state=*state_code*]	This message reports on the result of arbitration. In the event of arbitration failure, an *error_message* and an arbitration *state_code* are provided; definitions for both of these are found in include/kernel/signaldata/ArbitSignalData.hpp.	ArbitResult	NodeRestart	2	ALERT
Node *node_id*: GCP Take over started	This node is attempting to assume	GCP_TakeoverStarted	NodeRestart	7	INFO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
	responsibility for the next global checkpoint (that is, it is becoming the master node)				
Node *node_id*: GCP Take over completed	This node has become the master, and has assumed responsibility for the next global checkpoint	GCP_TakeoverCompleted	NodeRestart	7	INFO
Node *node_id*: LCP Take over started	This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node)	LCP_TakeoverStarted	NodeRestart	7	INFO
Node *node_id*: LCP Take over completed	This node has become the master, and has assumed responsibility for the next set of local checkpoints	LCP_TakeoverCompleted	NodeRestart	7	INFO
Node *node_id*: Trans. Count = *transactions*, Commit Count = *commits*, Read Count = *reads*, Simple Read Count = *simple_reads*, Write Count = *writes*, AttrInfo Count = *AttrInfo_objects*, Concurrent Operations = *concurrent_operations*, Abort Count = *aborts*, Scans = *scans*, Range scans = *range_scans*	This report of transaction activity is given approximately once every 10 seconds	TransReportCounters	Statistic	8	INFO
Node *node_id*: Operations=*operations*	Number of operations performed by this	OperationReportCounters	Statistic	8	INFO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
	node, provided approximately once every 10 seconds				
Node *node_id*: Table with ID = *table_id* created	A table having the table ID shown has been created	TableCreated	Statistic	7	INFO
Node *node_id*: Mean loop Counter in doJob last 8192 times = *count*		JobStatistic	Statistic	9	INFO
Mean send size to Node = *node_id* last 4096 sends = *bytes* bytes	This node is sending an average of *bytes* bytes per send to node *node_id*	SendBytesStatistic	Statistic	9	INFO
Mean receive size to Node = *node_id* last 4096 sends = *bytes* bytes	This node is receiving an average of *bytes* of data each time it receives data from node *node_id*	ReceiveBytesStatistic	Statistic	9	INFO
Node *node_id*: Data usage is *data_memory_percentage*% (*data_pages_used* 32K pages of total *data_pages_total*) / Node *node_id*: Index usage is *index_memory_percentage*% (*index_pages_used* 8K pages of total *index_pages_total*)	This report is generated when a DUMP 1000 command is issued in the cluster management client; for more information, see DUMP 1000, in MySQL Cluster Internals	MemoryUsage	Statistic	5	INFO
Node *node1_id*: Transporter to node *node2_id* reported error *error_code*: *error_message*	A transporter error occurred while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL Cluster Internals	TransporterError	Error	2	ERROR

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
`Node node1_id: Transporter to node node2_id reported error error_code: error_message`	A warning of a potential transporter problem while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, for more information	`TransporterWarning`	`Error`	8	`WARNING`
`Node node1_id: Node node2_id missed heartbeat heartbeat_id`	This node missed a heartbeat from node *node2_id*	`MissedHeartbeat`	`Error`	8	`WARNING`
`Node node1_id: Node node2_id declared dead due to missed heartbeat`	This node has missed at least 3 heartbeats from node *node2_id*, and so has declared that node “dead”	`DeadDueToHeartbeat`	`Error`	8	`ALERT`
`Node node1_id: Node Sent Heartbeat to node = node2_id`	This node has sent a heartbeat to node *node2_id*	`SentHeartbeat`	`Info`	12	`INFO`
`Node node_id: Event buffer status: used=bytes_used (percent_used%) alloc=bytes_allocated (percent_available%) max=bytes_available apply_gci=latest_restorable_GCI latest_gci=latest_GCI`	This report is seen during heavy event buffer usage, for example, when many updates are being applied in a relatively short period of time; the report shows the number of bytes and the percentage of event buffer memory used, the bytes allocated and percentage still available, and the latest and latest restorable global checkpoints	`EventBufferStatus`	`Info`	7	`INFO`
`Node node_id: Entering single user mode, Node node_id:`	These reports are written to the cluster log when entering and exiting	`SingleUser`	`Info`	7	`INFO`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
`Entered single user mode Node API_node_id has exclusive access`, `Node node_id: Entering single user mode`	single user mode; *API_node_id* is the node ID of the API or SQL having exclusive access to the cluster (for more information, see Section 17.5.8, "MySQL Cluster Single User Mode"); the message `Unknown single user report API_node_id` indicates an error has taken place and should never be seen in normal operation				
`Node node_id: Backup backup_id started from node mgm_node_id`	A backup has been started using the management node having *mgm_node_id*; this message is also displayed in the cluster management client when the START BACKUP command is issued; for more information, see Section 17.5.3.2, "Using The MySQL Cluster Management Client to Create a Backup"	`BackupStarted`	`Backup`	7	`INFO`
`Node node_id: Backup backup_id started from node mgm_node_id completed. StartGCP: start_gcp StopGCP: stop_gcp #Records: records #LogRecords:`	The backup having the ID *backup_id* has been completed; for more information, see Section 17.5.3.2, "Using The MySQL Cluster Management Client to Create a Backup"	`BackupCompleted`	`Backup`	7	`INFO`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Log Message	Description	Event Name	Event Type	Priority	Severity
log_records Data: *data_bytes* bytes Log: *log_bytes* bytes					
Node *node_id*: Backup request from *mgm_node_id* failed to start. Error: *error_code*	The backup failed to start; for error codes, see MGM API Errors	BackupFailedToStart	Backup	7	ALERT
Node *node_id*: Backup *backup_id* started from *mgm_node_id* has been aborted. Error: *error_code*	The backup was terminated after starting, possibly due to user intervention	BackupAborted	Backup	7	ALERT

17.5.7.2 MySQL Cluster Log Startup Messages

Possible startup messages with descriptions are provided in the following list:

- Initial start, waiting for %s to connect, nodes [all: %s connected: %s no-wait: %s]
- Waiting until nodes: %s connects, nodes [all: %s connected: %s no-wait: %s]
- Waiting %u sec for nodes %s to connect, nodes [all: %s connected: %s no-wait: %s]
- Waiting for non partitioned start, nodes [all: %s connected: %s missing: %s no-wait: %s]
- Waiting %u sec for non partitioned start, nodes [all: %s connected: %s missing: %s no-wait: %s]
- Initial start with nodes %s [missing: %s no-wait: %s]
- Start with all nodes %s
- Start with nodes %s [missing: %s no-wait: %s]
- Start potentially partitioned with nodes %s [missing: %s no-wait: %s]
- Unknown startreport: 0x%x [%s %s %s %s]

17.5.7.3 MySQL Cluster: NDB Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Error Code	Error Name	Error Text
0x00	**TE_NO_ERROR**	No error
0x01	**TE_ERROR_CLOSING_SOCKET**	Error found during closing of socket
0x02	**TE_ERROR_IN_SELECT_BEFORE_ACCEPT**	Error found before accept. The transporter will retry
0x03	**TE_INVALID_MESSAGE_LENGTH**	Error found in message (invalid message length)
0x04	**TE_INVALID_CHECKSUM**	Error found in message (checksum)
0x05	**TE_COULD_NOT_CREATE_SOCKET**	Error found while creating socket(can't create socket)
0x06	**TE_COULD_NOT_BIND_SOCKET**	Error found while binding server socket
0x07	**TE_LISTEN_FAILED**	Error found while listening to server socket
0x08	**TE_ACCEPT_RETURN_ERROR**	Error found during accept(accept return error)
0x0b	**TE_SHM_DISCONNECT**	The remote node has disconnected
0x0c	**TE_SHM_IPC_STAT**	Unable to check shm segment
0x0d	**TE_SHM_UNABLE_TO_CREATE_SEGMENT**	Unable to create shm segment
0x0e	**TE_SHM_UNABLE_TO_ATTACH_SEGMENT**	Unable to attach shm segment
0x0f	**TE_SHM_UNABLE_TO_REMOVE_SEGMENT**	Unable to remove shm segment
0x10	**TE_TOO_SMALL_SIGID**	Sig ID too small
0x11	**TE_TOO_LARGE_SIGID**	Sig ID too large
0x12	**TE_WAIT_STACK_FULL**	Wait stack was full
0x13	**TE_RECEIVE_BUFFER_FULL**	Receive buffer was full
0x14	**TE_SIGNAL_LOST_SEND_BUFFER_FULL**	Send buffer was full,and trying to force send fails
0x15	**TE_SIGNAL_LOST**	Send failed for unknown reason(signal lost)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Error Code	Error Name	Error Text
0x16	**TE_SEND_BUFFER_FULL**	The send buffer was full, but sleeping for a while solved
0x0017	**TE_SCI_LINK_ERROR**	There is no link from this node to the switch
0x18	**TE_SCI_UNABLE_TO_START_SEQUENCE**	Could not start a sequence, because system resources are exumed or no sequence has been created
0x19	**TE_SCI_UNABLE_TO_REMOVE_SEQUENCE**	Could not remove a sequence
0x1a	**TE_SCI_UNABLE_TO_CREATE_SEQUENCE**	Could not create a sequence, because system resources are exempted. Must reboot
0x1b	**TE_SCI_UNRECOVERABLE_DATA_TFX_ERROR**	Tried to send data on redundant link but failed
0x1c	**TE_SCI_CANNOT_INIT_LOCALSEGMENT**	Cannot initialize local segment
0x1d	**TE_SCI_CANNOT_MAP_REMOTESEGMENT**	Cannot map remote segment
0x1e	**TE_SCI_UNABLE_TO_UNMAP_SEGMENT**	Cannot free the resources used by this segment (step 1)
0x1f	**TE_SCI_UNABLE_TO_REMOVE_SEGMENT**	Cannot free the resources used by this segment (step 2)
0x20	**TE_SCI_UNABLE_TO_DISCONNECT_SEGMENT**	Cannot disconnect from a remote segment
0x21	**TE_SHM_IPC_PERMANENT**	Shm ipc Permanent error
0x22	**TE_SCI_UNABLE_TO_CLOSE_CHANNEL**	Unable to close the sci channel and the resources allocated

17.5.8 MySQL Cluster Single User Mode

Single user mode enables the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can use the `ALL STATUS` command to see when the cluster has entered single user mode.

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is `5` becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node will be rejected.

Note

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes—such as MySQL Servers—waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:

  1. Finish all single user mode transactions
  2. Issue the `EXIT SINGLE USER MODE` command
  3. Restart the cluster's data nodes

- Method 2:

  Restart database nodes prior to entering single user mode.

17.5.9 Quick Reference: MySQL Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to a MySQL Cluster, and in some cases provide information about the cluster itself.

- `SHOW ENGINE NDB STATUS`, `SHOW ENGINE NDBCLUSTER STATUS`

  The output of this statement contains information about the server's connection to the cluster, creation and usage of MySQL Cluster objects, and binary logging for MySQL Cluster replication.

  See Section 13.7.5.12, “SHOW ENGINE Syntax”, for a usage example and more detailed information.

- `SHOW ENGINES [LIKE 'NDB%']`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

See Section 13.7.5.13, “SHOW ENGINES Syntax”, for more detailed information.

- `SHOW VARIABLES LIKE 'NDB%'`

  This statement provides a list of most server system variables relating to the `NDB` storage engine, and their values, as shown here:

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| ndb_autoincrement_prefetch_sz     | 32    |
| ndb_cache_check_time              | 0     |
| ndb_extra_logging                 | 0     |
| ndb_force_send                    | ON    |
| ndb_index_stat_cache_entries      | 32    |
| ndb_index_stat_enable             | OFF   |
| ndb_index_stat_update_freq        | 20    |
| ndb_report_thresh_binlog_epoch_slip | 3     |
| ndb_report_thresh_binlog_mem_usage | 10    |
| ndb_use_copying_alter_table       | OFF   |
| ndb_use_exact_count               | ON    |
| ndb_use_transactions              | ON    |
+-----------------------------------+-------+
```

  See Section 5.1.4, “Server System Variables”, for more information.

- `SHOW STATUS LIKE 'NDB%'`

  This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the host name and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```
mysql> SHOW STATUS LIKE 'NDB%';
+--------------------------+---------------+
| Variable_name            | Value         |
+--------------------------+---------------+
| Ndb_cluster_node_id      | 10            |
| Ndb_config_from_host     | 192.168.0.103 |
| Ndb_config_from_port     | 1186          |
| Ndb_number_of_data_nodes | 4             |
+--------------------------+---------------+
```

  If the MySQL server was built with clustering support, but it is not connected to a cluster, all rows in the output of this statement contain a zero or an empty string:

```
mysql> SHOW STATUS LIKE 'NDB%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Ndb_cluster_node_id      | 0     |
| Ndb_config_from_host     |       |
| Ndb_config_from_port     | 0     |
| Ndb_number_of_data_nodes | 0     |
+--------------------------+-------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

See also Section 13.7.5.32, “SHOW STATUS Syntax”.

17.5.10 MySQL Cluster Security Issues

This section discusses security considerations to take into account when setting up and running MySQL Cluster.

Topics covered in this chapter include the following:

- MySQL Cluster and network security issues
- Configuration issues relating to running MySQL Cluster securely
- MySQL Cluster and the MySQL privilege system
- MySQL standard security procedures as applicable to MySQL Cluster

17.5.10.1 MySQL Cluster Security and Networking Issues

In this section, we discuss basic network security issues as they relate to MySQL Cluster. It is extremely important to remember that MySQL Cluster “out of the box” is not secure; you or your network administrator must take the proper steps to ensure that your cluster cannot be compromised over the network.

Cluster communication protocols are inherently insecure, and no encryption or similar security measures are used in communications between nodes in the cluster. Because network speed and latency have a direct impact on the cluster's efficiency, it is also not advisable to employ SSL or other encryption to network connections between nodes, as such schemes will effectively slow communications.

It is also true that no authentication is used for controlling API node access to a MySQL Cluster. As with encryption, the overhead of imposing authentication requirements would have an adverse impact on Cluster performance.

In addition, there is no checking of the source IP address for either of the following when accessing the cluster:

- SQL or API nodes using “free slots” created by empty `[mysqld]` or `[api]` sections in the `config.ini` file

  This means that, if there are any empty `[mysqld]` or `[api]` sections in the `config.ini` file, then any API nodes (including SQL nodes) that know the management server's host name (or IP address) and port can connect to the cluster and access its data without restriction. (See Section 17.5.10.2, “MySQL Cluster and MySQL Privileges”, for more information about this and related issues.)

Note

You can exercise some control over SQL and API node access to the cluster by specifying a `HostName` parameter for all `[mysqld]` and `[api]` sections in the `config.ini` file. However, this also means that, should you wish to connect an API node to the cluster from a previously unused host, you need to add an `[api]` section containing its host name to the `config.ini` file.

More information is available elsewhere in this chapter about the `HostName` parameter. Also see Section 17.3.1, “Quick Test Setup of MySQL Cluster”, for configuration examples using `HostName` with API nodes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Any `ndb_mgm` client

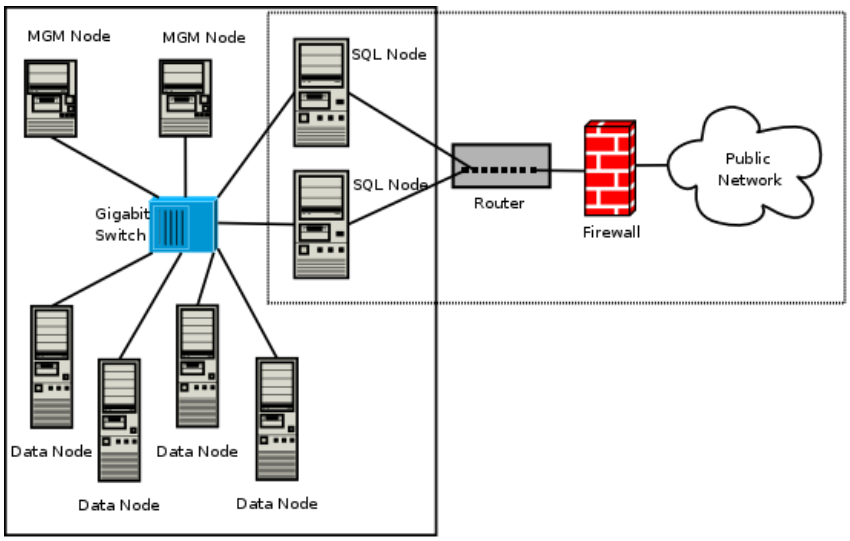
  This means that any cluster management client that is given the management server's host name (or IP address) and port (if not the standard port) can connect to the cluster and execute any management client command. This includes commands such as `ALL STOP` and `SHUTDOWN`.

For these reasons, it is necessary to protect the cluster on the network level. The safest network configuration for Cluster is one which isolates connections between Cluster nodes from any other network communications. This can be accomplished by any of the following methods:

1. Keeping Cluster nodes on a network that is physically separate from any public networks. This option is the most dependable; however, it is the most expensive to implement.

   We show an example of a MySQL Cluster setup using such a physically segregated network here:

   Figure 17.7 MySQL Cluster with Hardware Firewall

   This setup has two networks, one private (solid box) for the Cluster management servers and data nodes, and one public (dotted box) where the SQL nodes reside. (We show the management and data nodes connected using a gigabit switch since this provides the best performance.) Both networks are protected from the outside by a hardware firewall, sometimes also known as a *network-based firewall*.

   This network setup is safest because no packets can reach the cluster's management or data nodes from outside the network—and none of the cluster's internal communications can reach the outside—without going through the SQL nodes, as long as the SQL nodes do not permit any packets to be forwarded. This means, of course, that all SQL nodes must be secured against hacking attempts.

   Important

   With regard to potential security vulnerabilities, an SQL node is no different from any other MySQL server. See Section 6.1.3, “Making MySQL Secure Against Attackers”, for a description of techniques you can use to secure MySQL servers.

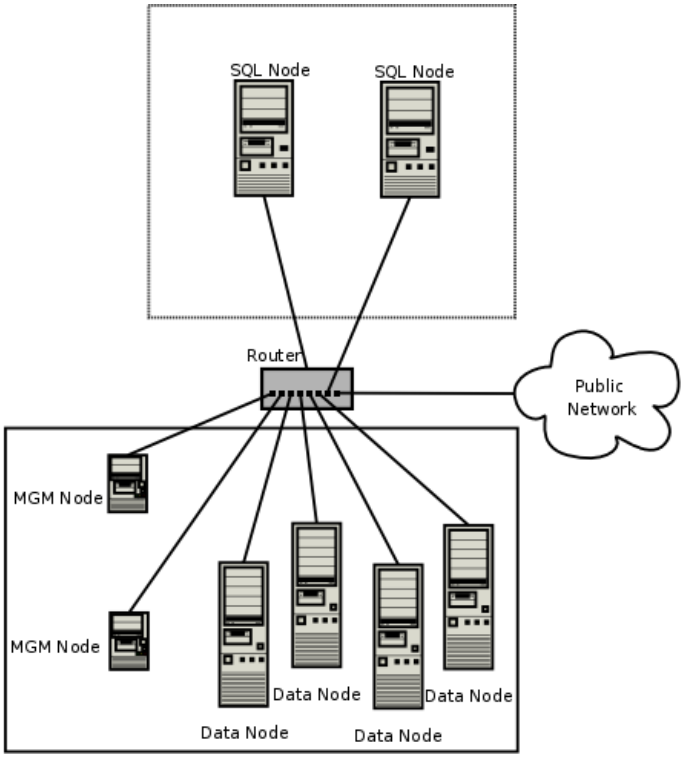
2. Using one or more software firewalls (also known as *host-based firewalls*) to control which packets pass through to the cluster from portions of the network that do not require access to it. In this type of setup, a software firewall must be installed on every host in the cluster which might otherwise be accessible from outside the local network.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The host-based option is the least expensive to implement, but relies purely on software to provide protection and so is the most difficult to keep secure.

This type of network setup for MySQL Cluster is illustrated here:

Figure 17.8 MySQL Cluster with Software Firewalls



Using this type of network setup means that there are two zones of MySQL Cluster hosts. Each cluster host must be able to communicate with all of the other machines in the cluster, but only those hosting SQL nodes (dotted box) can be permitted to have any contact with the outside, while those in the zone containing the data nodes and management nodes (solid box) must be isolated from any machines that are not part of the cluster. Applications using the cluster and user of those applications must *not* be permitted to have direct access to the management and data node hosts.

To accomplish this, you must set up software firewalls that limit the traffic to the type or types shown in the following table, according to the type of node that is running on each cluster host computer:

Type of Node to be Accessed	**Traffic to Permit**
SQL or API node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from within the network in which the cluster resides and is on the port that your application is using.
Data node or Management node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from the IP address of an SQL or API node.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

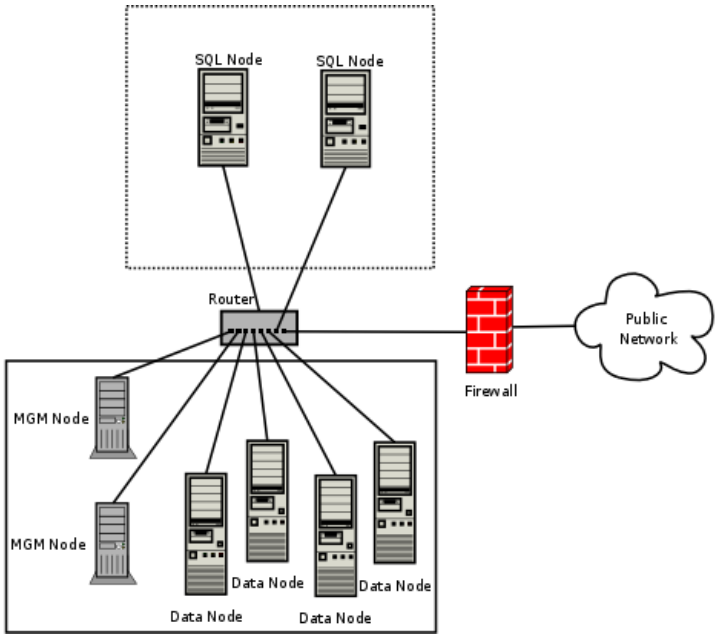
Any traffic other than that shown in the table for a given node type should be denied.

The specifics of configuring a firewall vary from firewall application to firewall application, and are beyond the scope of this Manual. `iptables` is a very common and reliable firewall application, which is often used with `APF` as a front end to make configuration easier. You can (and should) consult the documentation for the software firewall that you employ, should you choose to implement a MySQL Cluster network setup of this type, or of a “mixed” type as discussed under the next item.

3. It is also possible to employ a combination of the first two methods, using both hardware and software to secure the cluster—that is, using both network-based and host-based firewalls. This is between the first two schemes in terms of both security level and cost. This type of network setup keeps the cluster behind the hardware firewall, but permits incoming packets to travel beyond the router connecting all cluster hosts to reach the SQL nodes.

   One possible network deployment of a MySQL Cluster using hardware and software firewalls in combination is shown here:

   Figure 17.9 MySQL Cluster with a Combination of Hardware and Software Firewalls



   In this case, you can set the rules in the hardware firewall to deny any external traffic except to SQL nodes and API nodes, and then permit traffic to them only on the ports required by your application.

Whatever network configuration you use, remember that your objective from the viewpoint of keeping the cluster secure remains the same—to prevent any unessential traffic from reaching the cluster while ensuring the most efficient communication between the nodes in the cluster.

Because MySQL Cluster requires large numbers of ports to be open for communications between nodes, the recommended option is to use a segregated network. This represents the simplest way to prevent unwanted traffic from reaching the cluster.

Note

If you wish to administer a MySQL Cluster remotely (that is, from outside the local network), the recommended way to do this is to use `ssh` or another secure

This documentation is for an older version. If you're
This documentation is for an older version. If you're

login shell to access an SQL node host. From this host, you can then run the management client to access the management server safely, from within the Cluster's own local network.

Even though it is possible to do so in theory, it is *not* recommended to use `ndb_mgm` to manage a Cluster directly from outside the local network on which the Cluster is running. Since neither authentication nor encryption takes place between the management client and the management server, this represents an extremely insecure means of managing the cluster, and is almost certain to be compromised sooner or later.

17.5.10.2 MySQL Cluster and MySQL Privileges

In this section, we discuss how the MySQL privilege system works in relation to MySQL Cluster and the implications of this for keeping a MySQL Cluster secure.

Standard MySQL privileges apply to MySQL Cluster tables. This includes all MySQL privilege types (`SELECT` privilege, `UPDATE` privilege, `DELETE` privilege, and so on) granted on the database, table, and column level. As with any other MySQL Server, user and privilege information is stored in the `mysql` system database. The SQL statements used to grant and revoke privileges on `NDB` tables, databases containing such tables, and columns within such tables are identical in all respects with the `GRANT` and `REVOKE` statements used in connection with database objects involving any (other) MySQL storage engine. The same thing is true with respect to the `CREATE USER` and `DROP USER` statements.

It is important to keep in mind that the MySQL grant tables use the `MyISAM` storage engine. Because of this, those tables are not duplicated or shared among MySQL servers acting as SQL nodes in a MySQL Cluster. By way of example, suppose that two SQL nodes **A** and **B** are connected to the same MySQL Cluster, which has an `NDB` table named `mytable` in a database named `mydb`, and that you execute an SQL statement on server **A** that creates a new user `jon@localhost` and grants this user the `SELECT` privilege on that table:

```
mysql> GRANT SELECT ON mydb.mytable
    ->   TO jon@localhost IDENTIFIED BY 'mypass';
```

This user is *not* created on server **B**. For this to take place, the statement must also be run on server **B**. Similarly, statements run on server **A** and affecting the privileges of existing users on server **A** do not affect users on server **B** unless those statements are actually run on server **B** as well.

In other words, *changes in users and their privileges do not automatically propagate between SQL nodes*. Synchronization of privileges between SQL nodes must be done either manually or by scripting an application that periodically synchronizes the privilege tables on all SQL nodes in the cluster.

Conversely, because there is no way in MySQL to deny privileges (privileges can either be revoked or not granted in the first place, but not denied as such), there is no special protection for `NDB` tables on one SQL node from users that have privileges on another SQL node. The most far-reaching example of this is the MySQL `root` account, which can perform any action on any database object. In combination with empty `[mysqld]` or `[api]` sections of the `config.ini` file, this account can be especially dangerous. To understand why, consider the following scenario:

- The `config.ini` file contains at least one empty `[mysqld]` or `[api]` section. This means that the Cluster management server performs no checking of the host from which a MySQL Server (or other API node) accesses the MySQL Cluster.
- There is no firewall, or the firewall fails to protect against access to the Cluster from hosts external to the network.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The host name or IP address of the Cluster's management server is known or can be determined from outside the network.

If these conditions are true, then anyone, anywhere can start a MySQL Server with `--ndbcluster --ndb-connectstring=management_host` and access the Cluster. Using the MySQL `root` account, this person can then perform the following actions:

- Execute a `SHOW DATABASES` statement to obtain a list of all databases that exist in the cluster
- Execute a `SHOW TABLES FROM some_database` statement to obtain a list of all `NDB` tables in a given database
- Run any legal MySQL statements on any of those tables, such as:
  - `SELECT * FROM some_table` to read all the data from any table
  - `DELETE FROM some_table` to delete all the data from a table
  - `DESCRIBE some_table` or `SHOW CREATE TABLE some_table` to determine the table schema
  - `UPDATE some_table SET column1 = any_value1` to fill a table column with “garbage” data; this could actually cause much greater damage than simply deleting all the data

    Even more insidious variations might include statements like these:

    ```
    UPDATE some_table SET an_int_column = an_int_column + 1
    ```

    or

    ```
    UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
    ```

    Such malicious statements are limited only by the imagination of the attacker.

  The only tables that would be safe from this sort of mayhem would be those tables that were created using storage engines other than `NDB`, and so not visible to a “rogue” SQL node.

Note

A user who can log in as `root` can also access the `INFORMATION_SCHEMA` database and its tables, and so obtain information about databases, tables, stored routines, scheduled events, and any other database objects for which metadata is stored in `INFORMATION_SCHEMA`.

  It is also a very good idea to use different passwords for the `root` accounts on different cluster SQL nodes.

In sum, you cannot have a safe MySQL Cluster if it is directly accessible from outside your local network.

Important

Never leave the MySQL root account password empty. This is just as true when running MySQL as a MySQL Cluster SQL node as it is when running it as a standalone (non-Cluster) MySQL Server, and should be done as part of the MySQL installation process before configuring the MySQL Server as an SQL node in a MySQL Cluster.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You should never convert the system tables in the `mysql` database to use the `NDB` storage engine. There are a number of reasons why you should not do this, but the most important reason is this: *Many of the SQL statements that affect `mysql` tables storing information about user privileges, stored routines, scheduled events, and other database objects cease to function if these tables are changed to use any storage engine other than `MyISAM`*. This is a consequence of various MySQL Server internals which are not expected to change in the foreseeable future.

If you need to synchronize `mysql` system tables between SQL nodes, you can use standard MySQL replication to do so, or employ a script to copy table entries between the MySQL servers.

Summary. The two most important points to remember regarding the MySQL privilege system with regard to MySQL Cluster are:

1. Users and privileges established on one SQL node do not automatically exist or take effect on other SQL nodes in the cluster.

   Conversely, removing a user or privilege on one SQL node in the cluster does not remove the user or privilege from any other SQL nodes.

2. Once a MySQL user is granted privileges on an `NDB` table from one SQL node in a MySQL Cluster, that user can “see” any data in that table regardless of the SQL node from which the data originated.

17.5.10.3 MySQL Cluster and MySQL Security Procedures

In this section, we discuss MySQL standard security procedures as they apply to running MySQL Cluster.

In general, any standard procedure for running MySQL securely also applies to running a MySQL Server as part of a MySQL Cluster. First and foremost, you should always run a MySQL Server as the `mysql` system user; this is no different from running MySQL in a standard (non-Cluster) environment. The `mysql` system account should be uniquely and clearly defined. Fortunately, this is the default behavior for a new MySQL installation. You can verify that the `mysqld` process is running as the system user `mysql` by using the system command such as the one shown here:

```
shell> ps aux | grep mysql
root     10467  0.0  0.1   3616  1380 pts/3    S    11:53   0:00 \
  /bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql    10512  0.2  2.5  58528 26636 pts/3    Sl   11:53   0:00 \
  /usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql \
  --datadir=/usr/local/mysql/var --user=mysql --ndbcluster \
  --ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid \
  --log-error=/usr/local/mysql/var/mothra.err
jon      10579  0.0  0.0   2736   688 pts/0    S+   11:54   0:00 grep mysql
```

If the `mysqld` process is running as any other user than `mysql`, you should immediately shut it down and restart it as the `mysql` user. If this user does not exist on the system, the `mysql` user account should be created, and this user should be part of the `mysql` user group; in this case, you should also make sure that the MySQL `DataDir` on this system is owned by the `mysql` user, and that the SQL node's `my.cnf` file includes `user=mysql` in the `[mysqld]` section. Alternatively, you can start the server with `--user=mysql` on the command line, but it is preferable to use the `my.cnf` option, since you might forget to use the command-line option and so have `mysqld` running as another user unintentionally. The `mysqld_safe` startup script forces MySQL to run as the `mysql` user.

Important

Never run `mysqld` as the system root user. Doing so means that potentially any file on the system can be read by MySQL, and thus—should MySQL be compromised—by an attacker.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As mentioned in the previous section (see Section 17.5.10.2, “MySQL Cluster and MySQL Privileges”), you should always set a root password for the MySQL Server as soon as you have it running. You should also delete the anonymous user account that is installed by default. You can accomplish these tasks using the following statements:

```
shell> mysql -u root

mysql> UPDATE mysql.user
    ->     SET Password=PASSWORD('secure_password')
    ->     WHERE User='root';

mysql> DELETE FROM mysql.user
    ->     WHERE User='';

mysql> FLUSH PRIVILEGES;
```

Be very careful when executing the `DELETE` statement not to omit the `WHERE` clause, or you risk deleting *all* MySQL users. Be sure to run the `FLUSH PRIVILEGES` statement as soon as you have modified the `mysql.user` table, so that the changes take immediate effect. Without `FLUSH PRIVILEGES`, the changes do not take effect until the next time that the server is restarted.

Note

Many of the MySQL Cluster utilities such as `ndb_show_tables`, `ndb_desc`, and `ndb_select_all` also work without authentication and can reveal table names, schemas, and data. By default these are installed on Unix-style systems with the permissions `wxr-xr-x` (755), which means they can be executed by any user that can access the `mysql/bin` directory.

See Section 17.4, “MySQL Cluster Programs”, for more information about these utilities.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 18 Stored Programs and Views

Table of Contents

This chapter discusses stored programs and views, which are database objects defined in terms of SQL code that is stored on the server for later execution.

Stored programs include these objects:

- Stored routines, that is, stored procedures and functions. A stored procedure is invoked using the `CALL` statement. A procedure does not have a return value but can modify its parameters for later inspection by the caller. It can also generate result sets to be returned to the client program. A stored function is used much like a built-in function. you invoke it in an expression and it returns a value during expression evaluation.
- Triggers. A trigger is a named database object that is associated with a table and that is activated when a particular event occurs for the table, such as an insert or update.

Views are stored queries that when referenced produce a result set. A view acts as a virtual table.

This chapter describes how to use stored programs and views. The following sections provide additional information about SQL syntax for statements related to these objects:

- For each object type, there are `CREATE`, `ALTER`, and `DROP` statements that control which objects exist and how they are defined. See Section 13.1, “Data Definition Statements”.
- The `CALL` statement is used to invoke stored procedures. See Section 13.2.1, “CALL Syntax”.
- Stored program definitions include a body that may use compound statements, loops, conditionals, and declared variables. See Section 13.6, “MySQL Compound-Statement Syntax”.

18.1 Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (`;`) characters. For example,

This documentation is for an older version. If you're

This documentation is for an older version. If you're

the following stored procedure has a body made up of a `BEGIN ... END` block that contains a `SET` statement and a `REPEAT` loop that itself contains another `SET` statement:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

If you use the `mysql` client program to define a stored program containing semicolon characters, a problem arises. By default, `mysql` itself recognizes the semicolon as a statement delimiter, so you must redefine the delimiter temporarily to cause `mysql` to pass the entire stored program definition to the server.

To redefine the `mysql` delimiter, use the `delimiter` command. The following example shows how to do this for the `dorepeat()` procedure just shown. The delimiter is changed to `//` to enable the entire definition to be passed to the server as a single statement, and then restored to `;` before invoking the procedure. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

You can redefine the delimiter to a string other than `//`, and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (“`\`”) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
1 row in set (0.00 sec)
```

18.2 Using Stored Routines (Procedures and Functions)

Stored routines (procedures and functions) are supported in MySQL 5.0. A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

Stored routines require the `proc` table in the `mysql` database. This table is created during the MySQL 5.0 installation procedure. If you are upgrading to MySQL 5.0 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

Stored routines can be particularly useful in certain situations:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also enable you to have libraries of functions in the database server. This is a feature shared by modern application languages that enable such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2. All syntax described here is supported and any limitations and extensions are documented where appropriate.

Additional Resources

- You may find the Stored Procedures User Forum of use when working with stored procedures and functions.
- For answers to some commonly asked questions regarding stored routines in MySQL, see Section A.4, “MySQL 5.0 FAQ: Stored Procedures and Functions”.
- There are some restrictions on the use of stored routines. See Section C.1, “Restrictions on Stored Programs”.
- Binary logging for stored routines takes place as described in Section 18.6, “Binary Logging of Stored Programs”.

18.2.1 Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with the `CREATE PROCEDURE` and `CREATE FUNCTION` statements (see Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”). A procedure is invoked using a `CALL` statement (see Section 13.2.1, “CALL Syntax”), and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar

This documentation is for an older version. If you're
This documentation is for an older version. If you're

value. The body of a stored routine can use compound statements (see Section 13.6, “MySQL Compound-Statement Syntax”).

Stored routines can be dropped with the `DROP PROCEDURE` and `DROP FUNCTION` statements (see Section 13.1.16, “DROP PROCEDURE and DROP FUNCTION Syntax”), and altered with the `ALTER PROCEDURE` and `ALTER FUNCTION` statements (see Section 13.1.3, “ALTER PROCEDURE Syntax”).

As of MySQL 5.0.1, a stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE` *`db_name`* is performed (and undone when the routine terminates). `USE` statements within stored routines are not permitted.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

(In MySQL 5.0.0, stored routines are global and not associated with a database. They inherit the default database from the caller. If a `USE` *`db_name`* is executed within the routine, the original default database is restored upon routine exit.)

Stored functions cannot be recursive.

Recursion in stored procedures is permitted but disabled by default. To enable recursion, set the `max_sp_recursion_depth` server system variable to a value greater than zero. Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth`, it may be necessary to increase thread stack size by increasing the value of `thread_stack` at server startup. See Section 5.1.4, “Server System Variables”, for more information.

MySQL supports a very useful extension that enables the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the `CLIENT_MULTI_RESULTS` option when it connects. For C programs, this can be done with the `mysql_real_connect()` C API function. See Section 20.6.7.52, “mysql_real_connect()”, and Section 20.6.16, “C API Support for Multiple Statement Execution”.

18.2.2 Stored Routines and MySQL Privileges

Beginning with MySQL 5.0.3, the grant system takes stored routines into account as follows:

- The `CREATE ROUTINE` privilege is needed to create stored routines.
- The `ALTER ROUTINE` privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped from the creator when the routine is dropped.
- The `EXECUTE` privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine if necessary (and dropped from the creator when the routine is dropped). Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which enables users who have access to the database with which the routine is associated to execute the routine.
- If the `automatic_sp_privileges` system variable is 0, the `EXECUTE` and `ALTER ROUTINE` privileges are not automatically granted to and dropped from the routine creator.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

The server manipulates the `mysql.proc` table in response to statements that create, alter, or drop stored routines. It is not supported that the server will notice manual manipulation of this table.

18.2.3 Stored Routine Metadata

Metadata about stored routines can be obtained as follows:

- Query the `ROUTINES` table of the `INFORMATION_SCHEMA` database. See Section 19.8, "The INFORMATION_SCHEMA ROUTINES Table".
- Use the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements to see routine definitions. See Section 13.7.5.8, "SHOW CREATE PROCEDURE Syntax".
- Use the `SHOW PROCEDURE STATUS` and `SHOW FUNCTION STATUS` statements to see routine characteristics. See Section 13.7.5.26, "SHOW PROCEDURE STATUS Syntax".
- `INFORMATION_SCHEMA` does not have a `PARAMETERS` table until MySQL 5.5, so applications that need to acquire routine parameter information at runtime must use workarounds such as parsing the output of `SHOW CREATE` statements or the `param_list` column of the `mysql.proc` table. `param_list` contents can be processed from within a stored routine, unlike the output from `SHOW`.

18.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID()

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects (see Section 12.13, "Information Functions"). The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value. (Before MySQL 5.0.12, the value is not restored and following statements do see a changed value.)

18.3 Using Triggers

Support for triggers is included beginning with MySQL 5.0.2. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events. For example, rows can be inserted by `INSERT` or `LOAD DATA` statements, and an insert trigger activates for each inserted row. A trigger can be set to activate either before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

Important

MySQL triggers activate only for changes made to tables by SQL statements. They do not activate for changes in tables made by APIs that do not transmit SQL statements to the MySQL Server; in particular, they are not activated by updates made using the `NDB` API.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To use triggers if you have upgraded to MySQL 5.0 from an older release that did not support triggers, you should upgrade your grant tables so that they contain the trigger-related privileges. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

The following sections describe the syntax for creating and dropping triggers, show some examples of how to use them, and indicate how to obtain trigger metadata.

Additional Resources

- You may find the Triggers User Forum of use when working with triggers.
- For answers to commonly asked questions regarding triggers in MySQL, see Section A.5, “MySQL 5.0 FAQ: Triggers”.
- There are some restrictions on the use of triggers; see Section C.1, “Restrictions on Stored Programs”.
- Binary logging for triggers takes place as described in Section 18.6, “Binary Logging of Stored Programs”.

18.3.1 Trigger Syntax and Examples

To create a trigger or drop a trigger, use the `CREATE TRIGGER` or `DROP TRIGGER` statement, described in Section 13.1.11, “CREATE TRIGGER Syntax”, and Section 13.1.18, “DROP TRIGGER Syntax”.

Here is a simple example that associates a trigger with a table, to activate for `INSERT` operations. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

The `CREATE TRIGGER` statement creates a trigger named `ins_sum` that is associated with the `account` table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

- The keyword `BEFORE` indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is `AFTER`.
- The keyword `INSERT` indicates the trigger event; that is, the type of operation that activates the trigger. In the example, `INSERT` operations cause trigger activation. You can also create triggers for `DELETE` and `UPDATE` operations.
- The statement following `FOR EACH ROW` defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event. In the example, the trigger body is a simple `SET` that accumulates into a user variable the values inserted into the `amount` column. The statement refers to the column as `NEW.amount` which means “the value of the `amount` column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an `INSERT` statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+----------------------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

In this case, the value of `@sum` after the `INSERT` statement has executed is `14.98 + 1937.50 - 100`, or `1852.48`.

To destroy the trigger, use a `DROP TRIGGER` statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

In addition to the requirement that trigger names be unique for a schema, there are other limitations on the types of triggers you can create. In particular, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. To work around this, you can define a trigger that executes multiple statements by using the `BEGIN ... END` compound statement construct after `FOR EACH ROW`. (An example appears later in this section.)

Within the trigger body, the `OLD` and `NEW` keywords enable you to access columns in the rows affected by a trigger. `OLD` and `NEW` are MySQL extensions to triggers; they are not case sensitive.

In an `INSERT` trigger, only `NEW.col_name` can be used; there is no old row. In a `DELETE` trigger, only `OLD.col_name` can be used; there is no new row. In an `UPDATE` trigger, you can use `OLD.col_name` to refer to the columns of a row before it is updated and `NEW.col_name` to refer to the columns of the row after it is updated.

A column named with `OLD` is read only. You can refer to it (if you have the `SELECT` privilege), but not modify it. You can refer to a column named with `NEW` if you have the `SELECT` privilege for it. In a `BEFORE` trigger, you can also change its value with `SET NEW.col_name = value` if you have the `UPDATE` privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a `SET` statement has no effect in an `AFTER` trigger because the row change will have already occurred.)

In a `BEFORE` trigger, the `NEW` value for an `AUTO_INCREMENT` column is 0, not the sequence number that is generated automatically when the new row actually is inserted.

By using the `BEGIN ... END` construct, you can define a trigger that executes multiple statements. Within the `BEGIN` block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the `mysql` program to define a trigger that executes multiple statements, it is necessary to redefine the `mysql` statement delimiter so that you can use the `;` statement delimiter within the trigger definition. The following example illustrates these points. It defines an `UPDATE` trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a `BEFORE` trigger because the value must be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple `CALL` statement. This is also advantageous if you want to execute the same code from within several triggers.

There are limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the `CALL` statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through `OUT` or `INOUT` parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction, such as `START TRANSACTION`, `COMMIT`, or `ROLLBACK`. (`ROLLBACK to SAVEPOINT` is permitted because it does not end a transaction.).
- Prior to MySQL 5.0.10, triggers cannot contain direct references to tables by name.

See also Section C.1, “Restrictions on Stored Programs”.

MySQL handles errors during trigger execution as follows:

- If a `BEFORE` trigger fails, the operation on the corresponding row is not performed.
- A `BEFORE` trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An `AFTER` trigger is executed only if any `BEFORE` triggers and the row operation execute successfully.
- An error during either a `BEFORE` or `AFTER` trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Before MySQL 5.0.10, triggers cannot contain direct references to tables by name. Beginning with MySQL 5.0.10, you can write triggers such as the one named `testref` shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

delimiter |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
|

delimiter ;

INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Suppose that you insert the following values into table `test1` as shown here:

```
mysql> INSERT INTO test1 VALUES
    -> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

As a result, the four tables contain the following data:

```
mysql> SELECT * FROM test1;
+------+
| a1   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+----+
| a3 |
+----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+----+------+
| a4 | b4   |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+----+------+
|  1 |    3 |
|  2 |    0 |
|  3 |    1 |
|  4 |    2 |
|  5 |    0 |
|  6 |    0 |
|  7 |    1 |
|  8 |    1 |
|  9 |    0 |
| 10 |    0 |
+----+------+
10 rows in set (0.00 sec)
```

18.3.2 Trigger Metadata

Metadata about triggers can be obtained as follows:

- Query the `TRIGGERS` table of the `INFORMATION_SCHEMA` database. See Section 19.15, “The INFORMATION_SCHEMA TRIGGERS Table”.
- Use the `SHOW TRIGGERS` statement. See Section 13.7.5.35, “SHOW TRIGGERS Syntax”.

18.4 Using Views

MySQL supports views, including updatable views. Views are stored queries that when invoked produce a result set. A view acts as a virtual table.

To use views if you have upgraded to MySQL 5.0.1 from an older release, you should upgrade your grant tables so that they contain the view-related privileges. See Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”.

The following discussion describes the syntax for creating and dropping views, and shows some examples of how to use them.

Additional Resources

- You may find the Views User Forum of use when working with views.
- For answers to some commonly asked questions regarding views in MySQL, see Section A.6, “MySQL 5.0 FAQ: Views”.
- There are some restrictions on the use of views; see Section C.4, “Restrictions on Views”.

18.4.1 View Syntax

The `CREATE VIEW` statement creates a new view (see Section 13.1.12, “CREATE VIEW Syntax”). To alter the definition of a view or drop a view, use `ALTER VIEW` (see Section 13.1.5, “ALTER VIEW Syntax”), or `DROP VIEW` (see Section 13.1.19, “DROP VIEW Syntax”).

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
|    5 |    60 |   300 |
+------+-------+-------+
mysql> SELECT * FROM v WHERE qty = 5;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    5 |    60 |   300 |
+------+-------+-------+
```

18.4.2 View Processing Algorithms

The optional `ALGORITHM` clause for `CREATE VIEW` or `ALTER VIEW` is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`.

- For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.
- For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.
- For `UNDEFINED`, MySQL chooses which algorithm to use. It prefers `MERGE` over `TEMPTABLE` if possible, because `MERGE` is usually more efficient and because a view cannot be updatable if a temporary table is used.
- If no `ALGORITHM` clause is present, `UNDEFINED` is the default algorithm.

A reason to specify `TEMPTABLE` explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the `MERGE` algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be `UNDEFINED` for three reasons:

- No `ALGORITHM` clause is present in the `CREATE VIEW` statement.
- The `CREATE VIEW` statement has an explicit `ALGORITHM = UNDEFINED` clause.
- `ALGORITHM = MERGE` is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to `UNDEFINED`.

As mentioned earlier, `MERGE` is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the `MERGE` algorithm works. The examples assume that there is a view `v_merge` that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- `v_merge` becomes `t`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `*` becomes `vc1, vc2`, which corresponds to `c1, c2`
- The view `WHERE` clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

If the `MERGE` algorithm cannot be used, a temporary table must be used instead. `MERGE` cannot be used if the view contains any of the following constructs:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `LIMIT`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Assignment to user variables
- Refers only to literal values (in this case, there is no underlying table)

18.4.3 Updatable and Insertable Views

Some views are updatable and references to them can be used to specify tables to be updated in data change statements. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table.

For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Certain joins (see additional join discussion later in this section)
- Reference to nonupdatable view in the `FROM` clause
- Subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are not permitted even though they might be theoretically updatable.

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references. They must not be expressions or composite expressions, such as these:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `INFORMATION_SCHEMA.VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable.

If a view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and are rejected. (Note that even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the `updatable_views_with_limit` system variable. See Section 5.1.4, “Server System Variables”.

Earlier discussion in this section pointed out that a view is not insertable if not all columns are simple column references (for example, if it contains columns that are expressions or composite expressions). Although such a view is not insertable, it can be updatable if you update only columns that are not expressions. Consider this view:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is an expression. But it is updatable if the update does not try to update `col2`. This update is permissible:

```
UPDATE v SET col1 = 0;
```

This update is not permissible because it attempts to update an expression column:

```
UPDATE v SET col2 = 0;
```

For a multiple-table updatable view, `INSERT` can work if it inserts into a single table. `DELETE` is not supported.

`INSERT DELAYED` is not supported for views.

If a table contains an `AUTO_INCREMENT` column, inserting into an insertable view on the table that does not include the `AUTO_INCREMENT` column does not change the value of `LAST_INSERT_ID()`, because the side effects of inserting default values into columns not part of the view should not be visible.

18.4.4 The View WITH CHECK OPTION Clause

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts to rows for which the `WHERE` clause in the *`select_statement`* is not true. It aslo prevents updates to rows for which the `WHERE` clause is true but the update would cause it to be not true (in other words, it prevents visible rows from being updated to nonvisible rows).

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. When neither keyword is given, the default is `CASCADED`. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well.

Consider the definitions for the following table and set of views:

```
CREATE TABLE t1 (a INT);
CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
WITH CHECK OPTION;
CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
WITH LOCAL CHECK OPTION;
CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
WITH CASCADED CHECK OPTION;
```

Here the `v2` and `v3` views are defined in terms of another view, `v1`. `v2` has a `LOCAL` check option, so inserts are tested only against the `v2` check. `v3` has a `CASCADED` check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

18.4.5 View Metadata

Metadata about views can be obtained as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Query the `VIEWS` table of the `INFORMATION_SCHEMA` database. See Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”.
- Use the `SHOW CREATE VIEW` statement. See Section 13.7.5.10, “SHOW CREATE VIEW Syntax”.

18.5 Access Control for Stored Programs and Views

Stored programs and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. These privileges are controlled by their `DEFINER` attribute, and, if there is one, their `SQL SECURITY` characteristic.

All stored programs (procedures, functions, and triggers) and views can have a `DEFINER` attribute that names a MySQL account. If the `DEFINER` attribute is omitted from a stored program or view definition, the default account is the user who creates the object.

In addition, stored routines (procedures and functions) and views can have a `SQL SECURITY` characteristic with a value of `DEFINER` or `INVOKER` to specify whether the object executes in definer or invoker context. If the `SQL SECURITY` characteristic is omitted, the default is definer context.

Triggers have no `SQL SECURITY` characteristic and always execute in definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

Definer and invoker security contexts differ as follows:

- A stored program or view that executes in definer security context executes with the privileges of the account named by its `DEFINER` attribute. These privileges may be entirely different from those of the invoking user. The invoker must have appropriate privileges to reference the object (for example, `EXECUTE` to call a stored procedure or `SELECT` to select from a view), but when the object executes, the invoker's privileges are ignored and only the `DEFINER` account privileges matter. If this account has few privileges, the object is correspondingly limited in the operations it can perform. If the `DEFINER` account is highly privileged (such as a `root` account), the object can perform powerful operations *no matter who invokes it.*
- A stored routine or view that executes in invoker security context can perform only operations for which the invoker has privileges. The `DEFINER` attribute can be specified but has no effect for objects that execute in invoker context.

Consider the following stored procedure:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Any user who has the `EXECUTE` privilege for `p1` can invoke it with a `CALL` statement. However, when `p1` executes, it does so in `DEFINER` security context and thus executes with the privileges of `'admin'@'localhost'`, the account named in the `DEFINER` attribute. This account must have the `EXECUTE` privilege for `p1` as well as the `UPDATE` privilege for the table `t1`. Otherwise, the procedure fails.

Now consider this stored procedure, which is identical to `p1` except that its `SQL SECURITY` characteristic is `INVOKER`:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()
SQL SECURITY INVOKER
BEGIN
  UPDATE t1 SET counter = counter + 1;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
END;
```

`p2`, unlike `p1`, executes in `INVOKER` security context. The `DEFINER` attribute is irrelevant and `p2` executes with the privileges of the invoking user. `p2` fails if the invoker lacks the `EXECUTE` privilege for `p2` or the `UPDATE` privilege for the table `t1`.

MySQL uses the following rules to control which accounts a user can specify in an object `DEFINER` attribute:

- You can specify a `DEFINER` value other than your own account only if you have the `SUPER` privilege.
- If you do not have the `SUPER` privilege, the only legal user value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.

To minimize the risk potential for stored program and view creation and use, follow these guidelines:

- For a stored routine or view, use `SQL SECURITY INVOKER` in the object definition when possible so that it can be used only by users with permissions appropriate for the operations performed by the object.
- If you create definer-context stored programs or views while using an account that has the `SUPER` privilege, specify an explicit `DEFINER` attribute that names an account possessing only the privileges required for the operations performed by the object. Specify a highly privileged `DEFINER` account only when absolutely necessary.
- Administrators can prevent users from specifying highly privileged `DEFINER` accounts by not granting them the `SUPER` privilege.
- Definer-context objects should be written keeping in mind that they may be able to access data for which the invoking user has no privileges. In some cases, you can prevent reference to these objects by not granting unauthorized users particular privileges:
  - A stored procedure or function cannot be referenced by a user who does not have the `EXECUTE` privilege for it.
  - A view cannot be referenced by a user who does not have the appropriate privilege for it (`SELECT` to select from it, `INSERT` to insert into it, and so forth).

  However, no such control exists for triggers because users do not reference them directly. A trigger always executes in `DEFINER` context and is activated by access to the table with which it is associated, even ordinary table accesses by users with no special privileges. If the `DEFINER` account is highly privileged, the trigger can perform sensitive or dangerous operations. This remains true if the `SUPER` privilege needed to create the trigger is revoked from the account of the user who created it. Administrators should be especially careful about granting users that privilege.

18.6 Binary Logging of Stored Programs

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. The binary log has two important purposes:

- For replication, the binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 16.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These

This documentation is for an older version. If you're
This documentation is for an older version. If you're

events bring databases up to date from the point of the backup. See Section 7.3.2, “Using Backups for Recovery”.

However, there are certain binary logging issues that apply with respect to stored programs (stored procedures and functions, and triggers):

- Logging occurs at the statement level. In some cases, it is possible that a statement will affect different sets of rows on a master and a slave.
- Replicated statements executed on a slave are processed by the slave SQL thread, which has full privileges. It is possible for a procedure to follow different execution paths on master and slave servers, so a user can write a routine containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges.
- If a stored program that modifies data is nondeterministic, it is not repeatable. This can result in different data on a master and slave, or cause restored data to differ from the original data.

This section describes how MySQL 5.0 handles binary logging for stored programs. The discussion first states the current conditions that the implementation places on the use of stored programs, and what you can do to avoid problems. Then it summarizes the changes that have taken place in the logging implementation. Finally, implementation details are given that provide information about when and why various changes were made. These details show how several aspects of the current logging behavior were implemented in response to shortcomings identified in earlier versions of MySQL.

In general, the issues described here occur due to the fact that binary logging occurs at the SQL statement level. MySQL 5.1 implements row-level binary logging, which solves or alleviates these issues because the log contains changes made to individual rows as a result of executing SQL statements.

Unless noted otherwise, the remarks here assume that you have enabled binary logging by starting the server with the `--log-bin` option. (See Section 5.4.3, “The Binary Log”.) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The current conditions on the use of stored functions in MySQL 5.0 can be summarized as follows. These conditions do not apply to stored procedures and they do not apply unless binary logging is enabled.

- To create or alter a stored function, you must have the `SUPER` privilege, in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege that is normally required. (Depending on the `DEFINER` value in the function definition, `SUPER` might be required regardless of whether binary logging is enabled. See Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)
- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.

  By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

  This function is deterministic (and does not modify data), so it is safe:

```
CREATE FUNCTION f1(i INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
BEGIN
  RETURN i;
END;
```

This function uses `UUID()`, which is not deterministic, so the function also is not deterministic and is not safe:

```
CREATE FUNCTION f2()
RETURNS CHAR(36) CHARACTER SET utf8
BEGIN
  RETURN UUID();
END;
```

This function modifies data, so it may not be safe:

```
CREATE FUNCTION f3(p_id INT)
RETURNS INT
BEGIN
  UPDATE t SET modtime = NOW() WHERE id = p_id;
  RETURN ROW_COUNT();
END;
```

Assessment of the nature of a function is based on the "honesty" of the creator: MySQL does not check that a function declared `DETERMINISTIC` is free of statements that produce nondeterministic results.

- To relax the preceding conditions on function creation (that you must have the `SUPER` privilege and that a function must be declared deterministic or to not modify data), set the global `log_bin_trust_function_creators` system variable to 1. By default, this variable has a value of 0, but you can change it like this:

  ```
  mysql> SET GLOBAL log_bin_trust_function_creators = 1;
  ```

  You can also set this variable by using the `--log-bin-trust-function-creators=1` option when starting the server.

  If binary logging is not enabled, `log_bin_trust_function_creators` does not apply. `SUPER` is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- For information about built-in functions that may be unsafe for replication (and thus cause stored functions that use them to be unsafe as well), see Section 16.4.1, "Replication Features and Issues".

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: `CREATE TRIGGER` does not have an optional `DETERMINISTIC` characteristic, so triggers are assumed to be always deterministic. However, this assumption might in some cases be invalid. For example, the `UUID()` function is nondeterministic (and does not replicate). You should be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with `CREATE TRIGGER` if you do not have the required privileges. On the slave side, the slave uses the trigger `DEFINER` attribute to determine which user is considered to be the creator of the trigger.

The rest of this section provides details on the development of stored routine logging. You need not read it unless you are interested in the background on the rationale for the current logging-related conditions on stored routine use.

The development of stored routine logging in MySQL 5.0 can be summarized as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Before MySQL 5.0.6: In the initial implementation of stored routine logging, statements that create stored routines and `CALL` statements are not logged. These omissions can cause problems for replication and data recovery.

- MySQL 5.0.6: Statements that create stored routines and `CALL` statements are logged. Stored function invocations are logged when they occur in statements that update data (because those statements are logged). However, function invocations are not logged when they occur in statements such as `SELECT` that do not change data, even if a data change occurs within a function itself; this can cause problems. Under some circumstances, functions and procedures can have different effects if executed at different times or on different (master and slave) machines, and thus can be unsafe for data recovery or replication. To handle this, measures are implemented to enable identification of safe routines and to prevent creation of unsafe routines except by users with sufficient privileges.

- MySQL 5.0.12: For stored functions, when a function invocation that changes data occurs within a nonlogged statement such as `SELECT`, the server logs a `DO func_name()` statement that invokes the function so that the function gets executed during data recovery or replication to slave servers. For stored procedures, the server does not log `CALL` statements. Instead, it logs individual statements within a procedure that are executed as a result of a `CALL`. This eliminates problems that may occur when a procedure would follow a different execution path on a slave than on the master.

- MySQL 5.0.16: The procedure logging changes made in 5.0.12 enable the conditions on unsafe routines to be relaxed for stored procedures. Consequently, the user interface for controlling these conditions is revised to apply only to functions. Procedure creators are no longer bound by them.

- MySQL 5.0.17: Logging of stored functions as `DO func_name()` statements (per the changes made in 5.0.12) are logged as `SELECT func_name()` statements instead for better control over error checking.

Routine logging before MySQL 5.0.6: Statements that create and use stored routines are not written to the binary log, but statements invoked within stored routines are logged. Suppose that you issue the following statements:

```
CREATE PROCEDURE mysp INSERT INTO t VALUES(1);
CALL mysp();
```

For this example, only the `INSERT` statement appears in the binary log. The `CREATE PROCEDURE` and `CALL` statements do not appear. The absence of routine-related statements in the binary log means that stored routines are not replicated correctly. It also means that for a data recovery operation, re-executing events in the binary log does not recover stored routines.

Routine logging changes in MySQL 5.0.6: To address the absence of logging for stored routine creation and `CALL` statements (and the consequent replication and data recovery concerns), the characteristics of binary logging for stored routines were changed as described here. (Some of the items in the following list point out issues that are dealt with in later versions.)

- The server writes `CREATE PROCEDURE`, `CREATE FUNCTION`, `ALTER PROCEDURE`, `ALTER FUNCTION`, `DROP PROCEDURE`, and `DROP FUNCTION` statements to the binary log. Also, the server logs `CALL` statements, not the statements executed within procedures. Suppose that you issue the following statements:

  ```
  CREATE PROCEDURE mysp INSERT INTO t VALUES(1);
  CALL mysp();
  ```

  For this example, the `CREATE PROCEDURE` and `CALL` statements appear in the binary log, but the `INSERT` statement does not appear. This corrects the problem that occurred before MySQL 5.0.6 such that only the `INSERT` was logged.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Logging `CALL` statements has a security implication for replication, which arises from two factors:
  - Statements executed on a slave are processed by the slave SQL thread which has full privileges.
  - It is possible for a procedure to follow different execution paths on master and slave servers.

  The implication is that although a user must have the `CREATE ROUTINE` privilege to create a routine, the user can write a routine containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges. For example, if the master and slave servers have server ID values of 1 and 2, respectively, a user on the master server could create and invoke an unsafe procedure `unsafe_sp()` as follows:

```
mysql> delimiter //
mysql> CREATE PROCEDURE unsafe_sp ()
    -> BEGIN
    ->   IF @@server_id=2 THEN DROP DATABASE accounting; END IF;
    -> END;
    -> //
mysql> delimiter ;
mysql> CALL unsafe_sp();
```

  The `CREATE PROCEDURE` and `CALL` statements are written to the binary log, so the slave will execute them. Because the slave SQL thread has full privileges, it will execute the `DROP DATABASE` statement that drops the `accounting` database. Thus, the `CALL` statement has different effects on the master and slave and is not replication-safe.

  The preceding example uses a stored procedure, but similar problems can occur for stored functions that are invoked within statements that are written to the binary log: Function invocation has different effects on the master and slave.

  To guard against this danger for servers that have binary logging enabled, MySQL 5.0.6 introduces the requirement that stored procedure and function creators must have the `SUPER` privilege, in addition to the usual `CREATE ROUTINE` privilege that is required. Similarly, to use `ALTER PROCEDURE` or `ALTER FUNCTION`, you must have the `SUPER` privilege in addition to the `ALTER ROUTINE` privilege. Without the `SUPER` privilege, an error will occur:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_routine_creators variable)
```

  If you do not want to require routine creators to have the `SUPER` privilege (for example, if all users with the `CREATE ROUTINE` privilege on your system are experienced application developers), set the global `log_bin_trust_routine_creators` system variable to 1. You can also set this variable by using the `--log-bin-trust-routine-creators=1` option when starting the server. If binary logging is not enabled, `log_bin_trust_routine_creators` does not apply. `SUPER` is not required for routine creation unless, as described previously, the `DEFINER` value in the routine definition requires it.

- If a routine that performs updates is nondeterministic, it is not repeatable. This can have two undesirable effects:
  - It will make a slave different from the master.
  - Restored data will be different from the original data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To deal with these problems, MySQL enforces the following requirement: On a master server, creation and alteration of a routine is refused unless you declare the routine to be deterministic or to not modify data. Two sets of routine characteristics apply here:

- The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a routine always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given. To declare that a routine is deterministic, you must specify `DETERMINISTIC` explicitly.
- The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the routine reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a routine does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

By default, for a `CREATE PROCEDURE` or `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This routine has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_routine_creators
variable)
```

If you set `log_bin_trust_routine_creators` to 1, the requirement that routines be deterministic or not modify data is dropped.

- A `CALL` statement is written to the binary log if the routine returns no error, but not otherwise. When a routine that modifies data fails, you get this warning:

```
ERROR 1417 (HY000): A routine failed and has neither NO SQL nor
READS SQL DATA in its declaration and binary logging is enabled; if
non-transactional tables were updated, the binary log will miss their
changes
```

This logging behavior has the potential to cause problems. If a routine partly modifies a nontransactional table (such as a `MyISAM` table) and returns an error, the binary log will not reflect these changes. To protect against this, you should use transactional tables in the routine and modify the tables within transactions.

If you use the `IGNORE` keyword with `INSERT`, `DELETE`, or `UPDATE` to ignore errors within a routine, a partial update might occur but no error will result. Such statements are logged and they replicate normally.

- Although statements normally are not written to the binary log if they are rolled back, `CALL` statements are logged even when they occur within a rolled-back transaction. This can result in a `CALL` being rolled back on the master but executed on slaves.
- If a stored function is invoked within a statement such as `SELECT` that does not modify data, execution of the function is not written to the binary log, even if the function itself modifies data. This logging behavior has the potential to cause problems. Suppose that a function `myfunc()` is defined as follows:

```
CREATE FUNCTION myfunc () RETURNS INT DETERMINISTIC
BEGIN
  INSERT INTO t (i) VALUES(1);
  RETURN 0;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
END;
```

Given that definition, the following statement is not written to the binary log because it is a `SELECT`. Nevertheless, it modifies the table `t` because `myfunc()` modifies `t`:

```
SELECT myfunc();
```

A workaround for this problem is to invoke functions that do updates only within statements that do updates (and which therefore are written to the binary log). Note that although the `DO` statement sometimes is executed for the side effect of evaluating an expression, `DO` is not a workaround here because it is not written to the binary log.

- On slave servers, `--replicate-*-table` rules do not apply to `CALL` statements or to statements within stored routines. These statements are always replicated. If such statements contain references to tables that do not exist on the slave, they could have undesirable effects when executed on the slave.

Routine logging changes in MySQL 5.0.12: The changes in 5.0.12 address several problems that were present in earlier versions:

- Stored function invocations in nonlogged statements such as `SELECT` were not being logged, even when a function itself changed data.
- Stored procedure logging at the `CALL` level could cause different effects on a master and slave if a procedure took different execution paths on the two machines.
- `CALL` statements were logged even when they occurred within a rolled-back transaction.

To deal with these issues, MySQL 5.0.12 implements the following changes to procedure and function logging:

- A stored function invocation is logged as a `DO` statement if the function changes data and occurs within a statement that would not otherwise be logged. This corrects the problem of nonreplication of data changes that result from use of stored functions in nonlogged statements. For example, `SELECT` statements are not written to the binary log, but a `SELECT` might invoke a stored function that makes changes. To handle this, a `DO func_name()` statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the master:

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
  RETURN 0;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

When the `SELECT` statement executes, the function `f1()` is invoked three times. Two of those invocations insert a row, and MySQL logs a `DO` statement for each of them. That is, MySQL writes the following statements to the binary log:

```
DO f1(1);
DO f1(2);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The server also logs a `DO` statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the `DO` statement to the log along with the expected error code. On the slave, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

Note: See later in this section for changes made in MySQL 5.0.19: These logged `DO func_name()` statements are logged as `SELECT func_name()` statements instead.

- Stored procedure calls are logged at the statement level rather than at the `CALL` level. That is, the server does not log the `CALL` statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the master will be observed on slave servers. This eliminates the problems that could result from a procedure having different execution paths on different machines. For example, the `DROP DATABASE` problem shown earlier for the `unsafe_sp()` procedure does not occur and the routine is no longer replication-unsafe because it has the same effect on master and slave servers.

  In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because statement execution within procedures is not quite the same as in nonprocedure context:

  - A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

    ```
    NAME_CONST(var_name, var_value)
    ```

    `var_name` is the local variable name, and *`var_value`* is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` has a value of *`var_value`*, and a "name" of *`var_name`*. Thus, if you invoke this function directly, you get a result like this:

    ```
    mysql> SELECT NAME_CONST('myname', 14);
    +--------+
    | myname |
    +--------+
    |     14 |
    +--------+
    ```

    `NAME_CONST()` enables a logged standalone statement to be executed on a slave with the same effect as the original statement that was executed on the master within a stored procedure.

    The use of `NAME_CONST()` can result in a problem for `CREATE TABLE ... SELECT` statements when the source column expressions refer to local variables. Converting these references to `NAME_CONST()` expressions can result in column names that are different on the master and slave servers, or names that are too long to be legal column identifiers. A workaround is to supply aliases for columns that refer to local variables. Consider this statement when `myvar` has a value of 1:

    ```
    CREATE TABLE t1 SELECT myvar;
    ```

    That will be rewritten as follows:

    ```
    CREATE TABLE t1 SELECT NAME_CONST(myvar, 1);
    ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To ensure that the master and slave tables have the same column names, write the statement like this:

```
CREATE TABLE t1 SELECT myvar AS myvar;
```

The rewritten statement becomes:

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1) AS myvar;
```

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a `SET` statement to the binary log to make sure that the variable exists on the slave with the same value as on the master. For example, if a statement refers to a variable `@my_var`, that statement will be preceded in the binary log by the following statement, where *`value`* is the value of `@my_var` on the master:

  ```
  SET @my_var = value;
  ```

- Procedure calls can occur within a committed or rolled-back transaction. Previously, `CALL` statements were logged even if they occurred within a rolled-back transaction. As of MySQL 5.0.12, transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs `BEGIN`, `COMMIT`, and `ROLLBACK` statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, `BEGIN` and `COMMIT` statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:

  - Updates to transactional tables are not logged.
  - Updates to nontransactional tables are logged because rollback does not cancel them.
  - Updates to a mix of transactional and nontransactional tables are logged surrounded by `BEGIN` and `ROLLBACK` so that slaves will make the same changes and rollbacks as on the master.

- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a `DO` statement (if it occurs within a statement that is not logged). For this reason, care still should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

- Because procedure logging occurs at the statement level rather than at the `CALL` level, interpretation of the `--replicate-*-table` options is revised to apply only to stored functions. They no longer apply to stored procedures, except those procedures that are invoked from within functions.

Routine logging changes in MySQL 5.0.16: In 5.0.12, a change was introduced to log stored procedure calls at the statement level rather than at the `CALL` level. This change eliminates the requirement that procedures be identified as safe. The requirement now exists only for stored functions, because they still appear in the binary log as function invocations rather than as the statements executed within the function. To reflect the lifting of the restriction on stored procedures, the `log_bin_trust_routine_creators` system variable is renamed to `log_bin_trust_function_creators` and the `--log-bin-trust-routine-creators` server option is renamed to `--log-bin-trust-function-creators`. (For

This documentation is for an older version. If you're
This documentation is for an older version. If you're

backward compatibility, the old names are recognized but result in a warning.) Error messages that now apply only to functions and not to routines in general are re-worded.

Routine logging changes in MySQL 5.0.19: In 5.0.12, a change was introduced to log a stored function invocation as `DO func_name()` if the invocation changes data and occurs within a nonlogged statement, or if the function invokes a stored procedure that produces an error. In 5.0.19, these invocations are logged as `SELECT func_name()` instead. The change to `SELECT` was made because use of `DO` was found to yield insufficient control over error code checking.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 19 INFORMATION_SCHEMA Tables

Table of Contents

`INFORMATION_SCHEMA` provides access to database metadata.

Metadata is data about the data, such as the name of a database or table, the data type of a column, or access privileges. Other terms that sometimes are used for this information are *data dictionary* and *system catalog*.

`INFORMATION_SCHEMA` is the information database, the place that stores information about all the other databases that the MySQL server maintains. Inside `INFORMATION_SCHEMA` there are several read-only tables. They are actually views, not base tables, so there are no files associated with them, and you cannot set triggers on them. Also, there is no database directory with that name.

Although you can select `INFORMATION_SCHEMA` as the default database with a `USE` statement, you can only read the contents of tables, not perform `INSERT`, `UPDATE`, or `DELETE` operations on them.

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
    -> FROM information_schema.tables
    -> WHERE table_schema = 'db5'
    -> ORDER BY table_name DESC;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| v56        | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v          | VIEW       | NULL   |
| tables     | BASE TABLE | MyISAM |
| t7         | BASE TABLE | MyISAM |
| t3         | BASE TABLE | MyISAM |
| t2         | BASE TABLE | MyISAM |
| t          | BASE TABLE | MyISAM |
```


This documentation is for an older version. If you're

This documentation is for an older version. If you're

```
| pk         | BASE TABLE | InnoDB |
| loop       | BASE TABLE | MyISAM |
| kurs       | BASE TABLE | MyISAM |
| k          | BASE TABLE | MyISAM |
| into       | BASE TABLE | MyISAM |
| goto       | BASE TABLE | MyISAM |
| fk2        | BASE TABLE | InnoDB |
| fk         | BASE TABLE | InnoDB |
+------------+------------+--------+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database `db5`, in reverse alphabetic order, showing just three pieces of information: the name of the table, its type, and its storage engine.

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where *N* is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns. Values such as table names in `INFORMATION_SCHEMA` columns are treated as strings, not identifiers, and are not compared using the identifier rules described in Section 9.2.2, “Identifier Case Sensitivity”. If the result of a string operation on an `INFORMATION_SCHEMA` column differs from expectations, a workaround is to use an explicit `COLLATE` clause to force a suitable collation (Section 10.1.7.2, “Using COLLATE in SQL Statements”). You can also use the `UPPER()` or `LOWER()` function. For example, in a `WHERE` clause, you might use:

```
WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'
```

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA.ROUTINES` table), users who have insufficient privileges will see `NULL`.

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules. That is, all access is done on tables.
- Nobody needs to learn a new statement syntax. Because they already know how `SELECT` works, they only need to learn the object names.
- The implementor need not worry about adding keywords.
- There are millions of possible output variations, instead of just one. This provides more flexibility for applications that have varying requirements about what metadata they need.
- Migration is easier because every other DBMS does it this way.

However, because `SHOW` is popular and because it might be confusing were it to disappear, the advantages of conventional syntax are not a sufficient reason to eliminate `SHOW`. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as well. These are described in Section 19.18, “Extensions to SHOW Statements”.

There is no difference between the privileges required for `SHOW` statements and those required to select information from `INFORMATION_SCHEMA`. In either case, you have to have some privilege on an object in order to see information about it.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The implementation for the INFORMATION_SCHEMA table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the ENGINE column in the INFORMATION_SCHEMA.TABLES table.

Although other DBMSs use a variety of names, like syscat or system, the standard name is INFORMATION_SCHEMA.

The following sections describe each of the tables and columns that are in INFORMATION_SCHEMA. For each column, there are three pieces of information:

- "INFORMATION_SCHEMA Name" indicates the name for the column in the INFORMATION_SCHEMA table. This corresponds to the standard SQL name unless the "Remarks" field says "MySQL extension."
- "SHOW Name" indicates the equivalent field name in the closest SHOW statement, if there is one.
- "Remarks" provides additional information where applicable. If this field is NULL, it means that the value of the column is always NULL. If this field says "MySQL extension," the column is a MySQL extension to standard SQL.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked "MySQL extension". (For example, we changed COLLATION to TABLE_COLLATION in the TABLES table.) See the list of reserved words near the end of this article: http://web.archive.org/web/20070409075643rn_1/www.dbazine.com/db2/db2-disarticles/gulutzan5.

Many sections indicate what SHOW statement is equivalent to a SELECT that retrieves information from INFORMATION_SCHEMA. For SHOW statements that display information for the default database if you omit a FROM *db_name* clause, you can often select information for the default database by adding an AND TABLE_SCHEMA = DATABASE() condition to the WHERE clause of a query that retrieves information from an INFORMATION_SCHEMA table.

For answers to questions that are often asked concerning the INFORMATION_SCHEMA database, see Section A.7, "MySQL 5.0 FAQ: INFORMATION_SCHEMA".

19.1 The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPTION	Description	MySQL extension
MAXLEN	Maxlen	MySQL extension

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE CHARACTER_SET_NAME LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

19.2 The INFORMATION_SCHEMA COLLATIONS Table

The COLLATIONS table provides information about collations for each character set.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	MySQL extension
ID	Id	MySQL extension
IS_DEFAULT	Default	MySQL extension
IS_COMPILED	Compiled	MySQL extension
SORTLEN	Sortlen	MySQL extension

- COLLATION_NAME is the collation name.
- CHARACTER_SET_NAME is the name of the character set with which the collation is associated.
- ID is the collation ID.
- IS_DEFAULT indicates whether the collation is the default for its character set.
- IS_COMPILED indicates whether the character set is compiled into the server.
- SORTLEN is related to the amount of memory required to sort strings expressed in the character set.

Collation information is also available from the SHOW COLLATION statement. The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE COLLATION_NAME LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

19.3 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from SHOW COLLATION.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

19.4 The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL extension
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
PRIVILEGES	Privileges	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In SHOW, the Type display includes values from several different COLUMNS columns.
- ORDINAL_POSITION is necessary because you might want to say ORDER BY ORDINAL_POSITION. Unlike SHOW, SELECT does not have automatic ordering.
- CHARACTER_OCTET_LENGTH should be the same as CHARACTER_MAXIMUM_LENGTH, except for multibyte character sets.
- CHARACTER_SET_NAME can be derived from Collation. For example, if you say SHOW FULL COLUMNS FROM t, and you see in the Collation column a value of latin1_swedish_ci, the character set is what is before the first underscore: latin1.

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']
  [AND column_name LIKE 'wild']

SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE 'wild']
```

19.5 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. This information comes from the mysql.columns_priv grant table.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		*'user_name'@'host_name'* value
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- In the output from SHOW FULL COLUMNS, the privileges are all in one field and in lowercase, for example, select,insert,update,references. In COLUMN_PRIVILEGES, there is one privilege per row, in uppercase.
- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES.
- If the user has GRANT OPTION privilege, IS_GRANTABLE should be YES. Otherwise, IS_GRANTABLE should be NO. The output does not list GRANT OPTION as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES

SHOW GRANTS ...
```

19.6 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

Notes:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of `ORDINAL_POSITION` is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of `POSITION_IN_UNIQUE_CONSTRAINT` is `NULL` for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

  Suppose that there are two tables name `t1` and `t3` that have the following definitions:

```
CREATE TABLE t1
(
    s1 INT,
    s2 INT,
    s3 INT,
    PRIMARY KEY(s3)
) ENGINE=InnoDB;

CREATE TABLE t3
(
    s1 INT,
    s2 INT,
    s3 INT,
    KEY(s1),
    CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

  For those two tables, the `KEY_COLUMN_USAGE` table has two rows:

  - One row with `CONSTRAINT_NAME` = `'PRIMARY'`, `TABLE_NAME` = `'t1'`, `COLUMN_NAME` = `'s3'`, `ORDINAL_POSITION` = `1`, `POSITION_IN_UNIQUE_CONSTRAINT` = `NULL`.
  - One row with `CONSTRAINT_NAME` = `'CO'`, `TABLE_NAME` = `'t3'`, `COLUMN_NAME` = `'s2'`, `ORDINAL_POSITION` = `1`, `POSITION_IN_UNIQUE_CONSTRAINT` = `1`.

- `REFERENCED_TABLE_SCHEMA`, `REFERENCED_TABLE_NAME`, and `REFERENCED_COLUMN_NAME` were added in MySQL 5.0.6.

19.7 The INFORMATION_SCHEMA PROFILING Table

The `PROFILING` table provides statement profiling information. Its contents correspond to the information produced by the `SHOW PROFILES` and `SHOW PROFILE` statements (see Section 13.7.5.29, “SHOW PROFILES Syntax”). The table is empty unless the `profiling` session variable is set to 1.

`INFORMATION_SCHEMA` Name	`SHOW` Name	Remarks
`QUERY_ID`	`Query_ID`	
`SEQ`		
`STATE`	`Status`	
`DURATION`	`Duration`	
`CPU_USER`	`CPU_user`	
`CPU_SYSTEM`	`CPU_system`	
`CONTEXT_VOLUNTARY`	`Context_voluntary`	
`CONTEXT_INVOLUNTARY`	`Context_involuntary`	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INFORMATION_SCHEMA Name	SHOW Name	Remarks
BLOCK_OPS_IN	Block_ops_in	
BLOCK_OPS_OUT	Block_ops_out	
MESSAGES_SENT	Messages_sent	
MESSAGES_RECEIVED	Messages_received	
PAGE_FAULTS_MAJOR	Page_faults_major	
PAGE_FAULTS_MINOR	Page_faults_minor	
SWAPS	Swaps	
SOURCE_FUNCTION	Source_function	
SOURCE_FILE	Source_file	
SOURCE_LINE	Source_line	

Notes:

- The PROFILING table was added in MySQL 5.0.37.
- QUERY_ID is a numeric statement identifier.
- SEQ is a sequence number indicating the display order for rows with the same QUERY_ID value.
- STATE is the profiling state to which the row measurements apply.
- DURATION indicates how long statement execution remained in the given state, in seconds.
- CPU_USER and CPU_SYSTEM indicate user and system CPU use, in seconds.
- CONTEXT_VOLUNTARY and CONTEXT_INVOLUNTARY indicate how many voluntary and involuntary context switches occurred.
- BLOCK_OPS_IN and BLOCK_OPS_OUT indicate the number of block input and output operations.
- MESSAGES_SENT and MESSAGES_RECEIVED indicate the number of communication messages sent and received.
- PAGE_FAULTS_MAJOR and PAGE_FAULTS_MINOR indicate the number of major and minor page faults.
- SWAPS indicates how many swaps occurred.
- SOURCE_FUNCTION, SOURCE_FILE, and SOURCE_LINE provide information indicating where in the source code the profiled state executes.

19.8 The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (both procedures and functions). The ROUTINES table does not include user-defined functions (UDFs).

The column named "mysql.proc name" indicates the mysql.proc table column that corresponds to the INFORMATION_SCHEMA.ROUTINES table column, if any.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
SPECIFIC_NAME	specific_name	

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
ROUTINE_CATALOG		NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}
DTD_IDENTIFIER		data type descriptor
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension

Notes:

- MySQL calculates EXTERNAL_LANGUAGE thus:
  - If mysql.proc.language='SQL', EXTERNAL_LANGUAGE is NULL
  - Otherwise, EXTERNAL_LANGUAGE is what is in mysql.proc.language. However, we do not have external languages yet, so it is always NULL.

19.9 The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CATALOG_NAME		NULL
SCHEMA_NAME	Database	
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

Notes:

- DEFAULT_COLLATION_NAME was added in MySQL 5.0.6.

The following statements are equivalent:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT SCHEMA_NAME AS `Database`
  FROM INFORMATION_SCHEMA.SCHEMATA
  [WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
  [LIKE 'wild']
```

19.10 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table

The SCHEMA_PRIVILEGES table provides information about schema (database) privileges. This information comes from the mysql.db grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value, MySQL extension
TABLE_CATALOG		NULL, MySQL extension
TABLE_SCHEMA		MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the mysql.db table.

19.11 The INFORMATION_SCHEMA STATISTICS Table

The STATISTICS table provides information about table indexes.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for `sp_statistics`, except that we replaced the name `QUALIFIER` with `CATALOG` and we replaced the name `OWNER` with `SCHEMA`.

  Clearly, the preceding table and the output from `SHOW INDEX` are derived from the same parent. So the correlation is already close.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'

SHOW INDEX
  FROM tbl_name
  FROM db_name
```

19.12 The INFORMATION_SCHEMA TABLES Table

The `TABLES` table provides information about tables in databases.

`INFORMATION_SCHEMA` Name	**`SHOW` Name**	**Remarks**
`TABLE_CATALOG`		`NULL`
`TABLE_SCHEMA`	`Table_...`	
`TABLE_NAME`	`Table_...`	
`TABLE_TYPE`		
`ENGINE`	`Engine`	MySQL extension
`VERSION`	`Version`	The version number of the table's `.frm` file, MySQL extension
`ROW_FORMAT`	`Row_format`	MySQL extension
`TABLE_ROWS`	`Rows`	MySQL extension
`AVG_ROW_LENGTH`	`Avg_row_length`	MySQL extension
`DATA_LENGTH`	`Data_length`	MySQL extension
`MAX_DATA_LENGTH`	`Max_data_length`	MySQL extension
`INDEX_LENGTH`	`Index_length`	MySQL extension
`DATA_FREE`	`Data_free`	MySQL extension
`AUTO_INCREMENT`	`Auto_increment`	MySQL extension
`CREATE_TIME`	`Create_time`	MySQL extension
`UPDATE_TIME`	`Update_time`	MySQL extension
`CHECK_TIME`	`Check_time`	MySQL extension
`TABLE_COLLATION`	`Collation`	MySQL extension
`CHECKSUM`	`Checksum`	MySQL extension
`CREATE_OPTIONS`	`Create_options`	MySQL extension
`TABLE_COMMENT`	`Comment`	MySQL extension

Notes:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- TABLE_SCHEMA and TABLE_NAME are a single field in a SHOW display, for example Table_in_db1.
- TABLE_TYPE should be BASE TABLE or VIEW. The TABLES table does not list TEMPORARY tables.
- The TABLE_ROWS column is NULL if the table is in the INFORMATION_SCHEMA database.

  For InnoDB tables, the row count is only a rough estimate used in SQL optimization.
- We have nothing for the table's default character set. TABLE_COLLATION is close, because collation names begin with a character set name.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']

SHOW TABLES
  FROM db_name
  [LIKE 'wild']
```

19.13 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notes:

- The CONSTRAINT_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name field in the output from SHOW INDEX when the Non_unique field is 0.
- The CONSTRAINT_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

19.14 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. This information comes from the mysql.tables_priv grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		*'user_name'@'host_name'* value

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES

SHOW GRANTS ...
```

19.15 The INFORMATION_SCHEMA TRIGGERS Table

The TRIGGERS table provides information about triggers. You must have the SUPER privilege to access this table. You can see information only if you have the SUPER privilege).

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TRIGGER_CATALOG		NULL
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		NULL
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		0
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	
ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED		NULL (0)
SQL_MODE		MySQL extension
DEFINER		MySQL extension

Notes:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `TRIGGERS` table was added in MySQL 5.0.10.
- The names in the "`SHOW` Name" column refer to the `SHOW TRIGGERS` statement. See Section 13.7.5.35, "SHOW TRIGGERS Syntax".
- `TRIGGER_SCHEMA` and `TRIGGER_NAME`: The name of the database in which the trigger occurs and the trigger name, respectively.
- `EVENT_MANIPULATION`: The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is `'INSERT'` (a row was inserted), `'DELETE'` (a row was deleted), or `'UPDATE'` (a row was modified).
- `EVENT_OBJECT_SCHEMA` and `EVENT_OBJECT_TABLE`: As noted in Section 18.3, "Using Triggers", every trigger is associated with exactly one table. These columns indicate the database in which this table occurs, and the table name, respectively.
- `ACTION_ORDER`: The ordinal position of the trigger's action within the list of all similar triggers on the same table. This value is always `0`, because it is not possible to have more than one trigger with the same `EVENT_MANIPULATION` and `ACTION_TIMING` on the same table.
- `ACTION_STATEMENT`: The trigger body; that is, the statement executed when the trigger activates. This text uses UTF-8 encoding.
- `ACTION_ORIENTATION`: Always contains the value `'ROW'`.
- `ACTION_TIMING`: Whether the trigger activates before or after the triggering event. The value is `'BEFORE'` or `'AFTER'`.
- `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW`: The old and new column identifiers, respectively. This means that `ACTION_REFERENCE_OLD_ROW` always contains the value `'OLD'` and `ACTION_REFERENCE_NEW_ROW` always contains the value `'NEW'`.
- `SQL_MODE`: The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.7, "Server SQL Modes".
- `DEFINER`: The account of the user who created the trigger, in `'user_name'@'host_name'` format. This column was added in MySQL 5.0.17.
- The following columns currently always contain `NULL`: `TRIGGER_CATALOG`, `EVENT_OBJECT_CATALOG`, `ACTION_CONDITION`, `ACTION_REFERENCE_OLD_TABLE`, `ACTION_REFERENCE_NEW_TABLE`, and `CREATED`.

Example, using the `ins_sum` trigger defined in Section 18.3, "Using Triggers":

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS
    -> WHERE TRIGGER_SCHEMA='test' AND TRIGGER_NAME='ins_sum'\G
*************************** 1. row ***************************
           TRIGGER_CATALOG: NULL
            TRIGGER_SCHEMA: test
              TRIGGER_NAME: ins_sum
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: NULL
       EVENT_OBJECT_SCHEMA: test
        EVENT_OBJECT_TABLE: account
              ACTION_ORDER: 0
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: SET @sum = @sum + NEW.amount
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: BEFORE
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                   CREATED: NULL
                 SQL_MODE:
                  DEFINER: me@localhost
```

19.16 The INFORMATION_SCHEMA USER_PRIVILEGES Table

The `USER_PRIVILEGES` table provides information about global privileges. This information comes from the `mysql.user` grant table.

`INFORMATION_SCHEMA` Name	**`SHOW` Name**	**Remarks**
`GRANTEE`		*`'user_name'@'host_name'`* value, MySQL extension
`TABLE_CATALOG`		`NULL`, MySQL extension
`PRIVILEGE_TYPE`		MySQL extension
`IS_GRANTABLE`		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the `mysql.user` table.

19.17 The INFORMATION_SCHEMA VIEWS Table

The `VIEWS` table provides information about views in databases. You must have the `SHOW VIEW` privilege to access this table.

`INFORMATION_SCHEMA` Name	**`SHOW` Name**	**Remarks**
`TABLE_CATALOG`		`NULL`
`TABLE_SCHEMA`		
`TABLE_NAME`		
`VIEW_DEFINITION`		
`CHECK_OPTION`		
`IS_UPDATABLE`		
`DEFINER`		
`SECURITY_TYPE`		

Notes:

- The `VIEW_DEFINITION` column has most of what you see in the `Create Table` field that `SHOW CREATE VIEW` produces. Skip the words before `SELECT` and skip the words `WITH CHECK OPTION`. Suppose that the original statement was:

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Then the view definition looks like this:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- The `CHECK_OPTION` column has a value of `NONE`, `CASCADE`, or `LOCAL`.
- MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable.

  If a view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it; for details, refer to Section 18.4.3, “Updatable and Insertable Views”.)
- `DEFINER`: The account of the user who created the view, in *`'user_name'@'host_name'`* format. `SECURITY_TYPE` has a value of `DEFINER` or `INVOKER`. The `DEFINER` and `SECURITY_TYPE` columns were added in MySQL 5.0.14.

MySQL lets you use different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
    -> WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+---------------------------------+
| VIEW_DEFINITION                 |
+---------------------------------+
| select concat('a','b') AS `col1` |
+---------------------------------+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

19.18 Extensions to SHOW Statements

Some extensions to `SHOW` statements accompany the implementation of `INFORMATION_SCHEMA`:

- `SHOW` can be used to get information about the structure of `INFORMATION_SCHEMA` itself.
- `SQL_MODE`: The SQL mode in effect when the routine was created or altered, and under which the routine executes. For the permitted values, see Section 5.1.7, “Server SQL Modes”.
- Several `SHOW` statements accept a `WHERE` clause that provides more flexibility in specifying which rows to display.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

These extensions are available beginning with MySQL 5.0.3.

`INFORMATION_SCHEMA` is an information database, so its name is included in the output from `SHOW DATABASES`. Similarly, `SHOW TABLES` can be used with `INFORMATION_SCHEMA` to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| KEY_COLUMN_USAGE                      |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| STATISTICS                            |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
16 rows in set (0.00 sec)
```

`SHOW COLUMNS` and `DESCRIBE` can display information about the columns in individual `INFORMATION_SCHEMA` tables.

`SHOW` statements that accept a `LIKE` clause to limit the rows displayed also have been extended to permit a `WHERE` clause that specifies more general conditions that selected rows must satisfy:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW VARIABLES
```

The `WHERE` clause, if present, is evaluated against the column names displayed by the `SHOW` statement. For example, the `SHOW CHARACTER SET` statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
...
```

To use a `WHERE` clause with `SHOW CHARACTER SET`, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string `'japanese'`:

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+---------+---------------------------+--------------------+--------+
| Charset | Description               | Default collation  | Maxlen |
+---------+---------------------------+--------------------+--------+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci   |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci   |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci  |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+--------------------+--------+
```

This statement displays the multibyte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+---------+---------------------------+--------------------+--------+
| Charset | Description               | Default collation  | Maxlen |
+---------+---------------------------+--------------------+--------+
| big5    | Big5 Traditional Chinese  | big5_chinese_ci    |      2 |
| ujis    | EUC-JP Japanese           | ujis_japanese_ci   |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci   |      2 |
| euckr   | EUC-KR Korean             | euckr_korean_ci    |      2 |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci  |      2 |
| gbk     | GBK Simplified Chinese    | gbk_chinese_ci     |      2 |
| utf8    | UTF-8 Unicode             | utf8_general_ci    |      3 |
| ucs2    | UCS-2 Unicode             | ucs2_general_ci    |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci  |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+--------------------+--------+
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 20 Connectors and APIs

Table of Contents

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), Perl, Python, PHP, Ruby, and native C and embedded MySQL instances.

Note

Connector version numbers do not correlate with MySQL Server version numbers. See Table 20.2, “MySQL Connector Versions and MySQL Server Versions”.

MySQL Connectors

Oracle develops a number of connectors:

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and OS X platforms.
- Connector/Net enables developers to create .NET applications that connect to MySQL. Connector/Net implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/Net can be written in any supported .NET language.

  The MySQL Visual Studio Plugin works with Connector/Net and Visual Studio 2005. The plugin is a MySQL DDEX Provider, which means that you can use the schema and data manipulation tools available in Visual Studio to create and edit objects within a MySQL database.
- Connector/J provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/C is a standalone replacement for the MySQL Client Library (`libmysqlclient`), to be used for C applications.

The MySQL C API

For direct access to using MySQL natively within a C application, there are two methods:

- The C API provides low-level access to the MySQL client/server protocol through the `libmysqlclient` client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

  `libmysqlclient` is included in MySQL distributions and in Connector/C distributions.
- `libmysqld` is an embedded MySQL server library that enables you to embed an instance of the MySQL server into your C applications.

  `libmysqld` is included in MySQL distributions, but not in Connector/C distributions.

See also Section 20.6.1, “MySQL C API Implementations”.

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where to start. A number of programmer's utilities are available to help with the process; see Section 4.7, “MySQL Program Development Utilities”.

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using `libmysqlclient` or by implementing a *native driver*. The two solutions offer different benefits:

- Using *`libmysqlclient`* offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through `libmysqlclient` and the performance may be lower as data is copied between the native language, and the MySQL API components.
- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they

This documentation is for an older version. If you're
This documentation is for an older version. If you're

can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

Table 20.1, “MySQL APIs and Interfaces” lists many of the libraries and interfaces available for MySQL. Table 20.2, “MySQL Connector Versions and MySQL Server Versions” shows which MySQL Server versions each connector supports.

Table 20.1 MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	`libmysqlclient`	See MySQL Bindings for GNU Ada
C	C API	`libmysqlclient`	See Section 20.6, “MySQL C API”.
C	Connector/C	Replacement for `libmysqlclient`	See MySQL Connector/C Developer Guide.
C++	Connector/C++	`libmysqlclient`	See MySQL Connector/C++ Developer Guide.
	MySQL++	`libmysqlclient`	See MySQL++ Web site.
	MySQL wrapped	`libmysqlclient`	See MySQL wrapped.
Cocoa	MySQL-Cocoa	`libmysqlclient`	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	`libmysqlclient`	See MySQL for D.
Eiffel	Eiffel MySQL	`libmysqlclient`	See Section 20.12, “MySQL Eiffel Wrapper”.
Erlang	`erlang-mysql-driver`	`libmysqlclient`	See `erlang-mysql-driver`.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	`hsql-mysql`	`libmysqlclient`	See MySQL driver for Haskell .
Java/ JDBC	Connector/J	Native Driver	See MySQL Connector/J 5.1 Developer Guide.
Kaya	MyDB	`libmysqlclient`	See MyDB.
Lua	LuaSQL	`libmysqlclient`	See LuaSQL.
.NET/ Mono	Connector/Net	Native Driver	See MySQL Connector/Net Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	`libmysqlclient`	See MySQL Bindings for Objective Caml.
Octave	Database bindings for GNU Octave	`libmysqlclient`	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	`libmysqlclient`	See MySQL Connector/ODBC Developer Guide.
Perl	`DBI`/`DBD::mysql`	`libmysqlclient`	See Section 20.8, “MySQL Perl API”.
	`Net::MySQL`	Native Driver	See `Net::MySQL` at CPAN
PHP	`mysql`, `ext/mysql` interface (deprecated)	`libmysqlclient`	See Original MySQL API.
	`mysqli`, `ext/mysqli` interface	`libmysqlclient`	See MySQL Improved Extension.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Environment	API	Type	Notes
	PDO_MYSQL	libmysqlclient	See MySQL Functions (PDO_MYSQL).
	PDO mysqlnd	Native Driver	
Python	Connector/Python	Native Driver	See MySQL Connector/Python Developer Guide.
Python	Connector/Python C Extension	libmysqlclient	See MySQL Connector/Python Developer Guide.
	MySQLdb	libmysqlclient	See Section 20.9, “MySQL Python API”.
Ruby	MySQL/Ruby	libmysqlclient	Uses libmysqlclient. See Section 20.10.1, “The MySQL/Ruby API”.
	Ruby/MySQL	Native Driver	See Section 20.10.2, “The Ruby/MySQL API”.
Scheme	Myscsh	libmysqlclient	See Myscsh.
SPL	sql_mysql	libmysqlclient	See sql_mysql for SPL.
Tcl	MySQLtcl	libmysqlclient	See Section 20.11, “MySQL Tcl API”.

Table 20.2 MySQL Connector Versions and MySQL Server Versions

Connector	Connector version	MySQL Server version
Connector/C	6.1.0 GA	5.6, 5.5, 5.1, 5.0, 4.1
Connector/C++	1.0.5 GA	5.6, 5.5, 5.1
Connector/J	5.1.8	5.6, 5.5, 5.1, 5.0, 4.1
Connector/Net	6.5	5.6, 5.5, 5.1, 5.0
Connector/Net	6.4	5.6, 5.5, 5.1, 5.0
Connector/Net	6.3	5.6, 5.5, 5.1, 5.0
Connector/Net	6.2 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	6.1 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	6.0 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	5.2 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	1.0 (No longer supported)	5.0, 4.0
Connector/ODBC	5.1	5.6, 5.5, 5.1, 5.0, 4.1.1+
Connector/ODBC	3.51 (Unicode not supported)	5.6, 5.5, 5.1, 5.0, 4.1
Connector/Python	2.0	5.7, 5.6, 5.5
Connector/Python	1.2	5.7, 5.6, 5.5

20.1 MySQL Connector/ODBC

The MySQL Connector/ODBC manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/ODBC Developer Guide
- Release notes: MySQL Connector/ODBC Release Notes

This documentation is for an older version. If you're
This documentation is for an older version. If you're

20.2 MySQL Connector/Net

The MySQL Connector/Net manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Net Developer Guide
- Release notes: MySQL Connector/Net Release Notes

20.3 MySQL Connector/J

The MySQL Connector/J manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/J 5.1 Developer Guide
- Release notes: MySQL Connector/J Release Notes

20.4 MySQL Connector/C

The MySQL Connector/C manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C Developer Guide
- Release notes: MySQL Connector/C Release Notes

20.5 libmysqld, the Embedded MySQL Server Library

The embedded MySQL server library is NOT part of MySQL 5.0. It is part of previous editions and will be included in future versions, starting with MySQL 5.1. You can find appropriate documentation in the corresponding manuals for these versions. In this manual, only an overview of the embedded library is provided.

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change an old threaded application to use the embedded library, you normally only have to add calls to the following functions.

Table 20.3 MySQL Embedded Server Library Functions

Function	When to Call
`mysql_library_init()`	Call it before any other MySQL function is called, preferably early in the `main()` function.
`mysql_library_end()`	Call it before your program exits.
`mysql_thread_init()`	Call it in each thread you create that accesses MySQL.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Function	When to Call
`mysql_thread_end()`	Call it before calling `pthread_exit()`.

Then, link your code with `libmysqld.a` instead of `libmysqlclient.a`. To ensure binary compatibility between your application and the server library, be sure to compile your application against headers for the same series of MySQL that was used to compile the server library. For example, if `libmysqld` was compiled against MySQL 4.1 headers, do not compile your application against MySQL 5.0 headers,

The `mysql_library_xxx()` functions are also included in `libmysqlclient.a` to enable you to change between the embedded and the client/server version by just linking your application with the right library. See Section 20.6.7.40, “mysql_library_init()”.

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default. To use authentication for the embedded server, specify the `--with-embedded-privilege-control` option when you invoke `configure` to configure your MySQL distribution.

20.6 MySQL C API

The C API provides low-level access to the MySQL client/server protocol and enables C programs to access database contents. The C API code is distributed with MySQL and implemented in the `libmysqlclient` library. See Section 20.6.1, “MySQL C API Implementations”.

Most other client APIs use the `libmysqlclient` library to communicate with the MySQL server. (Exceptions are except Connector/J and Connector/Net.) This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs because they are referenced from the library. For a list of these variables, see Section 4.1, “Overview of MySQL Programs”.

For instructions on building client programs using the C API, see Section 20.6.4.1, “Building C API Client Programs”. For programming with threads, see Section 20.6.4.2, “Writing C API Threaded Client Programs”. To create a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see Section 20.5, “libmysqld, the Embedded MySQL Server Library”.

Note

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the `mysql.h` file and `libmysqlclient.a` library used for compilation to verify that they are from the new MySQL distribution. If not, recompile the programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.17` to `libmysqlclient.so.18`). For additional compatibility information, see Section 20.6.4.3, “Running C API Client Programs”.

Clients have a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (16MB by default). Because buffer sizes are increased only as demand warrants, simply increasing the maximum limit does not in itself cause more resources to be used. This size check is mostly a precaution against erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each session's communication buffer is

This documentation is for an older version. If you're
This documentation is for an older version. If you're

dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The default maximum built into the client library is 1GB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` parameter at server startup. See Section 8.12.2, “Tuning Server Parameters”.

The MySQL server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

20.6.1 MySQL C API Implementations

The MySQL C API is a C-based API that client applications written in C can use to communicate with MySQL Server. Client programs refer to C API header files at compile time and link to a C API library file at link time. The library comes in two versions, depending on how the application is intended to communicate with the server:

- `libmysqlclient`: The client version of the library, used for applications that communicate over a network connection as a client of a standalone server process.
- `libmysqld`: The embedded server version of the library, used for applications intended to include an embedded MySQL server within the application itself. The application communicates with its own private server instance.

Both libraries have the same interface. In terms of C API calls, an application communicates with a standalone server the same way it communicates with an embedded server. A given client can be built to communicate with a standalone or embedded server, depending on whether it is linked against `libmysqlclient` or `libmysqld` at build time.

There are two ways to obtain the C API header and library files required to build C API client programs:

- Install a MySQL Server distribution. Server distributions include both `libmysqlclient` and `libmysqld`.
- Install a Connector/C distribution. Connector/C distributions include only `libmysqlclient`. They do not include `libmysqld`.

For both MySQL Server and Connector/C, you can install a binary distribution that contains the C API files pre-built, or you can use a source distribution and build the C API files yourself.

Normally, you install either a MySQL Server distribution or a Connector/C distribution, but not both. For information about issues involved with simultaneous MySQL Server and Connector/C installations, see Section 20.6.2, “Simultaneous MySQL Server and Connector/C Installations”.

The names of the library files to use when linking C API client applications depend on the library type and platform for which a distribution is built:

- On Unix (and Unix-like) sytems, the static library is `libmysqlclient.a`. The dynamic library is `libmysqlclient.so` on most Unix systems and `libmysqlclient.dylib` on OS X.

  For distributions that include embedded server libraries, the corresponding library names begin with `libmysqld` rather than `libmysqlclient`.

- On Windows, the static library is `mysqlclient.lib` and the dynamic library is `libmysql.dll`. Windows distributions also include `libmysql.lib`, a static import library needed for using the dynamic library.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For distributions that include embedded server libraries, the corresponding library names are `mysqlserver.lib`, `libmysqld.dll`, and `libmysqld.lib`.

Windows distributions also include a set of debug libraries. These have the same names as the nondebug libraries, but are located in the `lib/debug` library. You must use the debug libraries when compiling clients built using the debug C runtime.

20.6.2 Simultaneous MySQL Server and Connector/C Installations

MySQL Server and Connector/C installation packages both provide the files needed to build and run MySQL C API client programs. This section discusses when it is possible to install both products on the same system. For some packaging formats, this is possible without conflict. For others, both products cannot be installed at the same time.

This discussion assumes the use of similar package types for both products (for example, RPM packages for both products). It does not try to describe coexistence between packaging types (for example, use of RPM packages for one product and a `tar` file package for the other). Nor does it describe coexistence of packages provided by Oracle and those provided by third-party vendors.

If you install both products, it may be necessary to adjust your development tools or runtime environment to choose one set of header files and libraries over the other. See Section 20.6.4.1, “Building C API Client Programs”, and Section 20.6.4.3, “Running C API Client Programs”.

`tar` and Zip file packages install under the directory into which you unpack them. For example, you can unpack MySQL Server and Connector/C `tar` packages under `/usr/local` and they will unpack into distinct directory names without conflict.

Windows MSI installers use their own installation directory, so MySQL Server and Connector/C installers do not conflict.

OS X DMG packages install under the same parent directory but in a different subdirectory, so there is no conflict. For example:

```
/usr/local/mysql-5.6.11-osx10.7-x86_64/
/usr/local/mysql-connector-c-6.1.0-osx10.7-x86/
```

Solaris PKG packages install under the same parent directory but in a different subdirectory, so there is no conflict. For example:

```
/opt/mysql/mysql
/opt/mysql/connector-c
```

The Solaris Connector/C installer does not create any symlinks from system directories such as `/usr/bin` or `/usr/lib` into the installation directory. That must be done manually if desired after installation.

For RPM installations, there are several types of RPM packages. MySQL Server `shared` and `devel` RPM packages are similar to the corresponding Connector/C RPM packages. These RPM package types cannot coexist because the MySQL Server and Connector/C RPM packages use the same installation locations for the client library-related files. This means the following conditions hold:

- If MySQL Server `shared` and `devel` RPM packages are installed, they provide the C API headers and libraries, and there is no need to install the Connector/C RPM packages. To install the Connector/C packages anyway, you must first remove the corresponding MySQL Server packages.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- To install MySQL Server RPM packages if you already have Connector/C RPM packages installed, you must first remove the Connector/C RPM packages.

MySQL Server RPM packages other than `shared` and `devel` do not conflict with Connector/C packages and can be installed if Connector/C is installed. This includes the main server RPM that includes the `mysqld` server itself.

20.6.3 Example C API Client Programs

Many of the clients in MySQL source distributions are written in C, such as `mysql`, `mysqladmin`, and `mysqlshow`. If you are looking for examples that demonstrate how to use the C API, take a look at these clients: Obtain a source distribution and look in its `client` directory. See Section 2.5, “How to Get MySQL”.

20.6.4 Building and Running C API Client Programs

The following sections provide information on building client programs that use the C API. Topics include compiling and linking clients, writing threaded clients, and troubleshooting runtime problems.

20.6.4.1 Building C API Client Programs

This section provides guidelines for compiling C programs that use the MySQL C API.

Compiling MySQL Clients on Unix

You may need to specify an `-I` option when you compile client programs that use MySQL header files, so that the compiler can find them. For example, if the header files are installed in `/usr/local/mysql/include`, use this option in the compile command:

```
-I/usr/local/mysql/include
```

MySQL clients must be linked using the `-lmysqlclient -lz` options in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use these options in the link command:

```
-L/usr/local/mysql/lib -lmysqlclient -lz
```

The path names may differ on your system. Adjust the `-I` and `-L` options as necessary.

To make it simpler to compile MySQL programs on Unix, use the `mysql_config` script. See Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”.

`mysql_config` displays the options needed for compiling or linking:

```
shell> mysql_config --cflags
shell> mysql_config --libs
```

You can run those commands to get the proper options and add them manually to compilation or link commands. Alternatively, include the output from `mysql_config` directly within command lines using backticks:

```
shell> gcc -c `mysql_config --cflags` progname.c
shell> gcc -o progname progname.o `mysql_config --libs`
```

Compiling MySQL Clients on Microsoft Windows

This documentation is for an older version. If you're
This documentation is for an older version. If you're

To specify header and library file locations, use the facilities provided by your development environment.

On Windows, you can link your code with either the dynamic or static C client library. The static library is named `mysqlclient.lib` and the dynamic library is named `libmysql.dll`. In addition, the `libmysql.lib` static import library is needed for using the dynamic library.

If you link with the static library, failure can occur unless these conditions are satisfied:

- The client application must be compiled with the same version of Visual Studio used to compile the library.
- The client application should link the C runtime statically by using the `/MT` compiler option.

If the client application is built in in debug mode and uses the static debug C runtime (`/MTd` compiler option), it can link to the `mysqlclient.lib` static library if that library was built using the same option. If the client application uses the dynamic C runtime (`/MD` option, or `/MDd` option in debug mode), it must must be linked to the `libmysql.dll` dynamic library. It cannot link to the static client library.

The MSDN page describing the link options can be found here: http://msdn.microsoft.com/en-us/library/2kzt1wy3.aspx

Troubleshooting Problems Linking to the MySQL Client Library

Linking with the single-threaded library (`libmysqlclient`) may lead to linker errors related to `pthread` symbols. When using the single-threaded library, please compile your client with `MYSQL_CLIENT_NO_THREADS` defined. This can be done on the command line by using the `-D` option to the compiler, or in your source code before including the MySQL header files. This define should not be used when building for use with the thread-safe client library (`libmysqlclient_r`).

If the linker cannot find the MySQL client library, you might get undefined-reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-L`*`dir_path`* `-lmysqlclient` at the end of your link command, where *`dir_path`* represents the path name of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well. You can include `mysql_config` output directly in your compile or link command using backticks. For example:

```
shell> gcc -o progname progname.o `mysql_config --libs`
```

If an error occurs at link time that the `floor` symbol is undefined, link to the math library by adding `-lm` to the end of the compile/link line.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

This documentation is for an older version. If you're

Similarly, if you get undefined-reference errors for other functions that should exist on your system, such as `connect()`, check the manual page for the function in question to determine which libraries you should add to the link command.

If you get undefined-reference errors such as the following for functions that do not exist on your system, it usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

In this case, you should download the latest MySQL or Connector/C source distribution and compile the MySQL client library yourself. See Section 2.17, “Installing MySQL from Source”, and MySQL Connector/C Developer Guide.

20.6.4.2 Writing C API Threaded Client Programs

The client library is almost thread-safe. The biggest problem is that the subroutines in `sql/net_serv.cc` that read from sockets are not interrupt-safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, socket handling should be thread-safe.

To avoid aborting the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_library_init()`, `mysql_init()`, or `mysql_connect()`. To use your own `SIGPIPE` handler, first call `mysql_library_init()`, then install your handler.

Current binary distributions should have both a normal client library, `libmysqlclient`, and a thread-safe library, `libmysqlclient_r`. For threaded clients, link against the latter library. If “undefined symbol” errors occur, in most cases this is because you have not included the thread libraries on the link/compile command.

The thread-safe client library, `libmysqlclient_r`, is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Multiple threads cannot send a query to the MySQL server at the same time on the same connection. In particular, you must ensure that between calls to `mysql_query()` and `mysql_store_result()` in one thread, no other thread uses the same connection. You must have a mutex lock around your pair of `mysql_query()` and `mysql_store_result()` calls. After `mysql_store_result()` returns, the lock can be released and other threads may query the same connection.

  If you use POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

- Many threads can access different result sets that are retrieved with `mysql_store_result()`.

- To use `mysql_use_result()`, you must ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.

You need to know the following if you have a thread that did not create the connection to the MySQL database but is calling MySQL functions:

When you call `mysql_init()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things). If you call a MySQL function before the thread has called `mysql_init()`, the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later. To avoid problems, you must do the following:

1. Call `mysql_library_init()` before any other MySQL functions. It is not thread-safe, so call it before threads are created, or protect the call with a mutex.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. Arrange for `mysql_thread_init()` to be called early in the thread handler before calling any MySQL function. If you call `mysql_init()`, it will call `mysql_thread_init()` for you.

3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

The preceding notes regarding `mysql_init()` also apply to `mysql_connect()`, which calls `mysql_init()`.

20.6.4.3 Running C API Client Programs

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the `mysql.h` file and `libmysqlclient.a` library used for compilation to verify that they are from the new MySQL distribution. If not, recompile the programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.17` to `libmysqlclient.so.18`).

The major client library version determines compatibility. (For example, for `libmysqlclient.so.18.1.0`, the major version is 18.) For this reason, the libraries shipped with newer versions of MySQL are drop-in replacements for older versions that have the same major number. As long as the major library version is the same, you can upgrade the library and old applications should continue to work with it.

Undefined-reference errors might occur at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `libmysqlclient` library cannot be found, it means that your system cannot find the shared `libmysqlclient.so` library. The solution to this problem is to tell your system to search for shared libraries in the directory where that library is located. Use whichever of the following methods is appropriate for your system:

- Add the path of the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` or `LD_LIBRARY` environment variable.
- On OS X, add the path of the directory where `libmysqlclient.dylib` is located to the `DYLD_LIBRARY_PATH` environment variable.
- Copy the shared-library files (such as `libmysqlclient.so`) to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`. Be sure to copy all related files. A shared library might exist under several names, using symlinks to provide the alternate names.

Another way to solve this problem is by linking your program statically with the `-static` option, or by removing the dynamic MySQL libraries before linking your code. Before trying the second method, you should be sure that no other programs are using the dynamic libraries.

20.6.4.4 C API Server and Client Library Versions

The string and numeric forms of the MySQL server version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_server_info()` and `mysql_get_server_version()` functions.

The client library version is the MySQL version. For Connector/C, this is the MySQL version on which the Connector/C distribution is based. The string and numeric forms of this version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_client_info()` and `mysql_get_client_version()` functions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

20.6.5 C API Data Structures

This section describes C API data structures other than those used for prepared statements. For information about the latter, see Section 20.6.9, “C API Prepared Statement Data Structures”.

- `MYSQL`

  This structure represents a handle to one database connection. It is used for almost all MySQL functions. Do not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy is usable.

- `MYSQL_RES`

  This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

  This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

  This structure contains metadata: information about a field, such as the field's name, type, and size. Its members are described in more detail later in this section. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

  This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

- `my_ulonglong`

  The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()`, and `mysql_insert_id()`. This type provides a range of `0` to `1.84e19`.

  Some functions that return a row count using this type return -1 as an unsigned value to indicate an error or exceptional condition. You can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

  On some systems, attempting to print a value of type `my_ulonglong` does not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

  ```
  printf ("Number of rows: %lu\n",
          (unsigned long) mysql_num_rows(result));
  ```

- `my_bool`

  A boolean type, for values that are true (nonzero) or false (zero).

The `MYSQL_FIELD` structure contains the members described in the following list:

- `char * name`

This documentation is for an older version. If you're

The name of the field, as a null-terminated string. If the field was given an alias with an `AS` clause, the value of `name` is the alias.

- `char * org_name`

  The name of the field, as a null-terminated string. Aliases are ignored. For expressions, the value is an empty string.

- `char * table`

  The name of the table containing this field, if it is not a calculated field. For calculated fields, the `table` value is an empty string. If the column is selected from a view, `table` names the view. If the table or view was given an alias with an `AS` clause, the value of `table` is the alias. For a `UNION`, the value is the empty string.

- `char * org_table`

  The name of the table, as a null-terminated string. Aliases are ignored. If the column is selected from a view, `org_table` names the view. For a `UNION`, the value is the empty string.

- `char * db`

  The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, `db` is an empty string. For a `UNION`, the value is the empty string.

- `char * catalog`

  The catalog name. This value is always `"def"`.

- `char * def`

  The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `unsigned long length`

  The width of the field. This corresponds to the display length, in bytes.

  The server determines the `length` value before it generates the result set, so this is the minimum length required for a data type capable of holding the largest possible value from the result column, without knowing in advance the actual values that will be produced by the query for the result set.

- `unsigned long max_length`

  The maximum width of the field for the result set (the length in bytes of the longest field value for the rows actually in the result set). If you use `mysql_store_result()` or `mysql_list_fields()`, this contains the maximum length for the field. If you use `mysql_use_result()`, the value of this variable is zero.

  The value of `max_length` is the length of the string representation of the values in the result set. For example, if you retrieve a `FLOAT` column and the “widest” value is `-12.345`, `max_length` is 7 (the length of `'-12.345'`).

  If you are using prepared statements, `max_length` is not set by default because for the binary protocol the lengths of the values depend on the types of the values in the result set. (See Section 20.6.9, “C API Prepared Statement Data Structures”.) If you want the `max_length` values anyway, enable the `STMT_ATTR_UPDATE_MAX_LENGTH` option with `mysql_stmt_attr_set()` and the lengths will be set

This documentation is for an older version. If you're
This documentation is for an older version. If you're

when you call `mysql_stmt_store_result()`. (See Section 20.6.11.3, “mysql_stmt_attr_set()”, and Section 20.6.11.27, “mysql_stmt_store_result()”.)

- `unsigned int name_length`

  The length of `name`.

- `unsigned int org_name_length`

  The length of `org_name`.

- `unsigned int table_length`

  The length of `table`.

- `unsigned int org_table_length`

  The length of `org_table`.

- `unsigned int db_length`

  The length of `db`.

- `unsigned int catalog_length`

  The length of `catalog`.

- `unsigned int def_length`

  The length of `def`.

- `unsigned int flags`

  Bit-flags that describe the field. The `flags` value may have zero or more of the bits set that are shown in the following table.

Flag Value	Flag Description
`NOT_NULL_FLAG`	Field cannot be `NULL`
`PRI_KEY_FLAG`	Field is part of a primary key
`UNIQUE_KEY_FLAG`	Field is part of a unique key
`MULTIPLE_KEY_FLAG`	Field is part of a nonunique key
`UNSIGNED_FLAG`	Field has the `UNSIGNED` attribute
`ZEROFILL_FLAG`	Field has the `ZEROFILL` attribute
`BINARY_FLAG`	Field has the `BINARY` attribute
`AUTO_INCREMENT_FLAG`	Field has the `AUTO_INCREMENT` attribute
`ENUM_FLAG`	Field is an `ENUM`
`SET_FLAG`	Field is a `SET`
`BLOB_FLAG`	Field is a `BLOB` or `TEXT` (deprecated)
`TIMESTAMP_FLAG`	Field is a `TIMESTAMP` (deprecated)
`NUM_FLAG`	Field is numeric; see additional notes following table
`NO_DEFAULT_VALUE_FLAG`	Field has no default value; see additional notes following table

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Some of these flags indicate data type information and are superseded by or used in conjunction with the `MYSQL_TYPE_xxx` value in the `field->type` member described later:

- To check for `BLOB` or `TIMESTAMP` values, check whether `type` is `MYSQL_TYPE_BLOB` or `MYSQL_TYPE_TIMESTAMP`. (The `BLOB_FLAG` and `TIMESTAMP_FLAG` flags are unneeded.)
- `ENUM` and `SET` values are returned as strings. For these, check that the `type` value is `MYSQL_TYPE_STRING` and that the `ENUM_FLAG` or `SET_FLAG` flag is set in the `flags` value.

`NUM_FLAG` indicates that a column is numeric. This includes columns with a type of `MYSQL_TYPE_DECIMAL`, `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, `MYSQL_TYPE_FLOAT`, `MYSQL_TYPE_DOUBLE`, `MYSQL_TYPE_NULL`, `MYSQL_TYPE_LONGLONG`, `MYSQL_TYPE_INT24`, and `MYSQL_TYPE_YEAR`.

`NO_DEFAULT_VALUE_FLAG` indicates that a column has no `DEFAULT` clause in its definition. This does not apply to `NULL` columns (because such columns have a default of `NULL`), or to `AUTO_INCREMENT` columns (which have an implied default value). `NO_DEFAULT_VALUE_FLAG` was added in MySQL 5.0.2.

The following example illustrates a typical use of the `flags` value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field cannot be null\n");
```

You may use the convenience macros shown in the following table to determine the boolean status of the `flags` value.

Flag Status	Description
`IS_NOT_NULL(flags)`	True if this field is defined as `NOT NULL`
`IS_PRI_KEY(flags)`	True if this field is a primary key
`IS_BLOB(flags)`	True if this field is a `BLOB` or `TEXT` (deprecated; test `field->type` instead)

- `unsigned int decimals`

  The number of decimals for numeric fields.

- `unsigned int charsetnr`

  An ID number that indicates the character set/collation pair for the field.

  Normally, character values in result sets are converted to the character set indicated by the `character_set_results` system variable. In this case, `charsetnr` corresponds to the character set indicated by that variable. Character set conversion can be suppressed by setting `character_set_results` to `NULL`. In this case, `charsetnr` corresponds to the character set of the original table column or expression. See also Section 10.1.4, “Connection Character Sets and Collations”.

  To distinguish between binary and nonbinary data for string data types, check whether the `charsetnr` value is 63. If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`charsetnr` values are the same as those displayed in the `Id` column of the `SHOW COLLATION` statement or the `ID` column of the `INFORMATION_SCHEMA COLLATIONS` table. You can use those information sources to see which character set and collation specific `charsetnr` values indicate:

```
mysql> SHOW COLLATION WHERE Id = 63;
+----------+---------+----+---------+----------+---------+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+----------+---------+----+---------+----------+---------+
| binary    | binary  | 63 | Yes     | Yes      |       1 |
+----------+---------+----+---------+----------+---------+

mysql> SELECT COLLATION_NAME, CHARACTER_SET_NAME
    -> FROM INFORMATION_SCHEMA.COLLATIONS WHERE ID = 33;
+-----------------+--------------------+
| COLLATION_NAME  | CHARACTER_SET_NAME |
+-----------------+--------------------+
| utf8_general_ci | utf8               |
+-----------------+--------------------+
```

- `enum enum_field_types type`

  The type of the field. The `type` value may be one of the `MYSQL_TYPE_` symbols shown in the following table.

Type Value	Type Description
`MYSQL_TYPE_TINY`	`TINYINT` field
`MYSQL_TYPE_SHORT`	`SMALLINT` field
`MYSQL_TYPE_LONG`	`INTEGER` field
`MYSQL_TYPE_INT24`	`MEDIUMINT` field
`MYSQL_TYPE_LONGLONG`	`BIGINT` field
`MYSQL_TYPE_DECIMAL`	`DECIMAL` or `NUMERIC` field
`MYSQL_TYPE_NEWDECIMAL`	Precision math `DECIMAL` or `NUMERIC` field (MySQL 5.0.3 and up)
`MYSQL_TYPE_FLOAT`	`FLOAT` field
`MYSQL_TYPE_DOUBLE`	`DOUBLE` or `REAL` field
`MYSQL_TYPE_BIT`	`BIT` field (MySQL 5.0.3 and up)
`MYSQL_TYPE_TIMESTAMP`	`TIMESTAMP` field
`MYSQL_TYPE_DATE`	`DATE` field
`MYSQL_TYPE_TIME`	`TIME` field
`MYSQL_TYPE_DATETIME`	`DATETIME` field
`MYSQL_TYPE_YEAR`	`YEAR` field
`MYSQL_TYPE_STRING`	`CHAR` or `BINARY` field
`MYSQL_TYPE_VAR_STRING`	`VARCHAR` or `VARBINARY` field
`MYSQL_TYPE_BLOB`	`BLOB` or `TEXT` field (use `max_length` to determine the maximum length)
`MYSQL_TYPE_SET`	`SET` field
`MYSQL_TYPE_ENUM`	`ENUM` field
`MYSQL_TYPE_GEOMETRY`	Spatial field

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Type Value	Type Description
MYSQL_TYPE_NULL	NULL-type field

You can use the IS_NUM() macro to test whether a field has a numeric type. Pass the type value to IS_NUM() and it evaluates to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

ENUM and SET values are returned as strings. For these, check that the type value is MYSQL_TYPE_STRING and that the ENUM_FLAG or SET_FLAG flag is set in the flags value.

20.6.6 C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See Section 20.6.7, “C API Function Descriptions”.

Table 20.4 C API Function Names and Descriptions

Function	Description
my_init()	Initialize global variables, and thread handler in thread-safe programs
mysql_affected_rows()	Returns the number of rows changed/deleted/inserted by the last UPDATE, DELETE, or INSERT query
mysql_autocommit()	Toggles autocommit mode on/off
mysql_change_user()	Changes user and database on an open connection
mysql_character_set_name()	Return default character set name for current connection
mysql_close()	Closes a server connection
mysql_commit()	Commits the transaction
mysql_connect()	Connects to a MySQL server (this function is deprecated; use mysql_real_connect() instead)
mysql_create_db()	Creates a database (this function is deprecated; use the SQL statement CREATE DATABASE instead)
mysql_data_seek()	Seeks to an arbitrary row number in a query result set
mysql_debug()	Does a DBUG_PUSH with the given string
mysql_drop_db()	Drops a database (this function is deprecated; use the SQL statement DROP DATABASE instead)
mysql_dump_debug_info()	Makes the server write debug information to the log
mysql_eof()	Determines whether the last row of a result set has been read (this function is deprecated; mysql_errno() or mysql_error() may be used instead)
mysql_errno()	Returns the error number for the most recently invoked MySQL function
mysql_error()	Returns the error message for the most recently invoked MySQL function
mysql_escape_string()	Escapes special characters in a string for use in an SQL statement
mysql_fetch_field()	Returns the type of the next table field
mysql_fetch_field_direct()	Returns the type of a table field, given a field number

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Function	Description
mysql_fetch_fields()	Returns an array of all field structures
mysql_fetch_lengths()	Returns the lengths of all columns in the current row
mysql_fetch_row()	Fetches the next row from the result set
mysql_field_count()	Returns the number of result columns for the most recent statement
mysql_field_seek()	Puts the column cursor on a specified column
mysql_field_tell()	Returns the position of the field cursor used for the last mysql_fetch_field()
mysql_free_result()	Frees memory used by a result set
mysql_get_character_set_info()	Return information about default character set
mysql_get_client_info()	Returns client version information as a string
mysql_get_client_version()	Returns client version information as an integer
mysql_get_host_info()	Returns a string describing the connection
mysql_get_proto_info()	Returns the protocol version used by the connection
mysql_get_server_info()	Returns the server version number
mysql_get_server_version()	Returns version number of server as an integer
mysql_get_ssl_cipher()	Return current SSL cipher
mysql_hex_string()	Encode string in hexadecimal format
mysql_info()	Returns information about the most recently executed query
mysql_init()	Gets or initializes a MYSQL structure
mysql_insert_id()	Returns the ID generated for an AUTO_INCREMENT column by the previous query
mysql_kill()	Kills a given thread
mysql_library_end()	Finalize the MySQL C API library
mysql_library_init()	Initialize the MySQL C API library
mysql_list_dbs()	Returns database names matching a simple regular expression
mysql_list_fields()	Returns field names matching a simple regular expression
mysql_list_processes()	Returns a list of the current server threads
mysql_list_tables()	Returns table names matching a simple regular expression
mysql_more_results()	Checks whether any more results exist
mysql_next_result()	Returns/initiates the next result in multiple-result executions
mysql_num_fields()	Returns the number of columns in a result set
mysql_num_rows()	Returns the number of rows in a result set
mysql_options()	Sets connect options for mysql_real_connect()
mysql_ping()	Checks whether the connection to the server is working, reconnecting as necessary
mysql_query()	Executes an SQL query specified as a null-terminated string
mysql_real_connect()	Connects to a MySQL server
mysql_real_escape_string()	Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Function	Description
mysql_real_query()	Executes an SQL query specified as a counted string
mysql_refresh()	Flush or reset tables and caches
mysql_reload()	Tells the server to reload the grant tables
mysql_rollback()	Rolls back the transaction
mysql_row_seek()	Seeks to a row offset in a result set, using value returned from mysql_row_tell()
mysql_row_tell()	Returns the row cursor position
mysql_select_db()	Selects a database
mysql_server_end()	Finalize the MySQL C API library
mysql_server_init()	Initialize the MySQL C API library
mysql_set_character_set()	Set default character set for current connection
mysql_set_local_infile_default()	Set the LOAD DATA LOCAL INFILE handler callbacks to their default values
mysql_set_local_infile_handler()	Install application-specific LOAD DATA LOCAL INFILE handler callbacks
mysql_set_server_option()	Sets an option for the connection (like multi-statements)
mysql_sqlstate()	Returns the SQLSTATE error code for the last error
mysql_shutdown()	Shuts down the database server
mysql_ssl_set()	Prepare to establish SSL connection to server
mysql_stat()	Returns the server status as a string
mysql_store_result()	Retrieves a complete result set to the client
mysql_thread_end()	Finalize thread handler
mysql_thread_id()	Returns the current thread ID
mysql_thread_init()	Initialize thread handler
mysql_thread_safe()	Returns 1 if the clients are compiled as thread-safe
mysql_use_result()	Initiates a row-by-row result set retrieval
mysql_warning_count()	Returns the warning count for the previous SQL statement

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling mysql_library_init(). This function exists in both the libmysqlclient C client library and the libmysqld embedded server library, so it is used whether you build a regular client program by linking with the -libmysqlclient flag, or an embedded server application by linking with the -libmysqld flag.

2. Initialize a connection handler by calling mysql_init() and connect to the server by calling mysql_real_connect().

3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)

4. Close the connection to the MySQL server by calling mysql_close().

5. End use of the MySQL library by calling mysql_library_end().

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The purpose of calling `mysql_library_init()` and `mysql_library_end()` is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the client library, they provide improved memory management. If you do not call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

`mysql_library_init()` and `mysql_library_end()` are available as of MySQL 5.0.3. For older versions of MySQL, you can call `mysql_server_init()` and `mysql_server_end()` instead.

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. This should be done prior to any other client library call.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the host name, user name, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. As of MySQL 5.0.13, you can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL statements to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE`, and `EXPLAIN`. Treat these statements the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Choose the approach that is most appropriate for each client application. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you do not have random access to rows within the result set (you can only access rows sequentially), and the number of rows in the result set is unknown until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the statement was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the statement returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, and so forth), and was not expected to return rows. If `mysql_field_count()` is nonzero, the statement should have returned rows, but did not. This indicates that the statement was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` enable you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, enabling you to determine when an error occurred and what it was.

20.6.7 C API Function Descriptions

In the descriptions here, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a MySQL `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or nonzero to indicate an error. Note that “nonzero” means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                  /* correct */
    ... error ...

if (result < 0)              /* incorrect */
    ... error ...

if (result == -1)            /* incorrect */
    ... error ...
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

20.6.7.1 mysql_affected_rows()

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

`mysql_affected_rows()` may be called immediately after executing a statement with `mysql_query()` or `mysql_real_query()`. It returns the number of rows changed, deleted, or inserted by the last statement if it was an `UPDATE`, `DELETE`, or `INSERT`. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows “found”; that is, matched by the `WHERE` clause.

For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value is 1 if the row is inserted as a new row and 2 if an existing row is updated.

Following a `CALL` statement for a stored procedure, `mysql_affected_rows()` returns the value that it would return for the last statement executed within the procedure, or `0` if that statement would return `-1`. Within the procedure, you can use `ROW_COUNT()` at the SQL level to obtain the affected-rows value for individual statements.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`.

Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

Errors

None.

Example

```
char *stmt = "UPDATE products SET cost=cost*1.25
              WHERE group=10";
mysql_query(&mysql,stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

20.6.7.2 mysql_autocommit()

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

20.6.7.3 mysql_change_user()

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password,
const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that include no explicit database specifier.

`mysql_change_user()` fails if the connected user cannot be authenticated or does not have permission to use the database. In this case, the user and database are not changed.

Pass a `db` parameter of `NULL` if you do not want to have a default database.

This function resets the session state as if one had done a new connect and reauthenticated. (See Section 20.6.15, “Controlling Automatic Reconnection Behavior”.) It always performs a `ROLLBACK` of any active transactions, closes and drops all temporary tables, and unlocks all locked tables. Session system variables are reset to the values of the corresponding global system variables. Prepared statements are released and `HANDLER` variables are closed. Locks acquired with `GET_LOCK()` are released. These effects occur even if the user did not change.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`, plus:

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

An unknown error occurred.

- `ER_UNKNOWN_COM_ERROR`

  The MySQL server does not implement this command (probably an old server).

- `ER_ACCESS_DENIED_ERROR`

  The user or password was wrong.

- `ER_BAD_DB_ERROR`

  The database did not exist.

- `ER_DBACCESS_DENIED_ERROR`

  The user did not have access rights to the database.

- `ER_WRONG_DB_NAME`

  The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

20.6.7.4 mysql_character_set_name()

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set name for the current connection.

Return Values

The default character set name

Errors

None.

20.6.7.5 mysql_close()

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

This documentation is for an older version. If you're

Errors

None.

20.6.7.6 mysql_commit()

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

As of MySQL 5.0.3, the action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is 2, the server performs a release after terminating a transaction and closes the client connection. The client program should call `mysql_close()` to close the connection from the client side.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

20.6.7.7 mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Description

This function is deprecated. Use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you cannot retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

20.6.7.8 mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

20.6.7.9 mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number. Specify a value in the range from `0` to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

This documentation is for an older version. If you're

20.6.7.10 mysql_debug()

```
void mysql_debug(const char *debug)
```

Description

Does a `DBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See Section 21.3.3, “The DBUG Package”.

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in `/tmp/client.trace` on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

20.6.7.11 mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `DROP DATABASE` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Example

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

20.6.7.12 mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write debugging information to the error log. The connected user must have the `SUPER` privilege.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.13 mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`, `mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a nonzero value if the end of the result set was reached and zero if an error occurred.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero for success. Nonzero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

20.6.7.14 mysql_errno()

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix B, *Errors, Error Codes, and Common Problems*.

Note

Some functions such as `mysql_fetch_row()` do not set `mysql_errno()` if they succeed. A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL-specific error numbers returned by `mysql_errno()` differ from SQLSTATE values returned by `mysql_sqlstate()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

20.6.7.15 mysql_error()

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function did not fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_error()`, either of these two tests can be used to check for an error:

```
if(*mysql_error(&mysql))
{
  // an error occurred
}

if(mysql_error(&mysql)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. You can choose error messages in several different languages. See Section 10.2, “Setting the Error Message Language”.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

20.6.7.16 mysql_escape_string()

Note

This function should not be used. Use `mysql_real_escape_string()` instead.

This documentation is for an older version. If you're

`mysql_escape_string()` is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler as its first argument and escapes the string according to the current character set. `mysql_escape_string()` does not take a connection argument and does not respect the current character set.

20.6.7.17 mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL does not know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

20.6.7.18 mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. Use this function to retrieve the definition for an arbitrary column. Specify a value for `fieldnr` in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

20.6.7.19 mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

20.6.7.20 mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

This documentation is for an older version. If you're

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null bytes). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n",
                i, lengths[i]);
    }
}
```

20.6.7.21 mysql_fetch_row()

`MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)`

Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise, the field is empty.

Return Values

A `MYSQL_ROW` structure for the next row. `NULL` if there are no more rows to retrieve or if an error occurred.

Errors

Errors are not reset between calls to `mysql_fetch_row()`

- `CR_SERVER_LOST`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i],
              row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

20.6.7.22 mysql_field_count()

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

20.6.7.23 mysql_field_seek()

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)
```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

20.6.7.24 mysql_field_tell()

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Errors

None.

20.6.7.25 mysql_free_result()

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, and so forth. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

20.6.7.26 mysql_get_character_set_info()

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

Description

This function provides information about the default client character set. The default character set may be changed with the `mysql_set_character_set()` function.

This function was added in MySQL 5.0.10.

Example

This example shows the fields that are available in the `MY_CHARSET_INFO` structure:

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set+collation number: %d\n", cs.number);
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

20.6.7.27 mysql_get_client_info()

```
const char *mysql_get_client_info(void)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

Returns a string that represents the MySQL client library version; for example, `"5.0.96"`.

The function value is the MySQL version. For Connector/C, this is the MySQL version on which the Connector/C distribution is based. For more information, see Section 20.6.4.4, “C API Server and Client Library Versions”.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

20.6.7.28 mysql_get_client_version()

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the MySQL client library version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level (or minor version), and `ZZ` is the sub-version within the release level:

```
major_version*10000 + release_level*100 + sub_version
```

For example, `"5.0.96"` is returned as `50096`.

The function value is the MySQL version. For Connector/C, this is the MySQL version on which the Connector/C distribution is based. For more information, see Section 20.6.4.4, “C API Server and Client Library Versions”.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

20.6.7.29 mysql_get_host_info()

```
const char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server host name.

Return Values

A character string representing the server host name and the connection type.

Errors

None.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

20.6.7.30 mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

20.6.7.31 mysql_get_server_info()

```
const char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the MySQL server version; for example, `"5.0.96"`.

Return Values

A character string that represents the MySQL server version.

Errors

None.

20.6.7.32 mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns an integer that represents the MySQL server version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level (or minor version), and `ZZ` is the sub-version within the release level:

```
major_version*10000 + release_level*100 + sub_version
```

For example, `"5.0.96"` is returned as `50096`.

This function is useful in client programs for determining whether some version-specific server capability exists.

Return Values

An integer that represents the MySQL server version.

Errors

None.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

20.6.7.33 mysql_get_ssl_cipher()

```
const char *mysql_get_ssl_cipher(MYSQL *mysql)
```

Description

`mysql_get_ssl_cipher()` returns the encryption cipher used for the given connection to the server. `mysql` is the connection handler returned from `mysql_init()`.

This function was added in MySQL 5.0.23.

Return Values

A string naming the encryption cipher used for the connection, or `NULL` if no cipher is being used.

20.6.7.34 mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Description

This function creates a legal SQL string for use in an SQL statement. See Section 9.1.1, “String Literals”.

The string in the `from` argument is encoded in hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in the `to` argument, followed by a terminating null byte.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null byte.

The return value can be placed into an SQL statement using either `X'value'` or `0xvalue` format. However, the return value does not include the `X'...'` or `0x`. The caller must supply whichever of those is desired.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"X'");
end += mysql_hex_string(end,"What is this",12);
end = strmov(end,"',X'");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
end = strmov(end,"')");

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the encoded string that is placed into `to`, not including the terminating null character.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Errors

None.

20.6.7.35 mysql_info()

```
const char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed statement, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of statement, as described here. The numbers are illustrative only; the string contains values appropriate for the statement.

- `INSERT INTO ... SELECT ...`

  String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `LOAD DATA INFILE ...`

  String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

- `ALTER TABLE`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `UPDATE`

  String format: `Rows matched: 40 Changed: 40 Warnings: 0`

`mysql_info()` returns a non-`NULL` value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed statement. `NULL` if no information is available for the statement.

Errors

None.

20.6.7.36 mysql_init()

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In a nonmulti-threaded environment, `mysql_init()` invokes `mysql_library_init()` automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`. Before calling `mysql_init()`, either call `mysql_library_init()` prior to spawning any threads, or use a mutex to protect the `mysql_library_init()` call. This should be done prior to any other client library call.

Return Values

An initialized `MYSQL*` handle. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

20.6.7.37 mysql_insert_id()

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement. Use this function after you have performed an `INSERT` statement into a table that contains an `AUTO_INCREMENT` field, or have used `INSERT` or `UPDATE` to set a column value with `LAST_INSERT_ID(`*`expr`*`)`.

More precisely, `mysql_insert_id()` is updated under these conditions:

- `INSERT` statements that store a value into an `AUTO_INCREMENT` column. This is true whether the value is automatically generated by storing the special values `NULL` or `0` into the column, or is an explicit nonspecial value.
- In the case of a multiple-row `INSERT` statement, `mysql_insert_id()` returns the *first* automatically generated `AUTO_INCREMENT` value; if no such value is generated, it returns the *last* explicit value inserted into the `AUTO_INCREMENT` column.

  If no rows are successfully inserted, `mysql_insert_id()` returns 0.
- Starting in MySQL 5.0.54, if an `INSERT ... SELECT` statement is executed, and no automatically generated value is successfully inserted, `mysql_insert_id()` returns the ID of the last inserted row.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(`*`expr`*`)` into any column or by updating any column to `LAST_INSERT_ID(`*`expr`*`)`.
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

`mysql_insert_id()` returns `0` if the previous statement does not use an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the statement that generates the value.

The value of `mysql_insert_id()` is not affected by statements such as `SELECT` that return a result set.

The value of `mysql_insert_id()` is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

The `LAST_INSERT_ID()` SQL function returns the most recently generated `AUTO_INCREMENT` value, and is not reset between statements because the value of that function is maintained in the server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Another difference from `mysql_insert_id()` is that `LAST_INSERT_ID()` is not updated if you set an `AUTO_INCREMENT` column to a specific nonspecial value. See Section 12.13, “Information Functions”.

`mysql_insert_id()` returns `0` following a `CALL` statement for a stored procedure that generates an `AUTO_INCREMENT` value because in this case `mysql_insert_id()` applies to `CALL` and not the statement within the procedure. Within the procedure, you can use `LAST_INSERT_ID()` at the SQL level to obtain the `AUTO_INCREMENT` value.

The reason for the differences between `LAST_INSERT_ID()` and `mysql_insert_id()` is that `LAST_INSERT_ID()` is made easy to use in scripts while `mysql_insert_id()` tries to provide more exact information about what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

20.6.7.38 mysql_kill()

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `KILL` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.39 mysql_library_end()

```
void mysql_library_end(void)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

This function finalizes the MySQL library. Call it when you are done using the library (for example, after disconnecting from the server). The action taken by the call depends on whether your application is linked to the MySQL client library or the MySQL embedded server library. For a client program linked against the `libmysqlclient` library by using the `-lmysqlclient` flag, `mysql_library_end()` performs some memory management to clean up. For an embedded server application linked against the `libmysqld` library by using the `-lmysqld` flag, `mysql_library_end()` shuts down the embedded server and then cleans up.

For usage information, see Section 20.6.6, “C API Function Overview”, and Section 20.6.7.40, “mysql_library_init()”.

`mysql_library_end()` was added in MySQL 5.0.3. For older versions of MySQL, call `mysql_server_end()` instead.

20.6.7.40 mysql_library_init()

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

Call this function to initialize the MySQL library before you call any other MySQL function, whether your application is a regular client program or uses the embedded server. If the application uses the embedded server, this call starts the server and initializes any subsystems (`mysys`, `InnoDB`, and so forth) that the server uses.

After your application is done using the MySQL library, call `mysql_library_end()` to clean up. See Section 20.6.7.39, “mysql_library_end()”.

The choice of whether the application operates as a regular client or uses the embedded server depends on whether you use the `libmysqlclient` or `libmysqld` library at link time to produce the final executable. For additional information, see Section 20.6.6, “C API Function Overview”.

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. Do this prior to any other client library call.

The `argc` and `argv` arguments are analogous to the arguments to `main()`, and enable passing of options to the embedded server. For convenience, `argc` may be `0` (zero) if there are no command-line arguments for the server. This is the usual case for applications intended for use only as regular (nonembedded) clients, and the call typically is written as `mysql_library_init(0, NULL, NULL)`.

```
#include <mysql.h>
#include <stdlib.h>

int main(void) {
  if (mysql_library_init(0, NULL, NULL)) {
    fprintf(stderr, "could not initialize MySQL library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  return EXIT_SUCCESS;
}
```

When arguments are to be passed (`argc` is greater than `0`), the first element of `argv` is ignored (it typically contains the program name). `mysql_library_init()` makes a copy of the arguments so it is safe to destroy `argv` or `groups` after the call.

For embedded applications, if you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `groups` argument is an array of strings that indicate the groups in option files from which to read options. See Section 4.2.6, “Using Option Files”. Make the final entry in the array `NULL`. For convenience, if the `groups` argument itself is `NULL`, the `[server]` and `[embedded]` groups are used by default.

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
  "this_program",       /* this string is not used */
  "--datadir=.",
  "--key_buffer_size=32M"
};
static char *server_groups[] = {
  "embedded",
  "server",
  "this_program_SERVER",
  (char *)NULL
};

int main(void) {
  if (mysql_library_init(sizeof(server_args) / sizeof(char *),
                         server_args, server_groups)) {
    fprintf(stderr, "could not initialize MySQL library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();

  return EXIT_SUCCESS;
}
```

`mysql_library_init()` was added in MySQL 5.0.3. For older versions of MySQL, call `mysql_server_init()` instead.

Return Values

Zero for success. Nonzero if an error occurred.

20.6.7.41 mysql_list_dbs()

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters “`%`” or “`_`”, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW DATABASES [LIKE wild]`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.42 mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns an empty result set for which the metadata provides information about the columns in the given table that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters "`%`" or "`_`", or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

It is preferable to use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
int i;
MYSQL_RES *tbl_cols = mysql_list_fields(mysql, "mytbl", "f%");

unsigned int field_cnt = mysql_num_fields(tbl_cols);
printf("Number of columns: %d\n", field_cnt);

for (i=0; i < field_cnt; ++i)
{
  /* col describes i-th column of the table */
  MYSQL_FIELD *col = mysql_fetch_field_direct(tbl_cols, i);
  printf ("Column %d: %s\n", i, col->name);
}
mysql_free_result(tbl_cols);
```

20.6.7.43 mysql_list_processes()

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist` or a `SHOW PROCESSLIST` query.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.44 mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters "`%`" or "`_`", or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW TABLES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.45 mysql_more_results()

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you execute `CALL` statements, which can return multiple result sets.

`mysql_more_results()` true if more results exist from the currently executed statement, in which case the application must call `mysql_next_result()` to fetch the results.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See Section 20.6.16, "C API Support for Multiple Statement Execution", and Section 20.6.7.46, "mysql_next_result()".

Errors

None.

20.6.7.46 mysql_next_result()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
int mysql_next_result(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you use `CALL` statements to execute stored procedures, which can return multiple result sets.

`mysql_next_result()` reads the next statement result and returns a status to indicate whether more results exist. If `mysql_next_result()` returns an error, there are no more results.

Before each call to `mysql_next_result()`, you must call `mysql_free_result()` for the current statement if it is a statement that returned a result set (rather than just a result status).

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next statement. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`).

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_next_result()` to advance to the next result.

For an example that shows how to use `mysql_next_result()`, see Section 20.6.16, “C API Support for Multiple Statement Execution”.

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order. For example, if you did not call `mysql_use_result()` for a previous result set.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.47 mysql_num_fields()

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

To pass a `MYSQL*` argument instead, use `unsigned int mysql_field_count(MYSQL *mysql)`.

Description

Returns the number of columns in a result set.

You can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
           fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` returns 0. This happens only if something went wrong.

20.6.7.48 mysql_num_rows()

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

`mysql_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

20.6.7.49 mysql_options()

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

Call `mysql_options()` after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, specify a pointer to the value of the integer as the `arg` argument.

The following list describes the possible options, their effect, and how `arg` is used for each option. Several of the options apply only when the application is linked against the `libmysqld` embedded server library and are unused for applications linked against the `libmysqlclient` client library. For option descriptions that indicate `arg` is unused, its value is irrelevant; it is conventional to pass 0.

- `MYSQL_INIT_COMMAND` (argument type: `char *`)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SQL statement to execute when connecting to the MySQL server. Automatically re-executed if reconnection occurs.

- `MYSQL_OPT_COMPRESS` (argument: not used)

  Use the compressed client/server protocol.

- `MYSQL_OPT_CONNECT_TIMEOUT` (argument type: `unsigned int *`)

  The connect timeout in seconds.

- `MYSQL_OPT_GUESS_CONNECTION` (argument: not used)

  For an application linked against the `libmysqld` embedded server library, this enables the library to guess whether to use the embedded server or a remote server. “Guess” means that if the host name is set and is not `localhost`, it uses a remote server. This behavior is the default. `MYSQL_OPT_USE_EMBEDDED_CONNECTION` and `MYSQL_OPT_USE_REMOTE_CONNECTION` can be used to override it. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_LOCAL_INFILE` (argument type: optional pointer to `unsigned int`)

  If no pointer is given or if pointer points to an `unsigned int` that has a nonzero value, the `LOAD DATA LOCAL INFILE` statement is enabled.

- `MYSQL_OPT_NAMED_PIPE` (argument: not used)

  Use a named pipe to connect to the MySQL server on Windows, if the server permits named-pipe connections.

- `MYSQL_OPT_PROTOCOL` (argument type: `unsigned int *`)

  Type of protocol to use. Specify one of the enum values of `mysql_protocol_type` defined in `mysql.h`.

- `MYSQL_OPT_READ_TIMEOUT` (argument type: `unsigned int *`)

  The timeout in seconds for each attempt to read from the server. There are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP `Close_Wait_Timeout` value of 10 minutes. This option works only for TCP/IP connections and, prior to MySQL 5.0.25, only for Windows.

- `MYSQL_OPT_RECONNECT` (argument type: `my_bool *`)

  Enable or disable automatic reconnection to the server if the connection is found to have been lost. Reconnect has been off by default since MySQL 5.0.3; this option is new in 5.0.13 and provides a way to set reconnection behavior explicitly.

  Note: `mysql_real_connect()` incorrectly reset the `MYSQL_OPT_RECONNECT` option to its default value before MySQL 5.0.19. Therefore, prior to that version, if you want reconnect to be enabled for each connection, you must call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option after each call to `mysql_real_connect()`. This is not necessary as of 5.0.19: Call `mysql_options()` only before `mysql_real_connect()` as usual.

- `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` (argument type: `my_bool *`)

  Enable or disable verification of the server's Common Name value in its certificate against the host name used when connecting to the server. The connection is rejected if there is a mismatch. For encrypted

This documentation is for an older version. If you're
This documentation is for an older version. If you're

connections, this feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default. Added in MySQL 5.0.23.

- `MYSQL_OPT_USE_EMBEDDED_CONNECTION` (argument: not used)

  For an application linked against the `libmysqld` embedded server library, this forces the use of the embedded server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_USE_REMOTE_CONNECTION` (argument: not used)

  For an application linked against the `libmysqld` embedded server library, this forces the use of a remote server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_USE_RESULT` (argument: not used)

  This option is unused.

- `MYSQL_OPT_WRITE_TIMEOUT` (argument type: `unsigned int *`)

  The timeout in seconds for each attempt to write to the server. There is a retry if necessary, so the total effective timeout value is two times the option value. This option works only for TCP/IP connections and, prior to MySQL 5.0.25, only for Windows.

- `MYSQL_READ_DEFAULT_FILE` (argument type: `char *`)

  Read options from the named option file instead of from `my.cnf`.

- `MYSQL_READ_DEFAULT_GROUP` (argument type: `char *`)

  Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`.

- `MYSQL_REPORT_DATA_TRUNCATION` (argument type: `my_bool *`)

  Enable or disable reporting of data truncation errors for prepared statements using the `error` member of `MYSQL_BIND` structures. (Default: enabled.) Added in 5.0.3.

- `MYSQL_SECURE_AUTH` (argument type: `my_bool *`)

  Whether to connect to a server that does not support the password hashing used in MySQL 4.1.1 and later.

- `MYSQL_SET_CHARSET_DIR` (argument type: `char *`)

  The path name to the directory that contains character set definition files.

- `MYSQL_SET_CHARSET_NAME` (argument type: `char *`)

  The name of the character set to use as the default character set.

- `MYSQL_SET_CLIENT_IP` (argument type: `char *`)

  For an application linked against the `libmysqld` embedded server library (when `libmysqld` is compiled with authentication support), this means that the user is considered to have connected from the specified IP address (specified as a string) for authentication purposes. This option is ignored for applications linked against the `libmysqlclient` client library.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `MYSQL_SHARED_MEMORY_BASE_NAME` (argument type: `char *`)

  The name of the shared-memory object for communication to the server on Windows, if the server supports shared-memory connections. Specify the same value as the `--shared-memory-base-name` option used for the `mysqld` server you want to connect to.

The `client` group is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options.

Option	Description
`character-sets-dir=dir_name`	The directory where character sets are installed.
`compress`	Use the compressed client/server protocol.
`connect-timeout=seconds`	The connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
`database=db_name`	Connect to this database if no database was specified in the connect command.
`debug`	Debug options.
`default-character-set=charset_name`	The default character set to use.
`disable-local-infile`	Disable use of `LOAD DATA LOCAL INFILE`.
`host=host_name`	Default host name.
`init-command=stmt`	Statement to execute when connecting to MySQL server. Automatically re-executed if reconnection occurs.
`interactive-timeout=seconds`	Same as specifying `CLIENT_INTERACTIVE` to `mysql_real_connect()`. See Section 20.6.7.52, “mysql_real_connect()”.
`local-infile[={0|1}]`	If no argument or nonzero argument, enable use of `LOAD DATA LOCAL`; otherwise disable.
`max_allowed_packet=bytes`	Maximum size of packet that client can read from server.
`multi-queries`, `multi-results`	Enable multiple result sets from multiple-statement executions or stored procedures.
`multi-statements`	Enable the client to send multiple statements in a single string (separated by `;` characters).
`password=password`	Default password.
`pipe`	Use named pipes to connect to a MySQL server on Windows.
`port=port_num`	Default port number.
`protocol={TCP|SOCKET|PIPE|MEMORY}`	The protocol to use when connecting to the server.
`return-found-rows`	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.
`shared-memory-base-name=name`	Shared-memory name to use to connect to server.
`socket={file_name|pipe_name}`	Default socket file.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option	Description
`ssl-ca=file_name`	Certificate Authority file.
`ssl-capath=dir_name`	Certificate Authority directory.
`ssl-cert=file_name`	Certificate file.
`ssl-cipher=cipher_list`	Permissible SSL ciphers.
`ssl-key=file_name`	Key file.
`timeout=seconds`	Like `connect-timeout`.
`user`	Default user.

`timeout` has been replaced by `connect-timeout`, but `timeout` is still supported for backward compatibility.

For more information about option files used by MySQL programs, see Section 4.2.6, “Using Option Files”.

Return Values

Zero for success. Nonzero if you specify an unknown option.

Example

The following `mysql_options()` calls request the use of compression in the client/server protocol, cause options to be read from the `[odbc]` group of option files, and disable transaction autocommit mode:

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
mysql_options(&mysql,MYSQL_INIT_COMMAND,"SET autocommit=0");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

This code requests that the client use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

20.6.7.50 mysql_ping()

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down and auto-reconnect is enabled an attempt to reconnect is made. If the connection is down and auto-reconnect is disabled, `mysql_ping()` returns an error.

Auto-reconnect is enabled by default before MySQL 5.0.3 and enabled from 5.0.3 on. To enable auto-connect, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option. For details, see Section 20.6.7.49, “mysql_options()”.

`mysql_ping()` can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If `mysql_ping()`) does cause a reconnect, there is no explicit indication of it. To determine whether a reconnect occurs, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier has changed.

If reconnect occurs, some characteristics of the connection will have been reset. For details about these characteristics, see Section 20.6.15, “Controlling Automatic Reconnection Behavior”.

Return Values

Zero if the connection to the server is active. Nonzero if an error occurred. A nonzero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.51 mysql_query()

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

Description

Executes the SQL statement pointed to by the null-terminated string `stmt_str`. Normally, the string must consist of a single SQL statement without a terminating semicolon (“`;`”) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 20.6.16, “C API Support for Multiple Statement Execution”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “`\0`” character, which `mysql_query()` interprets as the end of the statement string.)

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See Section 20.6.7.22, “mysql_field_count()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.52 mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned long client_flag)
```

Description

`mysql_real_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any other API functions that require a valid `MYSQL` connection handle structure.

The parameters are specified as follows:

- For the first parameter, specify the address of an existing `MYSQL` structure. Before calling `mysql_real_connect()`, call `mysql_init()` to initialize the `MYSQL` structure. You can change a lot of connect options with the `mysql_options()` call. See Section 20.6.7.49, “mysql_options()”.

- The value of `host` may be either a host name or an IP address. If `host` is `NULL` or the string `"localhost"`, a connection to the local host is assumed. For Windows, the client connects using a shared-memory connection, if the server has shared-memory connections enabled. Otherwise, TCP/IP is used. For Unix, the client connects using a Unix socket file. For local connections, you can also influence the type of connection to use with the `MYSQL_OPT_PROTOCOL` or `MYSQL_OPT_NAMED_PIPE` options to `mysql_options()`. The type of connection must be supported by the server. For a `host` value of `"."` on Windows, the client connects using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.

- The `user` parameter contains the user's MySQL login ID. If `user` is `NULL` or the empty string `""`, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. See the Connector/ODBC section of Chapter 20, *Connectors and APIs*.

- The `passwd` parameter contains the password for `user`. If `passwd` is `NULL`, only entries in the `user` table for the user that have a blank (empty) password field are checked for a match. This enables the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether they have specified a password.

  Note

  Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.

- The `user` and `passwd` parameters use whatever character set has been configured for the `MYSQL` object. By default, this is `latin1`, but can be changed by calling `mysql_options(mysql, MYSQL_SET_CHARSET_NAME, "charset_name")` prior to connecting.

- `db` is the database name. If `db` is not `NULL`, the connection sets the default database to this value.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- If `port` is not 0, the value is used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe to use. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags to enable certain features.

Flag Name	Flag Description
`CLIENT_COMPRESS`	Use compression in the client/server protocol.
`CLIENT_FOUND_ROWS`	Return the number of found (matched) rows, not the number of changed rows.
`CLIENT_IGNORE_SIGPIPE`	Prevents the client library from installing a `SIGPIPE` signal handler. This can be used to avoid conflicts with a handler that the application has already installed.
`CLIENT_IGNORE_SPACE`	Permit spaces after function names. Makes all functions names reserved words.
`CLIENT_INTERACTIVE`	Permit `interactive_timeout` seconds of inactivity (rather than `wait_timeout` seconds) before closing the connection. The client's session `wait_timeout` variable is set to the value of the session `interactive_timeout` variable.
`CLIENT_LOCAL_FILES`	Enable `LOAD DATA LOCAL` handling.
`CLIENT_MULTI_RESULTS`	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This flag is automatically enabled if `CLIENT_MULTI_STATEMENTS` is enabled. See the note following this table for more information about this flag.
`CLIENT_MULTI_STATEMENTS`	Tell the server that the client may send multiple statements in a single string (separated by `;` characters). If this flag is not set, multiple-statement execution is disabled. See the note following this table for more information about this flag.
`CLIENT_NO_SCHEMA`	Do not permit *`db_name.tbl_name.col_name`* syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
`CLIENT_ODBC`	Unused.
`CLIENT_SSL`	Use SSL (encrypted protocol). Do not set this option within an application program; it is set internally in the client library. Instead, use `mysql_ssl_set()` before calling `mysql_real_connect()`.
`CLIENT_REMEMBER_OPTIONS`	Remember options specified by calls to `mysql_options()`. Without this option, if `mysql_real_connect()` fails, you must repeat the `mysql_options()` calls before trying to connect again. With this option, the `mysql_options()` calls need not be repeated.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`).

If you enable `CLIENT_MULTI_STATEMENTS` or `CLIENT_MULTI_RESULTS`, you should process the result for every call to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 20.6.16, “C API Support for Multiple Statement Execution”.

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the `mysql_real_connect()` call. To do this, call `mysql_options()` with the `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP` option before calling `mysql_real_connect()`. Then, in the `mysql_real_connect()` call, specify the “no-value” value for each parameter to be read from an option file:

- For `host`, specify a value of `NULL` or the empty string (`""`).
- For `user`, specify a value of `NULL` or the empty string.
- For `passwd`, specify a value of `NULL`. (For the password, a value of the empty string in the `mysql_real_connect()` call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For `db`, specify a value of `NULL` or the empty string.
- For `port`, specify a value of 0.
- For `unix_socket`, specify a value of `NULL`.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A `MYSQL*` connection handle if the connection was successful, `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- `CR_CONN_HOST_ERROR`

  Failed to connect to the MySQL server.

- `CR_CONNECTION_ERROR`

  Failed to connect to the local MySQL server.

- `CR_IPSOCK_ERROR`

  Failed to create an IP socket.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SOCKET_CREATE_ERROR`

  Failed to create a Unix socket.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_UNKNOWN_HOST`

  Failed to find the IP address for the host name.

- `CR_VERSION_ERROR`

  A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version.

- `CR_NAMEDPIPEOPEN_ERROR`

  Failed to create a named pipe on Windows.

- `CR_NAMEDPIPEWAIT_ERROR`

  Failed to wait for a named pipe on Windows.

- `CR_NAMEDPIPESETSTATE_ERROR`

  Failed to get a pipe handler on Windows.

- `CR_SERVER_LOST`

  If `connect_timeout` > 0 and it took longer than `connect_timeout` seconds to connect to the server or if the server died while executing the `init-command`.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

By using `mysql_options()` the MySQL library reads the `[client]` and `[your_prog_name]` sections in the `my.cnf` file which ensures that your program works, even if someone has set up MySQL in some nonstandard way.

Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. As of MySQL 5.0.13, you can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior.

20.6.7.53 mysql_real_escape_string()

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

Description

This function creates a legal SQL string for use in an SQL statement. See Section 9.1.1, “String Literals”.

The `mysql` argument must be a valid, open connection because character escaping depends on the character set in use by the server.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The string in the `from` argument is encoded to produce an escaped SQL string, taking into account the current character set of the connection. The result is placed in the `to` argument, followed by a terminating null byte.

Characters encoded are “`\`”, “`'`”, “`"`”, `NUL` (ASCII 0), “`\n`”, “`\r`”, and Control+Z. Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. `mysql_real_escape_string()` quotes the other characters to make them easier to read in log files. For comparison, see the quoting rules for literal strings and the `QUOTE()` SQL function in Section 9.1.1, “String Literals”, and Section 12.5, “String Functions”.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and there must be room for the terminating null byte.) When `mysql_real_escape_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null byte.

If you must change the character set of the connection, use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement. `mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string()`, which `SET NAMES` does not.

Example

The following example inserts two escaped strings into an `INSERT` statement, each within single quote characters:

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table VALUES(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql,end,"What is this",12);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql,end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the encoded string that is placed into the `to` argument, not including the terminating null character.

Errors

None.

20.6.7.54 mysql_real_query()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)
```

Description

Executes the SQL statement pointed to by `stmt_str`, a string `length` bytes long. Normally, the string must consist of a single SQL statement without a terminating semicolon (“`;`”) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 20.6.16, “C API Support for Multiple Statement Execution”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “`\0`” character, which `mysql_query()` interprets as the end of the statement string.) In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the statement string.

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See Section 20.6.7.22, “mysql_field_count()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.55 mysql_refresh()

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

This function flushes tables or caches, or resets replication server information. The connected user must have the `RELOAD` privilege.

The `options` argument is a bit mask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- `REFRESH_GRANT`

  Refresh the grant tables, like `FLUSH PRIVILEGES`.

- `REFRESH_LOG`

  Flush the logs, like `FLUSH LOGS`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `REFRESH_TABLES`

  Flush the table cache, like `FLUSH TABLES`.

- `REFRESH_HOSTS`

  Flush the host cache, like `FLUSH HOSTS`.

- `REFRESH_STATUS`

  Reset status variables, like `FLUSH STATUS`.

- `REFRESH_THREADS`

  Flush the thread cache.

- `REFRESH_SLAVE`

  On a slave replication server, reset the master server information and restart the slave, like `RESET SLAVE`.

- `REFRESH_MASTER`

  On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like `RESET MASTER`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.56 mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the `RELOAD` privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.57 mysql_rollback()

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

As of MySQL 5.0.3, the action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is 2, the server performs a release after terminating a transaction and closes the client connection. The client program should call `mysql_close()` to close the connection from the client side.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

20.6.7.58 mysql_row_seek()

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset, typically a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

20.6.7.59 mysql_row_tell()

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

Use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

20.6.7.60 mysql_select_db()

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that include no explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.61 mysql_set_character_set()

```
int mysql_set_character_set(MYSQL *mysql, const char *csname)
```

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

This function was added in MySQL 5.0.7.

Return Values

Zero for success. Nonzero if an error occurred.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
           mysql_character_set_name(&mysql));
}
```

20.6.7.62 mysql_set_local_infile_default()

```
void mysql_set_local_infile_default(MYSQL *mysql);
```

Description

Sets the `LOAD DATA LOCAL INFILE` callback functions to the defaults used internally by the C client library. The library calls this function automatically if `mysql_set_local_infile_handler()` has not been called or does not supply valid functions for each of its callbacks.

Return Values

None.

Errors

None.

20.6.7.63 mysql_set_local_infile_handler()

```
void mysql_set_local_infile_handler(MYSQL *mysql, int (*local_infile_init)(void
**, const char *, void *), int (*local_infile_read)(void *, char *, unsigned
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
int), void (*local_infile_end)(void *), int (*local_infile_error)(void *,
char*, unsigned int), void *userdata);
```

Description

This function installs callbacks to be used during the execution of `LOAD DATA LOCAL INFILE` statements. It enables application programs to exert control over local (client-side) data file reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use `mysql_set_local_infile_handler()`, you must write the following callback functions:

```
int
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open the data file, allocate data structures, and so forth. The first `void**` argument is a pointer to a pointer. You can set the pointer (that is, `*ptr`) to a value that will be passed to each of the other callbacks (as a `void*`). The callbacks can use this pointed-to value to maintain state information. The `userdata` argument is the same value that is passed to `mysql_set_local_infile_handler()`.

The initialization function should return zero for success, nonzero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. `buf` points to the buffer where the read data is stored, and `buf_len` is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after `local_infile_read()` has returned zero (EOF) or an error. Within this function, deallocate any memory allocated by `local_infile_init()` and perform any other cleanup necessary. It is invoked even if the initialization function returns an error.

```
int
local_infile_error(void *ptr,
                   char *error_msg,
                   unsigned int error_msg_len);
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. `error_msg` points to the buffer into which the message is written, and `error_msg_len` is the length of the buffer. Write the message as a null-terminated string, at most `error_msg_len`−1 bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by `ptr`, so that `local_infile_error()` can copy the message from there into `error_msg`.

After calling `mysql_set_local_infile_handler()` in your C code and passing pointers to your callback functions, you can then issue a `LOAD DATA LOCAL INFILE` statement (for example, by using

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`mysql_query()`). The client library automatically invokes your callbacks. The file name specified in `LOAD DATA LOCAL INFILE` will be passed as the second parameter to the `local_infile_init()` callback.

Return Values

None.

Errors

None.

20.6.7.64 mysql_set_server_option()

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values.

Option	Description
`MYSQL_OPTION_MULTI_STATEMENTS_ON`	Enable multiple-statement support
`MYSQL_OPTION_MULTI_STATEMENTS_OFF`	Disable multiple-statement support

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 20.6.16, “C API Support for Multiple Statement Execution”.

Enabling multiple-statement support with `MYSQL_OPTION_MULTI_STATEMENTS_ON` does not have quite the same effect as enabling it by passing the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`: `CLIENT_MULTI_STATEMENTS` also enables `CLIENT_MULTI_RESULTS`. If you are using the `CALL` SQL statement in your programs, multiple-result support must be enabled; this means that `MYSQL_OPTION_MULTI_STATEMENTS_ON` by itself is insufficient to permit the use of `CALL`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `ER_UNKNOWN_COM_ERROR`

  The server did not support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server did not support the option one tried to set.

20.6.7.65 mysql_shutdown()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have the `SHUTDOWN` privilege. The `shutdown_level` argument was added in MySQL 5.0.1. MySQL servers support only one type of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Dynamically linked executables which have been compiled with older versions of the `libmysqlclient` headers and call `mysql_shutdown()` need to be used with the old `libmysqlclient` dynamic library.

The shutdown process is described in Section 5.1.10, “The Server Shutdown Process”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.66 mysql_sqlstate()

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the most recently executed SQL statement. The error code consists of five characters. `'00000'` means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix B, *Errors, Error Codes, and Common Problems*.

SQLSTATE values returned by `mysql_sqlstate()` differ from MySQL-specific error numbers returned by `mysql_errno()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Not all MySQL error numbers are mapped to SQLSTATE error codes. The value `'HY000'` (general error) is used for unmapped error numbers.

If you call `mysql_sqlstate()` after `mysql_real_connect()` fails, `mysql_sqlstate()` might not return a useful value. For example, this happens if a host is blocked by the server and the connection is closed without any SQLSTATE value being sent to the client.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See Section 20.6.7.14, “mysql_errno()”, Section 20.6.7.15, “mysql_error()”, and Section 20.6.11.26, “mysql_stmt_sqlstate()”.

20.6.7.67 mysql_ssl_set()

```
my_bool mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const
char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless SSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the path name to the key file.
- `cert` is the path name to the certificate file.
- `ca` is the path name to the certificate authority file.
- `capath` is the path name to a directory that contains trusted SSL CA certificates in PEM format.
- `cipher` is a list of permissible ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns `0`. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

20.6.7.68 mysql_stat()

```
const char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.69 mysql_store_result()

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

You need not call `mysql_store_result()` or `mysql_use_result()` for other statements, but it does not do any harm or cause any notable performance degradation if you call `mysql_store_result()` in all cases. You can detect whether the statement has a result set by checking whether `mysql_store_result()` returns a nonzero value (more about this later).

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 20.6.16, “C API Support for Multiple Statement Execution”.

If you want to know whether a statement should return a result set, you can use `mysql_field_count()` to check for this. See Section 20.6.7.22, “mysql_field_count()”.

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

`mysql_store_result()` returns a null pointer if the statement did not return a result set (for example, if it was an `INSERT` statement).

`mysql_store_result()` also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking whether `mysql_error()` returns a nonempty string, `mysql_errno()` returns nonzero, or `mysql_field_count()` returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

After you have called `mysql_store_result()` and gotten back a result that is not a null pointer, you can call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

See Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

A `MYSQL_RES` result structure with the results. `NULL` (0) if an error occurred.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.70 mysql_thread_id()

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

Note

This function does not work correctly if thread IDs become larger than 32 bits, which can occur on some systems. To avoid problems with `mysql_thread_id()`, do not use it. To get the connection ID, execute a `SELECT CONNECTION_ID()` query and retrieve the result.

Return Values

The thread ID of the current connection.

Errors

None.

20.6.7.71 mysql_use_result()

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you should not use `mysql_use_result()` for locking reads if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the unfetched rows are returned as part of the result set for your next query. The C API gives the error `Commands out of sync; you can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

When using the `libmysqld` embedded server, the memory benefits are essentially lost because memory usage incrementally increases with each row retrieved until `mysql_free_result()` is called.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

20.6.7.72 mysql_warning_count()

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of errors, warnings, and notes generated during execution of the previous SQL statement.

Return Values

The warning count.

Errors

None.

20.6.8 C API Prepared Statements

The MySQL client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and nonprepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

For a list of SQL statements that can be used as prepared statements, see Section 13.5, “SQL Syntax for Prepared Statements”.

20.6.9 C API Prepared Statement Data Structures

Prepared statements use several data structures:

- To obtain a statement handle, pass a `MYSQL` connection handler to `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT` data structure. This structure is used for further operations with the statement. To specify the statement to prepare, pass the `MYSQL_STMT` pointer and the statement string to `mysql_stmt_prepare()`.
- To provide input parameters for a prepared statement, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_param()`. To receive output column values, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_result()`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `MYSQL_TIME` structure is used to transfer temporal data in both directions.

The following discussion describes the prepared statement data types in detail. For examples that show how to use them, see Section 20.6.11.10, “mysql_stmt_execute()”, and Section 20.6.11.11, “mysql_stmt_fetch()”.

- `MYSQL_STMT`

  This structure is a handle for a prepared statement. A handle is created by calling `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT`. The handle is used for all subsequent operations with the statement until you close it with `mysql_stmt_close()`, at which point the handle becomes invalid.

  The `MYSQL_STMT` structure has no members intended for application use. Applications should not try to copy a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

  Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- `MYSQL_BIND`

  This structure is used both for statement input (data values sent to the server) and output (result values returned from the server):

  - For input, use `MYSQL_BIND` structures with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`.
  - For output, use `MYSQL_BIND` structures with `mysql_stmt_bind_result()` to bind buffers to result set columns, for use in fetching rows with `mysql_stmt_fetch()`.

  To use a `MYSQL_BIND` structure, zero its contents to initialize it, then set its members appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

```
MYSQL_BIND bind[3];
memset(bind, 0, sizeof(bind));
```

  The `MYSQL_BIND` structure contains the following members for use by application programs. For several of the members, the manner of use depends on whether the structure is used for input or output.

  - `enum enum_field_types buffer_type`

    The type of the buffer. This member indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored. For permissible `buffer_type` values, see Section 20.6.9.1, “C API Prepared Statement Type Codes”.

  - `void *buffer`

    A pointer to the buffer to be used for data transfer. This is the address of a C language variable.

    For input, `buffer` is a pointer to the variable in which you store the data value for a statement parameter. When you call `mysql_stmt_execute()`, MySQL use the value stored in the variable in place of the corresponding parameter marker in the statement (specified with `?` in the statement string).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For output, `buffer` is a pointer to the variable in which to return a result set column value. When you call `mysql_stmt_fetch()`, MySQL stores a column value from the current row of the result set in this variable. You can access the value when the call returns.

To minimize the need for MySQL to perform type conversions between C language values on the client side and SQL values on the server side, use C variables that have types similar to those of the corresponding SQL values:

- For numeric data types, `buffer` should point to a variable of the proper numeric C type. For integer variables (which can be `char` for single-byte values or an integer type for larger values), you should also indicate whether the variable has the `unsigned` attribute by setting the `is_unsigned` member, described later.
- For character (nonbinary) and binary string data types, `buffer` should point to a character buffer.
- For date and time data types, `buffer` should point to a `MYSQL_TIME` structure.

For guidelines about mapping between C types and SQL types and notes about type conversions, see Section 20.6.9.1, “C API Prepared Statement Type Codes”, and Section 20.6.9.2, “C API Prepared Statement Type Conversions”.

- `unsigned long buffer_length`

  The actual size of `*buffer` in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the `buffer_length` value specifies the length of `*buffer` when used with `mysql_stmt_bind_param()` to specify input values, or the maximum number of output data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

- `unsigned long *length`

  A pointer to an `unsigned long` variable that indicates the actual number of bytes of data stored in `*buffer`. `length` is used for character or binary C data.

  For input parameter data binding, set `*length` to indicate the actual length of the parameter value stored in `*buffer`. This is used by `mysql_stmt_execute()`.

  For output value binding, MySQL sets `*length` when you call `mysql_stmt_fetch()`. The `mysql_stmt_fetch()` return value determines how to interpret the length:

  - If the return value is 0, `*length` indicates the actual length of the parameter value.
  - If the return value is `MYSQL_DATA_TRUNCATED`, `*length` indicates the nontruncated length of the parameter value. In this case, the minimum of `*length` and `buffer_length` indicates the actual length of the value.

  `length` is ignored for numeric and temporal data types because the `buffer_type` value determines the length of the data value.

  If you must determine the length of a returned value before fetching it, see Section 20.6.11.11, “mysql_stmt_fetch()”, for some strategies.

- `my_bool *is_null`

  This member points to a `my_bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`is_null` is a *pointer* to a boolean scalar, not a boolean scalar, to provide flexibility in how you specify `NULL` values:

- If your data values are always `NULL`, use `MYSQL_TYPE_NULL` as the `buffer_type` value when you bind the column. The other `MYSQL_BIND` members, including `is_null`, do not matter.
- If your data values are always `NOT NULL`, set `is_null = (my_bool*) 0`, and set the other members appropriately for the variable you are binding.
- In all other cases, set the other members appropriately and set `is_null` to the address of a `my_bool` variable. Set that variable's value to true or false appropriately between executions to indicate whether the corresponding data value is `NULL` or `NOT NULL`, respectively.

For output, when you fetch a row, MySQL sets the value pointed to by `is_null` to true or false according to whether the result set column value returned from the statement is or is not `NULL`.

- `my_bool is_unsigned`

  This member applies for C variables with data types that can be `unsigned` (`char`, `short int`, `int`, `long long int`). Set `is_unsigned` to true if the variable pointed to by `buffer` is `unsigned` and false otherwise. For example, if you bind a `signed char` variable to `buffer`, specify a type code of `MYSQL_TYPE_TINY` and set `is_unsigned` to false. If you bind an `unsigned char` instead, the type code is the same but `is_unsigned` should be true. (For `char`, it is not defined whether it is signed or unsigned, so it is best to be explicit about signedness by using `signed char` or `unsigned char`.)

  `is_unsigned` applies only to the C language variable on the client side. It indicates nothing about the signedness of the corresponding SQL value on the server side. For example, if you use an `int` variable to supply a value for a `BIGINT UNSIGNED` column, `is_unsigned` should be false because `int` is a signed type. If you use an `unsigned int` variable to supply a value for a `BIGINT` column, `is_unsigned` should be true because `unsigned int` is an unsigned type. MySQL performs the proper conversion between signed and unsigned values in both directions, although a warning occurs if truncation results.

- `my_bool *error`

  For output, set this member to point to a `my_bool` variable to have truncation information for the parameter stored there after a row fetching operation. When truncation reporting is enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `*error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option. The `error` member was added in MySQL 5.0.3.

- `MYSQL_TIME`

  This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. Set the `buffer` member to point to a `MYSQL_TIME` structure, and set the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types (`MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATE`, `MYSQL_TYPE_DATETIME`, `MYSQL_TYPE_TIMESTAMP`).

  The `MYSQL_TIME` structure contains the members listed in the following table.

Member	Description
`unsigned int year`	The year

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Member	Description
`unsigned int month`	The month of the year
`unsigned int day`	The day of the month
`unsigned int hour`	The hour of the day
`unsigned int minute`	The minute of the hour
`unsigned int second`	The second of the minute
`my_bool neg`	A boolean flag indicating whether the time is negative
`unsigned long second_part`	The fractional part of the second in microseconds; currently unused

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used. The `year`, `month`, and `day` elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The `hour`, `minute`, and `second` elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See Section 20.6.18, “C API Prepared Statement Handling of Date and Time Values”.

20.6.9.1 C API Prepared Statement Type Codes

The `buffer_type` member of `MYSQL_BIND` structures indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored.

The following table shows the permissible values for the `buffer_type` member of `MYSQL_BIND` structures for input values sent to the server. The table shows the C variable types that you can use, the corresponding type codes, and the SQL data types for which the supplied value can be used without conversion. Choose the `buffer_type` value according to the data type of the C language variable that you are binding. For the integer types, you should also set the `is_unsigned` member to indicate whether the variable is signed or unsigned.

Input Variable C Type	`buffer_type` Value	SQL Type of Destination Value
`signed char`	`MYSQL_TYPE_TINY`	`TINYINT`
`short int`	`MYSQL_TYPE_SHORT`	`SMALLINT`
`int`	`MYSQL_TYPE_LONG`	`INT`
`long long int`	`MYSQL_TYPE_LONGLONG`	`BIGINT`
`float`	`MYSQL_TYPE_FLOAT`	`FLOAT`
`double`	`MYSQL_TYPE_DOUBLE`	`DOUBLE`
`MYSQL_TIME`	`MYSQL_TYPE_TIME`	`TIME`
`MYSQL_TIME`	`MYSQL_TYPE_DATE`	`DATE`
`MYSQL_TIME`	`MYSQL_TYPE_DATETIME`	`DATETIME`
`MYSQL_TIME`	`MYSQL_TYPE_TIMESTAMP`	`TIMESTAMP`
`char[]`	`MYSQL_TYPE_STRING`	`TEXT`, `CHAR`, `VARCHAR`
`char[]`	`MYSQL_TYPE_BLOB`	`BLOB`, `BINARY`, `VARBINARY`
	`MYSQL_TYPE_NULL`	`NULL`

Use `MYSQL_TYPE_NULL` as indicated in the description for the `is_null` member in Section 20.6.9, “C API Prepared Statement Data Structures”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For input string data, use MYSQL_TYPE_STRING or MYSQL_TYPE_BLOB depending on whether the value is a character (nonbinary) or binary string:

- MYSQL_TYPE_STRING indicates character input string data. The value is assumed to be in the character set indicated by the character_set_client system variable. If the server stores the value into a column with a different character set, it converts the value to that character set.
- MYSQL_TYPE_BLOB indicates binary input string data. The value is treated as having the binary character set. That is, it is treated as a byte string and no conversion occurs.

The following table shows the permissible values for the buffer_type member of MYSQL_BIND structures for output values received from the server. The table shows the SQL types of received values, the corresponding type codes that such values have in result set metadata, and the recommended C language data types to bind to the MYSQL_BIND structure to receive the SQL values without conversion. Choose the buffer_type value according to the data type of the C language variable that you are binding. For the integer types, you should also set the is_unsigned member to indicate whether the variable is signed or unsigned.

SQL Type of Received Value	buffer_type Value	Output Variable C Type
TINYINT	MYSQL_TYPE_TINY	signed char
SMALLINT	MYSQL_TYPE_SHORT	short int
MEDIUMINT	MYSQL_TYPE_INT24	int
INT	MYSQL_TYPE_LONG	int
BIGINT	MYSQL_TYPE_LONGLONG	long long int
FLOAT	MYSQL_TYPE_FLOAT	float
DOUBLE	MYSQL_TYPE_DOUBLE	double
DECIMAL	MYSQL_TYPE_NEWDECIMAL	char[]
YEAR	MYSQL_TYPE_SHORT	short int
TIME	MYSQL_TYPE_TIME	MYSQL_TIME
DATE	MYSQL_TYPE_DATE	MYSQL_TIME
DATETIME	MYSQL_TYPE_DATETIME	MYSQL_TIME
TIMESTAMP	MYSQL_TYPE_TIMESTAMP	MYSQL_TIME
CHAR, BINARY	MYSQL_TYPE_STRING	char[]
VARCHAR, VARBINARY	MYSQL_TYPE_VAR_STRING	char[]
TINYBLOB, TINYTEXT	MYSQL_TYPE_TINY_BLOB	char[]
BLOB, TEXT	MYSQL_TYPE_BLOB	char[]
MEDIUMBLOB, MEDIUMTEXT	MYSQL_TYPE_MEDIUM_BLOB	char[]
LONGBLOB, LONGTEXT	MYSQL_TYPE_LONG_BLOB	char[]
BIT	MYSQL_TYPE_BIT	char[]

20.6.9.2 C API Prepared Statement Type Conversions

Prepared statements transmit data between the client and server using C language variables on the client side that correspond to SQL values on the server side. If there is a mismatch between the C variable type on the client side and the corresponding SQL value type on the server side, MySQL performs implicit type conversions in both directions.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL knows the type code for the SQL value on the server side. The `buffer_type` value in the `MYSQL_BIND` structure indicates the type code of the C variable that holds the value on the client side. The two codes together tell MySQL what conversion must be performed, if any. Here are some examples:

- If you use `MYSQL_TYPE_LONG` with an `int` variable to pass an integer value to the server that is to be stored into a `FLOAT` column, MySQL converts the value to floating-point format before storing it.
- If you fetch an SQL `MEDIUMINT` column value, but specify a `buffer_type` value of `MYSQL_TYPE_LONGLONG` and use a C variable of type `long long int` as the destination buffer, MySQL converts the `MEDIUMINT` value (which requires less than 8 bytes) for storage into the `long long int` (an 8-byte variable).
- If you fetch a numeric column with a value of 255 into a `char[4]` character array and specify a `buffer_type` value of `MYSQL_TYPE_STRING`, the resulting value in the array is a 4-byte string `'255\0'`.
- MySQL returns `DECIMAL` values as the string representation of the original server-side value, which is why the corresponding C type is `char[]`. For example, `12.345` is returned to the client as `'12.345'`. If you specify `MYSQL_TYPE_NEWDECIMAL` and bind a string buffer to the `MYSQL_BIND` structure, `mysql_stmt_fetch()` stores the value in the buffer as a string without conversion. If instead you specify a numeric variable and type code, `mysql_stmt_fetch()` converts the string-format `DECIMAL` value to numeric form.
- For the `MYSQL_TYPE_BIT` type code, `BIT` values are returned into a string buffer, which is why the corresponding C type is `char[]`. The value represents a bit string that requires interpretation on the client side. To return the value as a type that is easier to deal with, you can cause the value to be cast to integer using either of the following types of expressions:

  ```
  SELECT bit_col + 0 FROM t
  SELECT CAST(bit_col AS UNSIGNED) FROM t
  ```

  To retrieve the value, bind an integer variable large enough to hold the value and specify the appropriate corresponding integer type code.

Before binding variables to the `MYSQL_BIND` structures that are to be used for fetching column values, you can check the type codes for each column of the result set. This might be desirable if you want to determine which variable types would be best to use to avoid type conversions. To get the type codes, call `mysql_stmt_result_metadata()` after executing the prepared statement with `mysql_stmt_execute()`. The metadata provides access to the type codes for the result set as described in Section 20.6.11.22, “mysql_stmt_result_metadata()”, and Section 20.6.5, “C API Data Structures”.

To determine whether output string values in a result set returned from the server contain binary or nonbinary data, check whether the `charsetnr` value of the result set metadata is 63 (see Section 20.6.5, “C API Data Structures”). If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

If you cause the `max_length` member of the `MYSQL_FIELD` column metadata structures to be set (by calling `mysql_stmt_attr_set()`), be aware that the `max_length` values for the result set indicate the lengths of the longest string representation of the result values, not the lengths of the binary representation. That is, `max_length` does not necessarily correspond to the size of the buffers needed to fetch the values with the binary protocol used for prepared statements. Choose the size of the buffers according to the types of the variables into which you fetch the values. For example, a `TINYINT` column containing the value -128 might have a `max_length` value of 4. But the binary representation of any

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`TINYINT` value requires only 1 byte for storage, so you can supply a `signed char` variable in which to store the value and set `is_unsigned` to indicate that values are signed.

20.6.10 C API Prepared Statement Function Overview

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See Section 20.6.11, “C API Prepared Statement Function Descriptions”.

Function	Description
`mysql_stmt_affected_rows()`	Returns the number of rows changed, deleted, or inserted by prepared `UPDATE`, `DELETE`, or `INSERT` statement
`mysql_stmt_attr_get()`	Gets value of an attribute for a prepared statement
`mysql_stmt_attr_set()`	Sets an attribute for a prepared statement
`mysql_stmt_bind_param()`	Associates application data buffers with the parameter markers in a prepared SQL statement
`mysql_stmt_bind_result()`	Associates application data buffers with columns in a result set
`mysql_stmt_close()`	Frees memory used by a prepared statement
`mysql_stmt_data_seek()`	Seeks to an arbitrary row number in a statement result set
`mysql_stmt_errno()`	Returns the error number for the last statement execution
`mysql_stmt_error()`	Returns the error message for the last statement execution
`mysql_stmt_execute()`	Executes a prepared statement
`mysql_stmt_fetch()`	Fetches the next row of data from a result set and returns data for all bound columns
`mysql_stmt_fetch_column()`	Fetch data for one column of the current row of a result set
`mysql_stmt_field_count()`	Returns the number of result columns for the most recent statement
`mysql_stmt_free_result()`	Free the resources allocated to a statement handle
`mysql_stmt_init()`	Allocates memory for a `MYSQL_STMT` structure and initializes it
`mysql_stmt_insert_id()`	Returns the ID generated for an `AUTO_INCREMENT` column by a prepared statement
`mysql_stmt_num_rows()`	Returns the row count from a buffered statement result set
`mysql_stmt_param_count()`	Returns the number of parameters in a prepared statement
`mysql_stmt_param_metadata()`	(Return parameter metadata in the form of a result set) This function does nothing
`mysql_stmt_prepare()`	Prepares an SQL statement string for execution
`mysql_stmt_reset()`	Resets the statement buffers in the server
`mysql_stmt_result_metadata()`	Returns prepared statement metadata in the form of a result set
`mysql_stmt_row_seek()`	Seeks to a row offset in a statement result set, using value returned from `mysql_stmt_row_tell()`
`mysql_stmt_row_tell()`	Returns the statement row cursor position
`mysql_stmt_send_long_data()`	Sends long data in chunks to server
`mysql_stmt_sqlstate()`	Returns the SQLSTATE error code for the last statement execution
`mysql_stmt_store_result()`	Retrieves a complete result set to the client

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Call `mysql_stmt_init()` to create a statement handle, then `mysql_stmt_prepare()` to prepare the statement string, `mysql_stmt_bind_param()` to supply the parameter data, and `mysql_stmt_execute()` to execute the statement. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through `mysql_stmt_bind_param()`.

You can send text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See Section 20.6.11.25, “mysql_stmt_send_long_data()”.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

When statement execution has been completed, close the statement handle using `mysql_stmt_close()` so that all resources associated with it can be freed.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with `mysql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.
2. If the statement will produce a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.
3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.
6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.
7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and the number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the current row of the result set and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error number, error message, and SQLSTATE code using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes `Prepare` and `Execute` lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string `"SELECT ?"`.
2. Call `mysql_stmt_bind_param()` to bind the value `3` to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare  [1] SELECT ?
Execute  [1] SELECT 3
```

Each `Prepare` and `Execute` line in the log is tagged with a `[N]` statement identifier so that you can keep track of which prepared statement is being logged. *`N`* is a positive integer. If there are multiple prepared statements active simultaneously for the client, *`N`* may be greater than 1. Each `Execute` lines shows a prepared statement after substitution of data values for `?` parameters.

20.6.11 C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

To create a `MYSQL_STMT` handle, use the `mysql_stmt_init()` function.

20.6.11.1 mysql_stmt_affected_rows()

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_affected_rows()` may be called immediately after executing a statement with `mysql_stmt_execute()`. It is like `mysql_affected_rows()` but for prepared statements. For a description of what the affected-rows value returned by this function means, See Section 20.6.7.1, “mysql_affected_rows()”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Errors

None.

Example

See the Example in Section 20.6.11.10, “mysql_stmt_execute()”.

20.6.11.2 mysql_stmt_attr_get()

```
my_bool mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, `arg` should point to the value of the integer.

See Section 20.6.11.3, “mysql_stmt_attr_set()”, for a list of options and option types.

Note

In MySQL 5.0, `mysql_stmt_attr_get()` uses `unsigned long *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This was corrected in MySQL 5.1.7.

Return Values

Zero for success. Nonzero if `option` is unknown.

Errors

None.

20.6.11.3 mysql_stmt_attr_set()

```
my_bool mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. This function may be called multiple times to set several options.

The `option` argument is the option that you want to set. The `arg` argument is the value for the option. `arg` should point to a variable that is set to the desired attribute value. The variable type is as indicated in the following table.

The following table shows the possible `option` values.

Option	Argument Type	Function
`STMT_ATTR_UPDATE_MAX_LENGTH`	`my_bool *`	If set to 1, causes `mysql_stmt_store_result()` to update the metadata `MYSQL_FIELD->max_length` value.
`STMT_ATTR_CURSOR_TYPE`	`unsigned long *`	Type of cursor to open for statement when `mysql_stmt_execute()`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option	Argument Type	Function
		is invoked. `*arg` can be `CURSOR_TYPE_NO_CURSOR` (the default) or `CURSOR_TYPE_READ_ONLY`.
`STMT_ATTR_PREFETCH_ROWS`	`unsigned long *`	Number of rows to fetch from server at a time when using a cursor. `*arg` can be in the range from 1 to the maximum value of `unsigned long`. The default is 1.

Note

In MySQL 5.0, `mysql_stmt_attr_get()` uses `unsigned int *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This is corrected in MySQL 5.1.7.

If you use the `STMT_ATTR_CURSOR_TYPE` option with `CURSOR_TYPE_READ_ONLY`, a cursor is opened for the statement when you invoke `mysql_stmt_execute()`. If there is already an open cursor from a previous `mysql_stmt_execute()` call, it closes the cursor before opening a new one. `mysql_stmt_reset()` also closes any open cursor before preparing the statement for re-execution. `mysql_stmt_free_result()` closes any open cursor.

If you open a cursor for a prepared statement, `mysql_stmt_store_result()` is unnecessary, because that function causes the result set to be buffered on the client side.

The `STMT_ATTR_CURSOR_TYPE` option was added in MySQL 5.0.2. The `STMT_ATTR_PREFETCH_ROWS` option was added in MySQL 5.0.6.

Return Values

Zero for success. Nonzero if `option` is unknown.

Errors

None.

Example

The following example opens a cursor for a prepared statement and sets the number of rows to fetch at a time to 5:

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                         (void*) &prefetch_rows);
/* ... check return value ... */
```

20.6.11.4 mysql_stmt_bind_param()

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Description

`mysql_stmt_bind_param()` is used to bind input data for the parameter markers in the SQL statement that was passed to `mysql_stmt_prepare()`. It uses `MYSQL_BIND` structures to supply the data. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each `?` parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

Section 20.6.9, “C API Prepared Statement Data Structures”, describes the members of each `MYSQL_BIND` element and how they should be set to provide input values.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 20.6.11.10, “mysql_stmt_execute()”.

20.6.11.5 mysql_stmt_bind_result()

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (that is, bind) output columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol does not return data values in chunks.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol places data into the newly bound buffers when the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and length of the output buffer into which the value should be stored. Section 20.6.9, “C API Prepared Statement Data Structures”, describes the members of each `MYSQL_BIND` element and how they should be set to receive output values.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 20.6.11.11, “mysql_stmt_fetch()”.

20.6.11.6 mysql_stmt_close()

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Example

See the Example in Section 20.6.11.10, “mysql_stmt_execute()”.

20.6.11.7 mysql_stmt_data_seek()

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from `0` to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

None.

Errors

None.

20.6.11.8 mysql_stmt_errno()

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix B, *Errors, Error Codes, and Common Problems*.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

20.6.11.9 mysql_stmt_error()

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. Either of these two tests can be used to check for an error:

```
if(*mysql_stmt_errno(stmt))
{
  // an error occurred
}
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. You can choose error messages in several different languages.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

20.6.11.10 mysql_stmt_execute()

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

Statement processing following `mysql_stmt_execute()` depends on the type of statement:

- For an `UPDATE`, `DELETE`, or `INSERT`, the number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`.
- For a statement such as `SELECT` that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to Section 20.6.11.11, “mysql_stmt_fetch()”.

  Do not following invocation of `mysql_stmt_execute()` with a call to `mysql_store_result()` or `mysql_use_result()`. Those functions are not intended for processing results from prepared statements.

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using `mysql_stmt_init()`, `mysql_stmt_prepare()`, `mysql_stmt_param_count()`, `mysql_stmt_bind_param()`, `mysql_stmt_execute()`, and `mysql_stmt_affected_rows()`. The `mysql` variable is assumed to be a valid connection handle. For an example that shows how to retrieve data, see Section 20.6.11.11, “mysql_stmt_fetch()”.

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                     col2 VARCHAR(40),\
                                                     col3 SMALLINT,\
                                                     col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO \
                       test_table(col1,col2,col3) \
                       VALUES(?,?,?)"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
my_ulonglong  affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
my_bool       is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
   to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
               (unsigned long) affected_rows);
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
        The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
is_null= 0;               /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

Note

For complete examples on the use of prepared statement functions, refer to the file `tests/mysql_client_test.c`. This file can be obtained from a MySQL source distribution or from the source repository (see Section 2.17, “Installing MySQL from Source”).

20.6.11.11 mysql_stmt_fetch()

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_fetch()` returns the next row in the result set. It can be called only while the result set exists; that is, after a call to `mysql_stmt_execute()` for a statement such as `SELECT` that produces a result set.

`mysql_stmt_fetch()` returns row data using the buffers bound by `mysql_stmt_bind_result()`. It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the `length` pointer. All columns must be bound by the application before it calls `mysql_stmt_fetch()`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

By default, result sets are fetched unbuffered a row at a time from the server. To buffer the entire result set on the client, call `mysql_stmt_store_result()` after binding the data buffers and before calling `mysql_stmt_fetch()`.

If a fetched data value is a `NULL` value, the `*is_null` value of the corresponding `MYSQL_BIND` structure contains TRUE (1). Otherwise, the data and its length are returned in the `*buffer` and `*length` elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by `data_length`.

Type	Length
`MYSQL_TYPE_TINY`	1
`MYSQL_TYPE_SHORT`	2
`MYSQL_TYPE_LONG`	4
`MYSQL_TYPE_LONGLONG`	8
`MYSQL_TYPE_FLOAT`	4
`MYSQL_TYPE_DOUBLE`	8
`MYSQL_TYPE_TIME`	`sizeof(MYSQL_TIME)`
`MYSQL_TYPE_DATE`	`sizeof(MYSQL_TIME)`
`MYSQL_TYPE_DATETIME`	`sizeof(MYSQL_TIME)`
`MYSQL_TYPE_STRING`	`data length`
`MYSQL_TYPE_BLOB`	`data_length`

In some cases you might want to determine the length of a column value before fetching it with `mysql_stmt_fetch()`. For example, the value might be a long string or `BLOB` value for which you want to know how much space must be allocated. To accomplish this, you can use these strategies:

- Before invoking `mysql_stmt_fetch()` to retrieve individual rows, pass `STMT_ATTR_UPDATE_MAX_LENGTH` to `mysql_stmt_attr_set()`, then invoke `mysql_stmt_store_result()` to buffer the entire result on the client side. Setting the `STMT_ATTR_UPDATE_MAX_LENGTH` attribute causes the maximal length of column values to be indicated by the `max_length` member of the result set metadata returned by `mysql_stmt_result_metadata()`.

- Invoke `mysql_stmt_fetch()` with a zero-length buffer for the column in question and a pointer in which the real length can be stored. Then use the real length with `mysql_stmt_fetch_column()`.

```
real_length= 0;

bind[0].buffer= 0;
bind[0].buffer_length= 0;
bind[0].length= &real_length
mysql_stmt_bind_result(stmt, bind);

mysql_stmt_fetch(stmt);
if (real_length > 0)
{
  data= malloc(real_length);
  bind[0].buffer= data;
  bind[0].buffer_length= real_length;
  mysql_stmt_fetch_column(stmt, bind, 0, 0);
}
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.
1	Error occurred. Error code and message can be obtained by calling `mysql_stmt_errno()` and `mysql_stmt_error()`.
`MYSQL_NO_DATA`	No more rows/data exists
`MYSQL_DATA_TRUNCATED`	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is returned when truncation reporting is enabled. To determine which column values were truncated when this value is returned, check the `error` members of the `MYSQL_BIND` structures used for fetching values. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

- `CR_UNSUPPORTED_PARAM_TYPE`

  The buffer type is `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, but the data type is not `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP`.

- All other unsupported conversion errors are returned from `mysql_stmt_bind_result()`.

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in Section 20.6.11.10, “mysql_stmt_execute()”.) The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
                       FROM test_table"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
my_bool       is_null[4];
my_bool       error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_metadata(), \
           returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout,
        " total columns in SELECT statement: %d\n",
        column_count);

if (column_count != 4) /* validate column count */
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Now buffer all results to client (optional step) */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
  row_count++;
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                    ts.year, ts.month, ts.day,
                    ts.hour, ts.minute, ts.second,
                    length[3]);
  fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);


/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

20.6.11.12 mysql_stmt_fetch_column()

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int
column, unsigned long offset)
```

Description

Fetch one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO`

  Invalid column number.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `CR_NO_DATA`

  The end of the result set has already been reached.

20.6.11.13 mysql_stmt_field_count()

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

20.6.11.14 mysql_stmt_free_result()

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, `mysql_stmt_free_result()` closes it.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

20.6.11.15 mysql_stmt_init()

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a `MYSQL_STMT` handle. The handle should be freed with `mysql_stmt_close(MYSQL_STMT *)`.

See also Section 20.6.9, “C API Prepared Statement Data Structures”, for more information.

Return Values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Out of memory.

20.6.11.16 mysql_stmt_insert_id()

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the prepared `INSERT` or `UPDATE` statement. Use this function after you have executed a prepared `INSERT` statement on a table which contains an `AUTO_INCREMENT` field.

See Section 20.6.7.37, “mysql_insert_id()”, for more information.

Return Values

Value for `AUTO_INCREMENT` column which was automatically generated or explicitly set during execution of prepared statement, or value generated by `LAST_INSERT_ID(`*`expr`*`)` function. Return value is undefined if statement does not set `AUTO_INCREMENT` value.

Errors

None.

20.6.11.17 mysql_stmt_num_rows()

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handle. If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately. Otherwise, the row count is unavailable unless you count the rows as you fetch them.

`mysql_stmt_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_stmt_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

20.6.11.18 mysql_stmt_param_count()

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

See the Example in Section 20.6.11.10, “mysql_stmt_execute()”.

20.6.11.19 mysql_stmt_param_metadata()

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

Description

Return Values

Errors

20.6.11.20 mysql_stmt_prepare()

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long
length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `stmt_str` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (“`;`”) or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (`?`) characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are permitted in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not permitted for identifiers (such as table or column names), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns nonzero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

See the Example in Section 20.6.11.10, “mysql_stmt_execute()”.

20.6.11.21 mysql_stmt_reset()

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Resets a prepared statement on client and server to state after prepare. It resets the statement on the server, data sent using `mysql_stmt_send_long_data()`, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

An unknown error occurred.

20.6.11.22 mysql_stmt_result_metadata()

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysql_stmt_fetch()`.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 20.6.11.11, “mysql_stmt_fetch()”.

20.6.11.23 mysql_stmt_row_seek()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

20.6.11.24 mysql_stmt_row_tell()

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

20.6.11.25 mysql_stmt_send_long_data()

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int
parameter_number, const char *data, unsigned long length)
```

Description

Enables an application to send parameter data to the server in pieces (or “chunks”). Call this function after `mysql_stmt_bind_param()` and before `mysql_stmt_execute()`. It can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the `TEXT` or `BLOB` data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Note

The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See Section 20.6.11.21, “mysql_stmt_reset()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE`

  The parameter does not have a string or binary type.

- `CR_INVALID_PARAMETER_NO`

  Invalid parameter number.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

The following example demonstrates how to send the data for a `TEXT` column in chunks. It inserts the data value `'MySQL - The most popular Open Source database'` into the `text_column` column. The `mysql` variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO \
                      test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long       length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
 memset(bind, 0, sizeof(bind));
 bind[0].buffer_type= MYSQL_TYPE_STRING;
 bind[0].length= &length;
 bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply data in chunks to server */
 if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply the next piece of data */
 if (mysql_stmt_send_long_data(stmt,0,
          " - The most popular Open Source database",40))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

20.6.11.26 mysql_stmt_sqlstate()

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. `"00000"` means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix B, *Errors, Error Codes, and Common Problems*.

Not all MySQL errors are mapped to SQLSTATE codes. The value `"HY000"` (general error) is used for unmapped errors.

Return Values

A null-terminated character string containing the SQLSTATE error code.

20.6.11.27 mysql_stmt_store_result()

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```


This documentation is for an older version. If you're

Description

Result sets are produced by calling `mysql_stmt_execute()` to executed prepared statements for SQL statements such as `SELECT`, `SHOW`, `DESCRIBE`, and `EXPLAIN`. By default, result sets for successfully executed prepared statements are not buffered on the client and `mysql_stmt_fetch()` fetches them one at a time from the server. To cause the complete result set to be buffered on the client, call `mysql_stmt_store_result()` after binding data buffers with `mysql_stmt_bind_result()` and before calling `mysql_stmt_fetch()` to fetch rows. (For an example, see Section 20.6.11.11, “mysql_stmt_fetch()”.)

`mysql_stmt_store_result()` is optional for result set processing, unless you will call `mysql_stmt_data_seek()`, `mysql_stmt_row_seek()`, or `mysql_stmt_row_tell()`. Those functions require a seekable result set.

It is unnecessary to call `mysql_stmt_store_result()` after executing an SQL statement that does not produce a result set, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to Section 20.6.11.22, “mysql_stmt_result_metadata()”.

Note

MySQL does not by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications do not need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See Section 20.6.11.3, “mysql_stmt_attr_set()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

If the application is linked to the embedded server library, runtime error messages will indicate the `libmysqld` rather than `libmysqlclient` library, but the solution to the problem is the same as just described.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

20.6.12 C API Threaded Function Descriptions

To create a threaded client, use the functions described in the following sections. See also Section 20.6.4.2, “Writing C API Threaded Client Programs”.

20.6.12.1 my_init()

```
void my_init(void)
```

Description

`my_init()` initializes some global variables that MySQL needs. If you are using a thread-safe client library, it also calls `mysql_thread_init()` for this thread.

It is necessary for `my_init()` to be called early in the initialization phase of a program's use of the MySQL library. However, `my_init()` is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you ensure that your program invokes one of those functions before any other MySQL calls, there is no need to invoke `my_init()` explicitly.

To access the prototype for `my_init()`, your program should include these header files:

```
#include <my_global.h>
#include <my_sys.h>
```

Return Values

None.

20.6.12.2 mysql_thread_end()

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

`mysql_thread_end()` *is not invoked automatically by the client library*. It must be called explicitly to avoid a memory leak.

Return Values

None.

20.6.12.3 mysql_thread_init()

```
my_bool mysql_thread_init(void)
```

Description

This function must be called early within each created thread to initialize thread-specific variables. However, you may not necessarily need to invoke it explicitly: `mysql_thread_init()` is automatically called by `my_init()`, which itself is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you invoke any of those functions, `mysql_thread_init()` will be called for you.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Return Values

Zero for success. Nonzero if an error occurred.

20.6.12.4 mysql_thread_safe()

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client library is compiled as thread-safe.

Return Values

1 if the client library is thread-safe, 0 otherwise.

20.6.13 C API Embedded Server Function Descriptions

MySQL applications can be written to use an embedded server. See Section 20.5, “libmysqld, the Embedded MySQL Server Library”. To write such an application, you must link it against the `libmysqld` library by using the `-lmysqld` flag rather than linking it against the `libmysqlclient` client library by using the `-lmysqlclient` flag. However, the calls to initialize and finalize the library are the same whether you write a client application or one that uses the embedded server: Call `mysql_library_init()` to initialize the library and `mysql_library_end()` when you are done with it. See Section 20.6.6, “C API Function Overview”.

`mysql_library_init()` and `mysql_library_end()` are available as of MySQL 5.0.3. For earlier versions, call `mysql_server_init()` and `mysql_server_end()` instead, which are equivalent. `mysql_library_init()` and `mysql_library_end()` actually are `#define` symbols that make them equivalent to `mysql_server_init()` and `mysql_server_end()`, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`.

20.6.13.1 mysql_server_init()

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function initializes the MySQL library, which must be done before you call any other MySQL function.

As of MySQL 5.0.3, `mysql_server_init()` is deprecated and you should call `mysql_library_init()` instead. See Section 20.6.7.40, “mysql_library_init()”.

Return Values

Zero for success. Nonzero if an error occurred.

20.6.13.2 mysql_server_end()

```
void mysql_server_end(void)
```

Description

This function finalizes the MySQL library. You should call it when you are done using the library.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

As of MySQL 5.0.3, `mysql_server_end()` is deprecated and you should call `mysql_library_end()` instead. See Section 20.6.7.39, “mysql_library_end()”.

Return Values

None.

20.6.14 Common Questions and Problems When Using the C API

20.6.14.1 Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data could not be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a nonempty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a nonzero value, the statement should have produced a nonempty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

20.6.14.2 What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE`, or `DELETE`.

  For a fast re-create, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.

- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See Section 20.6.7.37, “mysql_insert_id()”.

- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

20.6.14.3 How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can check from your C applications whether a value was stored in an AUTO_INCREMENT column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an INSERT with an AUTO_INCREMENT index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

When a new AUTO_INCREMENT value has been generated, you can also obtain it by executing a SELECT LAST_INSERT_ID() statement with mysql_query() and retrieving the value from the result set returned by the statement.

For LAST_INSERT_ID(), the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another AUTO_INCREMENT column with a nonmagic value (that is, a value that is not NULL and not 0). Using LAST_INSERT_ID() and AUTO_INCREMENT columns simultaneously from multiple clients is perfectly valid. Each client will receive the last inserted ID for the last statement *that* client executed.

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');         # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

mysql_insert_id() returns the value stored into an AUTO_INCREMENT column, whether that value is automatically generated by storing NULL or 0 or was specified as an explicit value. LAST_INSERT_ID() returns only automatically generated AUTO_INCREMENT values. If you store an explicit value other than NULL or 0, it does not affect the value returned by LAST_INSERT_ID().

For more information on obtaining the last ID in an AUTO_INCREMENT column:

- For information on LAST_INSERT_ID(), which can be used within an SQL statement, see Section 12.13, “Information Functions”.
- For information on mysql_insert_id(), the function you use from within the C API, see Section 20.6.7.37, “mysql_insert_id()”.
- For information on obtaining the auto-incremented value when using Connector/J, see Retrieving AUTO_INCREMENT Column Values through JDBC.
- For information on obtaining the auto-incremented value when using Connector/ODBC, see Obtaining Auto-Increment Values.

20.6.15 Controlling Automatic Reconnection Behavior

The MySQL client library can perform an automatic reconnection to the server if it finds that the connection is down when you attempt to send a statement to the server to be executed. If auto-reconnect is enabled, the library tries once to reconnect to the server and send the statement again.

If it is important for your application to know that the connection has been dropped (so that it can exit or take action to adjust for the loss of state information), be sure that auto-reconnect is disabled. To ensure this, call mysql_options() with the MYSQL_OPT_RECONNECT option:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
my_bool reconnect = 0;
mysql_options(&mysql, MYSQL_OPT_RECONNECT, &reconnect);
```

Auto-reconnect was enabled by default until MySQL 5.0.3, and disabled by default thereafter. The `MYSQL_OPT_RECONNECT` option is available as of MySQL 5.0.13.

If the connection has gone down, the effect of `mysql_ping()` depends on the auto-reconnect state. If auto-reconnect is enabled, `mysql_ping()` performs a reconnect. Otherwise, it returns an error.

Some client programs might provide the capability of controlling automatic reconnection. For example, `mysql` reconnects by default, but the `--skip-reconnect` option can be used to suppress this behavior.

If an automatic reconnection does occur (for example, as a result of calling `mysql_ping()`), there is no explicit indication of it. To check for reconnection, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier changed.

Automatic reconnection can be convenient because you need not implement your own reconnect code, but if a reconnection does occur, several aspects of the connection state are reset on the server side and your application will not be notified.

The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session system variables are reinitialized to the values of the corresponding global system variables, including system variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` is reset to 0.
- Locks acquired with `GET_LOCK()` are released.

If the connection drops, it is possible that the session associated with the connection on the server side will still be running if the server has not yet detected that the client is no longer connected. In this case, any locks held by the original connection still belong to that session, so you may want to kill it by calling `mysql_kill()`.

20.6.16 C API Support for Multiple Statement Execution

By default, `mysql_query()` and `mysql_real_query()` interpret their statement string argument as a single statement to be executed, and you process the result according to whether the statement produces a result set (a set of rows, as for `SELECT`) or an affected-rows count (as for `INSERT`, `UPDATE`, and so forth).

MySQL also supports the execution of a string containing multiple statements separated by semicolon (`;`) characters. This capability is enabled by special options that are specified either when you connect to the server with `mysql_real_connect()` or after connecting by calling` `mysql_set_server_option()`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Executing a multiple-statement string can produce multiple result sets or row-count indicators. Processing these results involves a different approach than for the single-statement case: After handling the result from the first statement, it is necessary to check whether more results exist and process them in turn if so. To support multiple-result processing, the C API includes the `mysql_more_results()` and `mysql_next_result()` functions. These functions are used at the end of a loop that iterates as long as more results are available. *Failure to process the result this way may result in a dropped connection to the server.*

Multiple-result processing also is required if you execute `CALL` statements for stored procedures. Results from a stored procedure have these characteristics:

- Statements within the procedure may produce result sets (for example, if it executes `SELECT` statements). These result sets are returned in the order that they are produced as the procedure executes.

  In general, the caller cannot know how many result sets a procedure will return. Procedure execution may depend on loops or conditional statements that cause the execution path to differ from one call to the next. Therefore, you must be prepared to retrieve multiple results.

- The final result from the procedure is a status result that includes no result set. The status indicates whether the procedure succeeded or an error occurred.

The multiple statement and result capabilities can be used only with `mysql_query()` or `mysql_real_query()`. They cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement. See Section 20.6.8, “C API Prepared Statements”.

To enable multiple-statement execution and result processing, the following options may be used:

- The `mysql_real_connect()` function has a `flags` argument for which two option values are relevant:

  - `CLIENT_MULTI_RESULTS` enables the client program to process multiple results. This option *must* be enabled if you execute `CALL` statements for stored procedures that produce result sets. Otherwise, such procedures result in an error `Error 1312 (0A000): PROCEDURE` *`proc_name`* `can't return a result set in the given context`.

  - `CLIENT_MULTI_STATEMENTS` enables `mysql_query()` and `mysql_real_query()` to execute statement strings containing multiple statements separated by semicolons. This option also enables `CLIENT_MULTI_RESULTS` implicitly, so a `flags` argument of `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()` is equivalent to an argument of `CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS`. That is, `CLIENT_MULTI_STATEMENTS` is sufficient to enable multiple-statement execution and all multiple-result processing.

- After the connection to the server has been established, you can use the `mysql_set_server_option()` function to enable or disable multiple-statement execution by passing it an argument of `MYSQL_OPTION_MULTI_STATEMENTS_ON` or `MYSQL_OPTION_MULTI_STATEMENTS_OFF`. Enabling multiple-statement execution with this function also enables processing of “simple” results for a multiple-statement string where each statement produces a single result, but is *not* sufficient to permit processing of stored procedures that produce result sets.

The following procedure outlines a suggested strategy for handling multiple statements:

1. Pass `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()`, to fully enable multiple-statement execution and multiple-result processing.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. After calling `mysql_query()` or `mysql_real_query()` and verifying that it succeeds, enter a loop within which you process statement results.

3. For each iteration of the loop, handle the current statement result, retrieving either a result set or an affected-rows count. If an error occurs, exit the loop.

4. At the end of the loop, call `mysql_next_result()` to check whether another result exists and initiate retrieval for it if so. If no more results are available, exit the loop.

One possible implementation of the preceding strategy is shown following. The final part of the loop can be reduced to a simple test of whether `mysql_next_result()` returns nonzero. The code as written distinguishes between no more results and an error, which enables a message to be printed for the latter occurrence.

```
/* connect to server with the CLIENT_MULTI_STATEMENTS option */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
  printf("mysql_real_connect() failed\n");
  mysql_close(mysql);
  exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
                     "DROP TABLE IF EXISTS test_table;\
                      CREATE TABLE test_table(id INT);\
                      INSERT INTO test_table VALUES(10);\
                      UPDATE test_table SET id=20 WHERE id=10;\
                      SELECT * FROM test_table;\
                      DROP TABLE test_table");
if (status)
{
  printf("Could not execute statement(s)");
  mysql_close(mysql);
  exit(0);
}

/* process each statement result */
do {
  /* did current statement return data? */
  result = mysql_store_result(mysql);
  if (result)
  {
    /* yes; process rows and free the result set */
    process_result_set(mysql, result);
    mysql_free_result(result);
  }
  else          /* no result set or error */
  {
    if (mysql_field_count(mysql) == 0)
    {
      printf("%lld rows affected\n",
             mysql_affected_rows(mysql));
    }
    else  /* some error occurred */
    {
      printf("Could not retrieve result set\n");
      break;
    }
  }
  /* more results? -1 = no, >0 = error, 0 = yes (keep looping) */
  if ((status = mysql_next_result(mysql)) > 0)
    printf("Could not execute statement\n");
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
} while (status == 0);

mysql_close(mysql);
```

20.6.17 C API Prepared Statement Problems

Here follows a list of the currently known problems with prepared statements:

- `TIME`, `TIMESTAMP`, and `DATETIME` do not support parts of seconds (for example, from `DATE_FORMAT()`).
- When converting an integer to string, `ZEROFILL` is honored with prepared statements in some cases where the MySQL server does not print the leading zeros. (For example, with `MIN(*number-with-zerofill*)`).
- When converting a floating-point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- *Prepared statements do not use the query cache, even in cases where a query does not contain any placeholders*. See Section 8.10.3.1, “How the Query Cache Operates”.
- Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by `;` characters).
- In MySQL 5.0, prepared `CALL` statements cannot invoke stored procedures that return result sets because prepared statements do not support multiple result sets. Nor can the calling application access a stored procedure's `OUT` or `INOUT` parameters when the procedure returns. These capabilities are supported beginning with MySQL 5.5.

20.6.18 C API Prepared Statement Handling of Date and Time Values

The binary (prepared statement) protocol enables you to send and receive date and time values (`DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`), using the `MYSQL_TIME` structure. The members of this structure are described in Section 20.6.9, “C API Prepared Statement Data Structures”.

To send temporal data values, create a prepared statement using `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the `buffer_type` member to the type that indicates what kind of temporal value you're sending. For `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.
2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.
3. Fill in the members of the `MYSQL_TIME` structure that are appropriate for the type of temporal value to pass.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_stmt_bind_result()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Here is a simple example that inserts `DATE`, `TIME`, and `TIMESTAMP` data. The `mysql` variable is assumed to be a valid connection handle.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
                            timestamp_field) VALUES(?,?,?");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..
```

20.6.19 C API Support for Prepared CALL Statements

In MySQL 5.0, prepared `CALL` statements can be used only for stored procedures that produce at most one result set. Nor can the calling application use placeholders for `OUT` or `INOUT` parameters.

MySQL 5.5 expands prepared `CALL` statement support for stored procedures that produce multiple result sets and to provide placeholder access to `OUT` and `INOUT` parameters.

20.7 MySQL PHP API

The MySQL PHP API manual is now published in standalone form, not as part of the MySQL Reference Manual. See MySQL and PHP.

20.8 MySQL Perl API

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The `DBI` module.
2. The `DBD::mysql` module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see Section 2.22, “Perl Installation Notes”.
- For an example of reading options from option files, see Section 5.5.4, “Using Client Programs in a Multiple-Server Environment”.
- For secure coding tips, see Section 6.1.1, “Security Guidelines”.
- For debugging tips, see Section 21.3.1.4, “Debugging mysqld under gdb”.
- For some Perl-specific environment variables, see Section 2.21, “Environment Variables”.
- For considerations for running on OS X, see Section 2.11, “Installing MySQL on OS X”.
- For ways to quote string literals, see Section 9.1.1, “String Literals”.

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

  ```
  shell> perldoc DBI
  shell> perldoc DBI::FAQ
  shell> perldoc DBD::mysql
  ```

  You can also use `pod2man`, `pod2html`, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI Web site, http://dbi.perl.org/. That site hosts a general DBI mailing list. Oracle Corporation hosts a list specifically about `DBD::mysql`; see Section 1.6.1, “MySQL Mailing Lists”.

- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, http://dbi.perl.org/.

  For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is http://www.kitebird.com/mysql-perl/.

20.9 MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See MySQL Connector/Python Developer Guide for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

20.10 MySQL Ruby APIs

Two APIs are available for Ruby programmers developing MySQL applications:

- The MySQL/Ruby API is based on the `libmysqlclient` API library. For information on installing and using the MySQL/Ruby API, see Section 20.10.1, “The MySQL/Ruby API”.
- The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For information on installing and using the Ruby/MySQL API, see Section 20.10.2, “The Ruby/MySQL API”.

For background and syntax information about the Ruby language, see Ruby Programming Language.

20.10.1 The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysqlclient`.

For information on installing the module, and the functions exposed, see MySQL/Ruby.

20.10.2 The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see Ruby/MySQL.

20.11 MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

20.12 MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 21 Extending MySQL

Table of Contents

21.1 MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. To track or contribute to MySQL development, follow the instructions in Section 2.17.2, “Installing MySQL Using a Development Source Tree”. If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list has relatively low traffic. For details on how to subscribe, please see Section 1.6.1, “MySQL Mailing Lists”. Many MySQL developers at Oracle Corporation are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

21.1.1 MySQL Threads

The MySQL server creates the following threads:

- Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.
- Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

  For information about tuning the parameters that control thread resources, see Section 8.12.6.1, “How MySQL Uses Threads for Client Connections”.
- On a master replication server, connections from slave servers are handled like client connections: There is one thread per connected slave.
- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- A signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If `InnoDB` is used, there will be 4 additional threads by default. Those are file I/O threads, controlled by the `innodb_file_io_threads` parameter. See Section 14.2.2, "InnoDB Startup Options and System Variables".
- If `mysqld` is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
- If the server is started with the `--flush_time=val` option, a dedicated thread is created to flush all tables every *`val`* seconds.
- Each table for which `INSERT DELAYED` statements are issued gets its own thread. See Section 13.2.5.2, "INSERT DELAYED Syntax".

`mysqladmin processlist` only shows the connection, `INSERT DELAYED`, and replication threads.

21.1.2 The MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

You can also write your own test cases. For information about the MySQL Test Framework, including system requirements, see the manual available at http://dev.mysql.com/doc/mysqltest/2.0/en/.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (`mysqltest`), a Perl script to run all tests (`mysql-test-run.pl`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make test` from the source root directory, or change location to the `mysql-test` directory and type `./mysql-test-run.pl`. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run `./mysql-test-run.pl`. All tests should succeed. If any do not, feel free to try to find out why and report the problem if it indicates a bug in MySQL. See Section 1.7, "How to Report Bugs or Problems".

If one test fails, you should run `mysql-test-run.pl` with the `--force` option to check whether any other tests fail.

If you have a copy of `mysqld` running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports `9306` or `9307`. If either of those ports is taken, you should set the `MTR_BUILD_THREAD` environment variable to an appropriate value, and the test suite will use a different set of ports for master, slave, NDB, and Instance Manager). For example:

```
shell> export MTR_BUILD_THREAD=31
shell> ./mysql-test-run.pl [options] [test_name]
```

In the `mysql-test` directory, you can run an individual test case with `./mysql-test-run.pl` *`test_name`*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL `internals` mailing list. See Section 1.6.1, “MySQL Mailing Lists”.

21.2 Adding New Functions to MySQL

There are three ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”.
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.
- Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is not covered here. See Section 18.2, “Using Stored Routines (Procedures and Functions)”.

Each method of creating compiled functions has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` or `SOUNDEX()`.

See Section 9.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.c` file that is provided in MySQL source distributions.

21.2.1 Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values and can accept arguments of those same types.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number, types, and names of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- You can indicate that a function returns `NULL` or that an error occurred.

21.2.2 Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. MySQL source distributions include a file `sql/udf_example.c` that defines five UDF functions. Consult this file to see how UDF calling conventions work. The `include/mysql_com.h` header file defines UDF-related symbols and data structures, although you need not include this header file directly; it is included by `mysql.h`.

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. Note that these constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see Section 2.17.3, “MySQL Source-Configuration Options”, and Section 2.17.4, “Dealing with Problems Compiling MySQL”.

To be able to use UDFs, you must link `mysqld` dynamically. Don't configure MySQL using `--with-mysqld-ldflags=-all-static`. If you want to use a UDF that needs to access symbols from `mysqld` (for example, the `metaphone` function in `sql/udf_example.c` uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`). If you plan to use UDFs, the rule of thumb is to configure MySQL with `--with-mysqld-ldflags=-rdynamic` unless you have a very good reason not to.

For each function that you want to use in SQL statements, you should define corresponding C (or C ++) functions. In the following discussion, the name “xxx” is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

Note

When using C++ you can encapsulate your C functions within:

```
extern "C" { ... }
```

This ensures that your C++ function names remain readable in the completed UDF.

The following list describes the C/C++ functions that you write to implement the interface for a function named `XXX()`. The main function, `xxx()`, is required. In addition, a UDF requires at least one of the other functions described here, for reasons discussed in Section 21.2.2.6, “UDF Security Precautions”.

- `xxx()`

  The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here.

SQL Type	C/C++ Type
`STRING`	`char *`
`INTEGER`	`long long`
`REAL`	`double`

  It is also possible to declare a `DECIMAL` function, but currently the value is returned as a string, so you should write the UDF as though it were a `STRING` function. `ROW` functions are not implemented.

- `xxx_init()`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The initialization function for `xxx()`. If present, it can be used for the following purposes:

- To check the number of arguments to `XXX()`.
- To verify that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the required types when the main function is called.
- To allocate any memory required by the main function.
- To specify the maximum length of the result.
- To specify (for `REAL` functions) the maximum number of decimal places in the result.
- To specify whether the result can be `NULL`.

- `xxx_deinit()`

  The deinitialization function for `xxx()`. If present, it should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `XXX()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function `xxx()` once for each row. After all rows have been processed, MySQL calls the deinitialization function `xxx_deinit()` so that it can perform any required cleanup.

For aggregate functions that work like `SUM()`, you must also provide the following functions:

- `xxx_clear()`

  Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()`

  Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat steps 3 to 5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A

This documentation is for an older version. If you're
This documentation is for an older version. If you're

consequence of this requirement is that you are not permitted to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

21.2.2.1 UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. Section 21.2.2, "Adding a New User-Defined Function", describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

`DECIMAL` functions return string values and should be declared the same way as `STRING` functions. `ROW` functions are not implemented.

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `my_bool maybe_null`

  `xxx_init()` should set `maybe_null` to `1` if `xxx()` can return `NULL`. The default value is `1` if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

  The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. For example, if the function is passed `1.34`, `1.345`, and `1.3`, the default would be 3, because `1.345` has 3 decimal digits.

  For arguments that have no fixed number of decimals, the `decimals` value is set to 31, which is 1 more than the maximum number of decimals permitted for the `DECIMAL`, `FLOAT`, and `DOUBLE` data types.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A `decimals` value of 31 is used for arguments in cases such as a `FLOAT` or `DOUBLE` column declared without an explicit number of decimals (for example, `FLOAT` rather than `FLOAT(10,3)`) and for floating-point constants such as `1345E-3`. It is also used for string and other nonnumber arguments that might be converted within the function to numeric form.

The value to which the `decimals` member is initialized is only a default. It can be changed within the function to reflect the actual calculation performed. The default is determined such that the largest number of decimals of the arguments is used. If the number of decimals is 31 for even one of the arguments, that is the value used for `decimals`.

- `unsigned int max_length`

  The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

  If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- `char *ptr`

  A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

  ```
  initid->ptr = allocated_memory;
  ```

  In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

- `my_bool const_item`

  `xxx_init()` should set `const_item` to `1` if `xxx()` always returns the same value and to `0` otherwise.

21.2.2.2 UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. Section 21.2.2, “Adding a New User-Defined Function”, describes the order in which MySQL calls these functions.

- `xxx_reset()`

  This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given `UDF_ARGS` argument as the first value in your internal summary value for the group. Declare `xxx_reset()` as follows:

  ```
  void xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                 char *is_null, char *error);
  ```

  `xxx_reset()` is not needed or used in MySQL 5.0, in which the UDF interface uses `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work with older versions of the server. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `xxx_clear()`

  This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:

  ```
  void xxx_clear(UDF_INIT *initid, char *is_null, char *error);
  ```

  `is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

  If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

  `xxx_clear()` is required by MySQL 5.0.

- `xxx_add()`

  This function is called for all rows that belong to the same group. You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

  ```
  void xxx_add(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
  ```

The `xxx()` function for an aggregate UDF should be declared the same way as for a nonaggregate UDF. See Section 21.2.2.1, “UDF Calling Sequences for Simple Functions”.

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a nonaggregate UDF. See Section 21.2.2.4, “UDF Return Values and Error Handling”.

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for nonaggregate UDFs. See Section 21.2.2.3, “UDF Argument Processing”.

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

21.2.2.3 UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

  The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

  ```
  if (args->arg_count != 2)
  {
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

For other `UDF_ARGS` member values that are arrays, array references are zero-based. That is, refer to array members using index values from 0 to `args->arg_count` − 1.

- `enum Item_result *arg_type`

  A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, `REAL_RESULT`, and `DECIMAL_RESULT`.

  To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

  Arguments of type `DECIMAL_RESULT` are passed as strings, so you should handle them the same way as `STRING_RESULT` values.

  As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

  Exact-value decimal arguments such as `1.3` or `DECIMAL` column values are passed with a type of `DECIMAL_RESULT`. However, the values are passed as strings. If you want to receive a number, use the initialization function to specify that the argument should be coerced to a `REAL_RESULT` value:

```
args->arg_type[2] = REAL_RESULT;
```

Note

Prior to MySQL 5.0.3, decimal arguments were passed as `REAL_RESULT` values. If you upgrade to a newer version and find that your UDF now receives string values, use the initialization function to coerce the arguments to numbers as just described.

- `char **args`

  `args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a nonconstant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)`. A nonconstant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with nonconstant arguments.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

If argument `i` represents `NULL`, `args->args[i]` is a null pointer (0). If the argument is not `NULL`, functions can refer to it as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to enable handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. Do not assume that the string is null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

```
double    real_val;
real_val = *((double*) args->args[i]);
```

- For an argument of type `DECIMAL_RESULT`, the value is passed as a string and should be handled like a `STRING_RESULT` value.
- `ROW_RESULT` arguments are not implemented.

- `unsigned long *lengths`

  For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

- `char *maybe_null`

  For the initialization function, the `maybe_null` array indicates for each argument whether the argument value might be null (0 if no, 1 if yes).

- `char **attributes`

  `args->attributes` communicates information about the names of the UDF arguments. For argument `i`, the attribute name is available as a string in `args->attributes[i]` and the attribute length is `args->attribute_lengths[i]`. Do not assume that the string is null-terminated.

  By default, the name of a UDF argument is the text of the expression used to specify the argument. For UDFs, an argument may also have an optional `[AS] alias_name` clause, in which case the argument name is *`alias_name`*. The `attributes` value for each argument thus depends on whether an alias was given.

  Suppose that a UDF `my_udf()` is invoked as follows:

```
SELECT my_udf(expr1, expr2 AS alias1, expr3 alias2);
```

  In this case, the `attributes` and `attribute_lengths` arrays will have these values:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
args->attributes[0] = "expr1"
args->attribute_lengths[0] = 5

args->attributes[1] = "alias1"
args->attribute_lengths[1] = 6

args->attributes[2] = "alias2"
args->attribute_lengths[2] = 6
```

- `unsigned long *attribute_lengths`

  The `attribute_lengths` array indicates the length of each argument name.

21.2.2.4 UDF Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*length` to the length (in bytes) of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

MySQL passes a buffer to the `xxx()` function using the `result` parameter. This buffer is sufficiently long to hold 255 characters, which can be multibyte characters. The `xxx()` function can store the result in this buffer if it fits, in which case the return value should be a pointer to the buffer. If the function stores the result in a different buffer, it should return a pointer to that buffer.

If your string function does not use the supplied buffer (for example, if it needs to return a string longer than 255 characters), you must allocate the space for your own buffer with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See Section 21.2.2.1, “UDF Calling Sequences for Simple Functions”.

To indicate a return value of `NULL` in the main function, set `*is_null` to `1`:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

21.2.2.5 UDF Compiling and Installing

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.c` that is included in the MySQL source distribution.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If a UDF will be referred to in statements that will be replicated to slave servers, you must ensure that every slave also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.c` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it is more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP address for a host name.
- `reverse_lookup()` returns the host name for an IP address. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.
- `avgcost()` returns an average cost. This is an aggregate function.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.c
```

If you are using `gcc` with `configure` and `libtool` (which is how MySQL is configured), you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example.la
```

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.c` using `gcc` directly produces a file named `udf_example.so`. Compiling the shared object using `make` produces a file named something like `udf_example.so.0.0.0` in the `.libs` directory (the exact name may vary from platform to platform).

As of MySQL 5.0.67, copy the shared object to server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` system variable.

Prior to MySQL 5.0.67, or if the value of `plugin_dir` is empty, the shared object should be placed in a directory such as `/usr/lib` that is searched by your system's dynamic (runtime) linker, or you can add the directory in which you place the shared object to the linker configuration file (for example, `/etc/ld.so.conf`).

On many systems, you can also set the `LD_LIBRARY` or `LD_LIBRARY_PATH` environment variable to point at the directory where you have the files for your UDF. You should set the variable in `mysql.server` or `mysqld_safe` startup scripts and restart `mysqld`. You might do this if you want to place the object file in a directory accessible only to the server and not in a public directory. The `dlopen` manual page tells you which variable to use on your system.

The dynamic linker name is system-specific (for example, `ld-elf.so.1` on FreeBSD, `ld.so` on Linux, or `dyld` on OS X). Consult your system documentation for information about the linker name and how to configure it.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. Obtain the development source for MySQL 5.0. See Section 2.5, “How to Get MySQL”.
2. Obtain the `CMake` build utility, if necessary, from http://www.cmake.org. (Version 2.6 or later is required).
3. In the source tree, look in the `sql` directory. There are files named `udf_example.def` `udf_example.c` there. Copy both files from this directory to your working directory.
4. Create a `CMake` `makefile` (`CMakeLists.txt`) with these contents:

```
PROJECT(udf_example)

# Path for MySQL include directory
INCLUDE_DIRECTORIES("c:/mysql/include")

ADD_DEFINITIONS("-DHAVE_DLOPEN")
ADD_LIBRARY(udf_example MODULE udf_example.c udf_example.def)
TARGET_LINK_LIBRARIES(udf_example wsock32)
```

5. Create the VC project and solution files:

```
cmake -G "<Generator>"
```

   Invoking `cmake --help` shows you a list of valid Generators.

6. Create `udf_example.dll`:

```
devenv udf_example.sln /build Release
```

After the shared object file has been installed, notify `mysqld` about the new functions with the following statements. If object files have a suffix different from `.so` on your system, substitute the correct suffix throughout (for example, `.dll` on Windows).

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION sequence RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
    ->        RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
    ->        RETURNS REAL SONAME 'udf_example.so';
```

Once installed, a function remains installed until it is uninstalled.

To delete functions, use `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION sequence;
mysql> DROP FUNCTION lookup;
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the `func` system table in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the `INSERT` or `DELETE` privilege for the `mysql` database to create or drop functions, respectively.

You should not use `CREATE FUNCTION` to add a function that has previously been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

21.2.2.6 UDF Security Precautions

MySQL takes several measures to prevent misuse of user-defined functions.

UDF object files cannot be placed in arbitrary directories. They must be located in some system directory that the dynamic linker is configured to search. To enforce this restriction and prevent attempts at specifying path names outside of directories searched by the dynamic linker, MySQL checks the shared object file name specified in `CREATE FUNCTION` statements for path name delimiter characters. As of MySQL 5.0.3, MySQL also checks for path name delimiters in file names stored in the `mysql.func` table when it loads functions. This prevents attempts at specifying illegitimate path names through direct manipulation of the `mysql.func` table. For information about UDFs and the runtime linker, see Section 21.2.2.5, “UDF Compiling and Installing”.

To use `CREATE FUNCTION` or `DROP FUNCTION`, you must have the `INSERT` or `DELETE` privilege, respectively, for the `mysql` database. This is necessary because those statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. As of MySQL 5.0.3, `mysqld` supports an `--allow-suspicious-udfs` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` option. Otherwise, you should avoid enabling this capability.

21.2.3 Adding a New Native Function

To add a new native MySQL function, use the procedure described here, which requires that you use a source distribution. You cannot add native functions to a binary distribution because it is necessary to modify MySQL source code and compile MySQL from the modified source. If you migrate to another version of MySQL (for example, when a new version is released), you must repeat the procedure with the new version.

If the new native function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

To add a new native function, follow these steps to modify source files in the `sql` directory:

1. Add one line to `lex.h` that defines the function name in the `sql_functions[]` array.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

2. If the function prototype is simple (just takes zero, one, two, or three arguments), add a line to the `sql_functions[]` array in `lex.h` that specifies `SYM(FUNC_ARGN)` as the second argument (where *`N`* is the number of arguments the function takes). Also, add a function in `item_create.cc` that creates a function object. Look at `"ABS"` and `create_funcs_abs()` for an example of this.

   If the function prototype is not simple (for example, if it takes a variable number of arguments), you should make two changes to `sql_yacc.yy`. One is a line that indicates the preprocessor symbol that `yacc` should define; this should be added at the beginning of the file. The other is an “item” to be added to the `simple_expr` parsing rule that defines the function parameters. You will need an item for each syntax with which the function can be called. For an example that shows how this is done, check all occurrences of `ATAN` in `sql_yacc.yy`.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.

4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

   ```
   double   Item_func_newname::val()
   longlong Item_func_newname::val_int()
   String  *Item_func_newname::Str(String *str)
   ```

   If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. If the function is nondeterministic, include the following statement in the item constructor to indicate that function results should not be cached:

   ```
   current_thd->lex->safe_to_cache_query=0;
   ```

   A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations.

6. You should probably also define the following object function:

   ```
   void Item_func_newname::fix_length_and_dec()
   ```

   This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. Look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, do not use any global or static variables in the functions without protecting them with mutexes.

If you want to return `NULL` from `::val()`, `::val_int()`, or `::str()`, you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- The `::str()` function should return the string that holds the result, or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

21.3 Debugging and Porting MySQL

This section helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See http://www.mysql.com/support/supportedplatforms/database.html. If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (http://www.mysql.com/), recommending it to other users.

Note

If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server.

Both the server and the client need a working C++ compiler. We use `gcc` on many platforms. Other compilers that are known to work are Sun Studio, HP-UX `aCC`, IBM AIX `xlC_r`), Intel `ecc/icc`. With previous versions on the respective platforms, we also used Irix `cc` and Compaq `cxx`.

Important

If you are trying to build MySQL 5.0 with `icc` on the IA64 platform, and need support for MySQL Cluster, you should first ensure that you are using `icc` version 9.1.043 or later. (For details, see Bug #21875.)

To compile only the client, use `./configure --without-server`.

If you want or need to change any `Makefile` or the `configure` script, you also need GNU Automake and Autoconf. See Section 2.17.2, “Installing MySQL Using a Development Source Tree”.

All steps needed to remake everything from the most basic files.

```
/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug=full --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of MySQL! See Section 21.3.1, “Debugging a MySQL Server”.

Note

Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This ensures that your thread installation has even a remote chance to work!

21.3.1 Debugging a MySQL Server

This documentation is for an older version. If you're

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which disables all new, potentially unsafe functionality). See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”.

If `mysqld` doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `myisamchk`. See Chapter 5, *MySQL Server Administration*. You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 2.20, “Operating System-Specific Notes”.

21.3.1.1 Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `--with-debug` or the `--with-debug=full` option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... --debug` in this case.

If you are using `gcc`, the recommended `configure` line is:

```
CC=gcc CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors \
   -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
   --with-debug --with-extra-charsets=complex
```

This avoids problems with the `libstdc++` library and with C++ exceptions (many compilers have problems with C++ exceptions in threaded code) and compile a MySQL version with support for all character sets.

If you suspect a memory overrun error, you can configure MySQL with `--with-debug=full`, which installs a memory allocation (`SAFEMALLOC`) checker. However, running with `SAFEMALLOC` is quite slow, so if you get performance problems you should start `mysqld` with the `--skip-safemalloc` option. This disables the memory overrun checks for each call to `malloc()` and `free()`.

If `mysqld` stops crashing when you compile it with `--with-debug`, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` variables above and not use `--with-debug`. If `mysqld` dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry is written to `stderr`, which `mysqld_safe` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See Section 1.6.1, “MySQL Mailing Lists”. If you believe that you have found a bug, please use the instructions at Section 1.7, “How to Report Bugs or Problems”.

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files. See also Section 21.3.1.2, “Creating Trace Files”.

21.3.1.2 Creating Trace Files

If the `mysqld` server doesn't start or if you can cause it to crash quickly, you can try to create a trace file to find the problem.

To do this, you must have a `mysqld` that has been compiled with debugging support. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it is compiled with support for trace files. (On Windows, the debugging server is named `mysqld-debug` rather than `mysqld`.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` flag to not start `mysqld` as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the `mysqld` server with `mysqladmin shutdown`.

The trace file can become **very large**! To generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can open a bug report and upload the trace file to the report, so that a MySQL developer can take a look at it. For instructions, see Section 1.7, “How to Report Bugs or Problems”.

The trace file is made with the **DBUG** package by Fred Fish. See Section 21.3.3, “The DBUG Package”.

21.3.1.3 Using pdb to create a Windows crashdump

Starting with MySQL 5.0.24 the Program Database files (extension `pdb`) are included in the Noinstall distribution of MySQL. These files provide information for debugging your MySQL installation in the event of a problem.

The PDB file contains more detailed information about `mysqld` and other tools that enables more detailed trace and dump files to be created. You can use these with Dr Watson, `WinDbg` and Visual Studio to debug `mysqld`.

For more information on PDB files, see Microsoft Knowledge Base Article 121366. For more information on the debugging options available, see Debugging Tools for Windows.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Dr Watson is installed with all Windows distributions, but if you have installed Windows development tools, Dr Watson may have been replaced with WinDbg, the debugger included with Visual Studio, or the debugging tools provided with Borland or Delphi.

To generate a crash file using Dr Watson, follow these steps:

1. Start Dr Watson by running `drwtsn32.exe` interactively using the `-i` option:

   ```
   C:\> drwtsn32 -i
   ```

2. Set the **Log File Path** to the directory where you want to store trace files.

3. Make sure **Dump All Thread Contexts** and **Append To Existing Log File**.

4. Uncheck **Dump Symbol Table**, **Visual Notification**, **Sound Notification** and **Create Crash Dump File**.

5. Set the **Number of Instructions** to a suitable value to capture enough calls in the stacktrace. A value of at 25 should be enough.

Note that the file generated can become very large.

21.3.1.4 Debugging mysqld under gdb

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case, you can only have one thread active at a time. It is best to upgrade to `gdb` 5.1 because thread debugging works much better with this version!

NPTL threads (the new thread library on Linux) may cause problems while running `mysqld` under `gdb`. Some symptoms are:

- `mysqld` hangs during startup (before it writes `ready for connections`).
- `mysqld` crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting `gdb`:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within `gdb`.

Use the `--gdb` option to `mysqld` to install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It is very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` doesn't free the memory for old threads. You can avoid this problem by starting `mysqld` with `thread_cache_size` set to a value equal to `max_connections` + 1. In most cases just using `--thread_cache_size=5'` helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a SIGSEGV signal, you can start `mysqld` with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

This documentation is for an older version. If you're

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   quit
```

See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”.

If you are using `gdb` 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with `gdb`, you should download gdb 5.x and try this instead. The new `gdb` version has very improved thread handling!

Here is an example how to debug mysqld:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the above output in a bug report, which you can file using the instructions in Section 1.7, “How to Report Bugs or Problems”.

If `mysqld` hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

21.3.1.5 Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See Section 5.4.1, “The Error Log”. To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to gcc. See Section 21.3.1.1, “Compiling MySQL for Debugging”.

A stack trace in the error log looks something like this:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

You can use the `resolve_stack_dump` utility to determine where `mysqld` died by using the following procedure:

1. Copy the preceding numbers to a file, for example `mysqld.stack`:

   ```
   0x9da402
   0x6648e9
   0x7f1a5af000f0
   0x7f1a5a10f0f2
   0x7412cb
   0x688354
   0x688494
   0x67a170
   0x67f0ad
   0x67fdf8
   0x6811b6
   0x66e05e
   ```

2. Make a symbol file for the `mysqld` server:

   ```
   shell> nm -n libexec/mysqld > /tmp/mysqld.sym
   ```

   If `mysqld` is not linked statically, use the following command instead:

   ```
   shell> nm -D -n libexec/mysqld > /tmp/mysqld.sym
   ```

   If you want to decode C++ symbols, use the `--demangle`, if available, to `nm`. If your version of `nm` does not have this option, you will need to use the `c++filt` command after the stack dump has been produced to demangle the C++ names.

3. Execute the following command:

   ```
   shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack
   ```

   If you were not able to include demangled C++ names in your symbol file, process the `resolve_stack_dump` output using `c++filt`:

   ```
   shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack | c++filt
   ```

   This prints out where `mysqld` died. If that does not help you find out why `mysqld` died, you should create a bug report and include the output from the preceding command with the bug report.

   However, in most cases it does not help us to have just a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, in most cases we need to know the statement that killed `mysqld` and preferably a test case so that we can repeat the problem! See Section 1.7, “How to Report Bugs or Problems”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

21.3.1.6 Using Server Logs to Find Causes of Errors in mysqld

Note that before starting `mysqld` with the general query log enabled, you should check all your tables with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

If `mysqld` dies or hangs, you should start `mysqld` with the general query log enabled. See Section 5.4.2, "The General Query Log". When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you use the default general query log file, the log is stored in the database directory as *`host_name`*`.log` In most cases it is the last query in the log file that killed `mysqld`, but if possible you should verify this by restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See Section 13.8.2, "EXPLAIN Syntax".

You can find the queries that take a long time to execute by starting `mysqld` with the slow query log enabled. See Section 5.4.4, "The Slow Query Log".

If you find the text `mysqld restarted` in the error log file (normally named `hostname.err`) you probably have found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see Chapter 5, *MySQL Server Administration*), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at http://lists.mysql.com/, which also has links to online list archives.

If you have started `mysqld` with `--myisam-recover`, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See Section 5.1.3, "Server Command Options".

It is not a good sign if `mysqld` did die unexpectedly, but in this case, you should not investigate the `Checking table...` messages, but instead try to find out why `mysqld` died.

21.3.1.7 Making a Test Case If You Experience Table Corruption

The following procedure applies to `MyISAM` tables. For information about steps to take when encountering `InnoDB` table corruption, see Section 1.7, "How to Report Bugs or Problems".

If you encounter corrupted `MyISAM` tables or if `mysqld` always fails after some update statements, you can test whether the issue is reproducible by doing the following:

1. Stop the MySQL daemon with `mysqladmin shutdown`.
2. Make a backup of the tables to guard against the very unlikely case that the repair does something bad.
3. Check all tables with `myisamchk -s database/*.MYI`. Repair any corrupted tables with `myisamchk -r database/`*`table`*`.MYI`.
4. Make a second backup of the tables.
5. Remove (or move away) any old log files from the MySQL data directory if you need more space.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

6. Start `mysqld` with the binary log enabled. If you want to find a statement that crashes `mysqld`, you should start the server with the general query log enabled as well. See Section 5.4.2, “The General Query Log”, and Section 5.4.3, “The Binary Log”.

7. When you have gotten a crashed table, stop the `mysqld` server.

8. Restore the backup.

9. Restart the `mysqld` server *without* the binary log enabled.

10. Re-execute the statements with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL database directory with the name `hostname-bin.NNNNNN`.

11. If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found a reproducible bug. FTP the tables and the binary log to our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you are a support customer, you can use the MySQL Customer Support Center (http://www.mysql.com/support/) to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

The preceding discussion applies only to RHEL4. The patch is unnecessary for RHEL5.

21.3.2 Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `--with-debug` or `--with-debug=full`. See Section 2.17.3, “MySQL Source-Configuration Options”.

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See Section 1.7, “How to Report Bugs or Problems”.

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

21.3.3 The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is doing. See Section 21.3.1.2, “Creating Trace Files”.

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about

This documentation is for an older version. If you're
This documentation is for an older version. If you're

programming with the DBUG package, see the DBUG manual in the `dbug` directory of MySQL source distributions. It's best to use a recent distribution to get the most updated DBUG manual.

The DBUG package can be used by invoking a program with the `--debug[=debug_options]` or `-#[debug_options]` option. If you specify the `--debug` or `-#` option without a *`debug_options`* value, most MySQL programs use a default value. The server default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows. The effect of this default is:

- `d`: Enable output for all debug macros
- `t`: Trace function calls and exits
- `i`: Add PID to output lines
- `o,/tmp/mysqld.trace`, `O,\mysqld.trace`: Set the debug output file.

Most client programs use a default *`debug_options`* value of `d:t:o,/tmp/`*`program_name`*`.trace`, regardless of platform.

Here are some example debug control strings as they might be specified on a shell command line:

```
--debug=d:t
--debug=d:f,main,subr1:F:L:t,20
--debug=d,input,output,files:n
--debug=d:t:i:O,\\mysqld.trace
```

The *`debug_options`* value is a sequence of colon-separated fields:

```
field_1:field_2:...:field_N
```

Each field within the value consists of a mandatory flag character, optionally preceded by a `+` or `-` character, and optionally followed by a comma-delimited list of modifiers:

```
[+|-]flag[,modifier,modifier,...,modifier]
```

The following table describes the permitted flag characters. Unrecognized flag characters are silently ignored.

Flag	Description
`d`	Enable output from DBUG_*`XXX`* macros for the current state. May be followed by a list of keywords, which enables output only for the DBUG macros with that keyword. An empty list of keywords enables output for all macros. In MySQL, common debug macro keywords to enable are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.
`D`	Delay after each debugger output line. The argument is the delay, in tenths of seconds, subject to machine capabilities. For example, `D,20` specifies a delay of two seconds.
`f`	Limit debugging, tracing, and profiling to the list of named functions. An empty list enables all functions. The appropriate `d` or `t` flags must still be given; this flag only limits their actions if they are enabled.
`F`	Identify the source file name for each line of debug or trace output.
`i`	Identify the process with the PID or thread ID for each line of debug or trace output.
`L`	Identify the source file line number for each line of debug or trace output.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`n`	Print the current function nesting depth for each line of debug or trace output.
`N`	Number each line of debug output.
`o`	Redirect the debugger output stream to the specified file. The default output is `stderr`.
`O`	Like `o`, but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
`p`	Limit debugger actions to specified processes. A process must be identified with the `DBUG_PROCESS` macro and match one in the list for debugger actions to occur.
`P`	Print the current process name for each line of debug or trace output.
`r`	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
`S`	Do function `_sanity(_file_,_line_)` at each debugged function until `_sanity()` returns something that differs from 0. (Mostly used with `safemalloc` to find memory leaks.)
`t`	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

The leading `+` or `-` character and trailing list of modifiers are used for flag characters such as `d` or `f` that can enable a debug operation for all applicable modifiers or just some of them:

- With no leading `+` or `-`, the flag value is set to exactly the modifier list as given.
- With a leading `+` or `-`, the modifiers in the list are added to or subtracted from the current modifier list.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 22 MySQL Enterprise Edition

Table of Contents

MySQL Enterprise Edition is a commercial product. Like MySQL Community Edition, MySQL Enterprise Edition includes MySQL Server, a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, crash-recovery, and row-level locking capabilities. In addition, MySQL Enterprise Edition includes the following components designed to provide monitoring and online backup, as well as improved security and scalability:

The following sections briefly discuss each of these components and indicate where to find more detailed information. To learn more about commercial products, see http://www.mysql.com/products/.

- MySQL Enterprise Monitor
- MySQL Enterprise Backup
- MySQL Enterprise Security
- MySQL Enterprise Encryption
- MySQL Enterprise Audit
- MySQL Enterprise Firewall
- MySQL Enterprise Thread Pool

22.1 MySQL Enterprise Monitor Overview

MySQL Enterprise Monitor is an enterprise monitoring system for MySQL that keeps an eye on your MySQL servers, notifies you of potential issues and problems, and advises you how to fix the issues. MySQL Enterprise Monitor can monitor all kinds of configurations, from a single MySQL server that is important to your business, all the way up to a huge farm of MySQL servers powering a busy web site.

The following discussion briefly summarizes the basic components that make up the MySQL Enterprise Monitor product. For more information, see the MySQL Enterprise Monitor manual, available at http://dev.mysql.com/doc/mysql-monitor/en/.

MySQL Enterprise Monitor components can be installed in various configurations depending on your database and network topology, to give you the best combination of reliable and responsive monitoring data, with minimal overhead on the database server machines. A typical MySQL Enterprise Monitor installation consists of:

- One or more MySQL servers to monitor. MySQL Enterprise Monitor can monitor both Community and Enterprise MySQL server releases.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- A MySQL Enterprise Monitor Agent for each monitored host.
- A single MySQL Enterprise Service Manager, which collates information from the agents and provides the user interface to the collected data.

MySQL Enterprise Monitor is designed to monitor one or more MySQL servers. The monitoring information is collected by using an agent, *MySQL Enterprise Monitor Agent*. The agent communicates with the hosts and MySQL servers that it monitors, collecting variables, status and health information, and sending this information to the MySQL Enterprise Service Manager.

The information collected by the agent about each MySQL server and host you are monitoring is sent to the *MySQL Enterprise Service Manager*. This server collates all of the information from the agents. As it collates the information sent by the agents, the MySQL Enterprise Service Manager continually tests the collected data, comparing the status of the server to reasonable values. When thresholds are reached, the server can trigger an event (including an alarm and notification) to highlight a potential issue, such as low memory, high CPU usage, or more complex conditions such insufficient buffer sizes and status information. We call each test, with its associated threshold value, a *rule*.

These rules, and the alarms and notifications, are each known as a *MySQL Enterprise Advisors*. Advisors form a critical part of the MySQL Enterprise Service Manager, as they provide warning information and troubleshooting advice about potential problems.

The MySQL Enterprise Service Manager includes a web server, and you interact with it through any web browser. This interface, the MySQL Enterprise Monitor User Interface, displays all of the information collected by the agents, and lets you view all of your servers and their current status as a group or individually. You control and configure all aspects of the service using the MySQL Enterprise Monitor User Interface.

The information supplied by the MySQL Enterprise Monitor Agent processes also includes statistical and query information, which you can view in the form of graphs. For example, you can view aspects such as server load, query numbers, or index usage information as a graph over time. The graph lets you pinpoint problems or potential issues on your server, and can help diagnose the impact from database or external problems (such as external system or network failure) by examining the data from a specific time interval.

The MySQL Enterprise Monitor Agent can also be configured to collect detailed information about the queries executed on your server, including the row counts and performance times for executing each query. You can correlate the detailed query data with the graphical information to identify which queries were executing when you experienced a particularly high load, index or other issue. The query data is supported by a system called Query Analyzer, and the data can be presented in different ways depending on your needs.

22.2 MySQL Enterprise Backup Overview

MySQL Enterprise Backup performs hot backup operations for MySQL databases. The product is architected for efficient and reliable backups of tables created by the InnoDB storage engine. For completeness, it can also back up tables from MyISAM and other storage engines.

The following discussion briefly summarizes MySQL Enterprise Backup. For more information, see the MySQL Enterprise Backup manual, available at http://dev.mysql.com/doc/mysql-enterprise-backup/en/.

Hot backups are performed while the database is running and applications are reading and writing to it. This type of backup does not block normal database operations, and it captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database “grows up” -- when the data is large enough that the backup takes significant time, and when your data is important enough to your business that you must capture every last change, without taking your application, web site, or web service offline.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. For efficient backup operations, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine.

22.3 MySQL Enterprise Security Overview

MySQL Enterprise Edition provides plugins that implement authentication using external services:

- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory. For more information, see The PAM Authentication Plugin.
- MySQL Enterprise Edition includes an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For more information, see The Windows Native Authentication Plugin.
- MySQL Enterprise Edition includes a keyring plugin that uses Oracle Key Vault for keyring backend storage. For more information, see The MySQL Keyring.

For other related Enterprise security features, see Section 22.4, “MySQL Enterprise Encryption Overview”.

22.4 MySQL Enterprise Encryption Overview

MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

- Implement added data protection using public-key asymmetric cryptography
- Create public and private keys and digital signatures
- Perform asymmetric encryption and decryption
- Use cryptographic hashing for digital signing and data verification and validation

For more information, see MySQL Enterprise Encryption Functions.

For other related Enterprise security features, see Section 22.3, “MySQL Enterprise Security Overview”.

22.5 MySQL Enterprise Audit Overview

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For more information, see MySQL Enterprise Audit Log Plugin.

22.6 MySQL Enterprise Firewall Overview

MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against whitelists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement whitelist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording or protecting mode, for training in the accepted statement patterns or protection against unacceptable statements.

For more information, see MySQL Enterprise Firewall.

22.7 MySQL Enterprise Thread Pool Overview

MySQL Enterprise Edition includes the MySQL Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. In MySQL Enterprise Edition, a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

For more information, see The Thread Pool Plugin.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Appendix A MySQL 5.0 Frequently Asked Questions

Table of Contents

A.1 MySQL 5.0 FAQ: General

A.1.1. Which version of MySQL is production-ready (GA)?

MySQL 5.7 and MySQL 5.6 are supported for production use.

MySQL 5.7 achieved General Availability (GA) status with MySQL 5.7.9, which was released for production use on 21 October 2015.

MySQL 5.6 achieved General Availability (GA) status with MySQL 5.6.10, which was released for production use on 5 February 2013.

MySQL 5.5 achieved General Availability (GA) status with MySQL 5.5.8, which was released for production use on 3 December 2010. The MySQL 5.5 series is no longer current, but still supported in production.

MySQL 5.1 achieved General Availability (GA) status with MySQL 5.1.30, which was released for production use on 14 November 2008. Active development for MySQL 5.1 has ended.

MySQL 5.0 achieved General Availability (GA) status with MySQL 5.0.15, which was released for production use on 19 October 2005. Active development for MySQL 5.0 has ended.

A.1.2. What is the state of development (non-GA) versions?

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL follows a milestone release model that introduces pre-production-quality features and stabilizes them to release quality (see http://dev.mysql.com/doc/mysql-development-cycle/en/index.html). This process then repeats, so releases cycle between pre-production and release quality status. Please check the change logs to identify the status of a given release.

MySQL 5.4 was a development series. Work on this series has ceased.

A successor to MySQL 5.7 is being actively developed using the milestone release methodology described above.

A.1.3. Can MySQL 5.0 do subqueries?

Yes. See Section 13.2.9, “Subquery Syntax”.

A.1.4. Can MySQL 5.0 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see Section 13.2.10, “UPDATE Syntax”; for that required to perform multiple-table deletes, see Section 13.2.2, “DELETE Syntax”.

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN ... END` block. See Section 18.3, “Using Triggers”.

A.1.5. Does MySQL 5.0 have a Query Cache? Does it work on Server, Instance or Database?

Yes. The query cache operates on the server level, caching complete result sets matched with the original query string. If an exactly identical query is made (which often happens, particularly in web applications), no parsing or execution is necessary; the result is sent directly from the cache. Various tuning options are available. See Section 8.10.3, “The MySQL Query Cache”.

A.1.6. Does MySQL 5.0 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 5.0 can also handle inserts in a multi-master replication setup. With the `auto_increment_increment` and `auto_increment_offset` system variables, you can set each server to generate auto-increment values that don't conflict with other servers. The `auto_increment_increment` value should be greater than the number of servers, and each server should have a unique offset.

A.1.7. Does MySQL 5.0 have a `NOW()` function with fractions of seconds?

No, but support was added in 5.6.4.

Also, MySQL does parse time strings with a fractional component. See Section 11.3.2, “The TIME Type”.

A.1.8. Does MySQL 5.0 work with multi-core processors?

Yes. MySQL is fully multi-threaded, and will make use of multiple CPUs, provided that the operating system supports them.

A.1.9. Why do I see multiple processes for `mysqld`?

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

A.1.10. Can MySQL 5.0 perform ACID transactions?

Yes. All current MySQL versions support transactions. The `InnoDB` storage engine offers full ACID transactions with row-level locking, multi-versioning, nonlocking repeatable reads, and all four SQL standard isolation levels.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

The NDB storage engine supports the READ COMMITTED transaction isolation level only.

A.2 MySQL 5.0 FAQ: Storage Engines

A.2.1. Where can I obtain complete documentation for MySQL storage engines?

See Chapter 14, *Storage Engines*. That chapter contains information about all MySQL storage engines except for the NDB storage engine used for MySQL Cluster; NDB is covered in Chapter 17, *MySQL Cluster*.

A.2.2. Are there any new storage engines in MySQL 5.0?

Yes. The FEDERATED storage engine, new in MySQL 5.0, allows the server to access tables on other (remote) servers. See Section 14.7, “The FEDERATED Storage Engine”.

A.2.3. Have any storage engines been removed in MySQL 5.0?

Yes. MySQL 5.0 no longer supports the ISAM storage engine. If you have any existing ISAM tables from previous versions of MySQL, you should convert these to MyISAM before upgrading to MySQL 5.0.

A.2.4. What are the unique benefits of the ARCHIVE storage engine?

The ARCHIVE storage engine is ideally suited for storing large amounts of data without indexes; it has a very small footprint, and performs selects using table scans. See Section 14.8, “The ARCHIVE Storage Engine”, for details.

A.2.5. Do the new features in MySQL 5.0 apply to all storage engines?

The general new features such as views, stored procedures, triggers, INFORMATION_SCHEMA, precision math (DECIMAL column type), and the BIT column type, apply to all storage engines. There are also additions and changes for specific storage engines.

A.3 MySQL 5.0 FAQ: Server SQL Mode

A.3.1. What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see Section 5.1.7, “Server SQL Modes”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.3.2. How many server SQL modes are there?

Each mode can be independently switched on and off. See Section 5.1.7, “Server SQL Modes”, for a complete list of available modes.

A.3.3. How do you determine the server SQL mode?

You can set the default SQL mode (for `mysqld` startup) with the `--sql-mode` option. Using the statement `SET [GLOBAL|SESSION] sql_mode='modes'`, you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement.

A.3.4. Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using `SET [GLOBAL|SESSION] sql_mode='modes'`.

A.3.5. Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes `TRADITIONAL`, `STRICT_TRANS_TABLES`, or `STRICT_ALL_TABLES` is enabled. Options can be combined, so you can add restrictions to a mode. See Section 5.1.7, “Server SQL Modes”, for more information.

A.3.6. Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However—if you do require it—strict mode can provide such validation.

A.3.7. What is the default server SQL mode when MySQL 5.0 is installed?

By default, no special modes are enabled. For information about all available modes and MySQL's default behavior, see Section 5.1.7, “Server SQL Modes”.

A.4 MySQL 5.0 FAQ: Stored Procedures and Functions

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.4.1. Does MySQL 5.0 support stored procedures and functions?

Yes. MySQL 5.0 supports two types of stored routines—stored procedures and stored functions.

A.4.2. Where can I find documentation for MySQL stored procedures and stored functions?

See Section 18.2, “Using Stored Routines (Procedures and Functions)”.

A.4.3. Is there a discussion forum for MySQL stored procedures?

Yes. See http://forums.mysql.com/list.php?98.

A.4.4. Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books—such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer—which give a comprehensive overview of the standard, including coverage of stored procedures.

A.4.5. How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see Section 19.8, “The INFORMATION_SCHEMA ROUTINES Table”).

A.4.6. Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named *`dbname`*, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see Section 19.8, “The INFORMATION_SCHEMA ROUTINES Table”.

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See Section 13.7.5.8, “SHOW CREATE PROCEDURE Syntax”, for more information.

A.4.7. Where are stored procedures stored?

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In the `proc` table of the `mysql` system database. However, you should not access the tables in the system database directly. Instead, use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See Section 13.7.5.8, "SHOW CREATE PROCEDURE Syntax", for more information about these statements.

You can also query the `ROUTINES` table in the `INFORMATION_SCHEMA` database—see Section 19.8, "The INFORMATION_SCHEMA ROUTINES Table", for information about this table.

A.4.8. Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 5.0.

A.4.9. Can a stored procedure call another stored procedure?

Yes.

A.4.10. Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to activate.

A.4.11. Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

A.4.12. Do stored procedures have a statement for raising application errors?

Not in MySQL 5.0. The SQL standard `SIGNAL` and `RESIGNAL` statements are implemented in MySQL 5.5.

A.4.13. Do stored procedures provide exception handling?

MySQL implements `HANDLER` definitions according to the SQL standard. See Section 13.6.7.2, "DECLARE ... HANDLER Syntax", for details.

A.4.14. Can MySQL 5.0 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that—for instance—in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.15. Is `WITH RECOMPILE` supported for stored procedures?

Not in MySQL 5.0.

A.4.16. Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 5.0.

A.4.17. Can I pass an array as input to a stored procedure?

Not in MySQL 5.0.

A.4.18. Can I pass a cursor as an `IN` parameter to a stored procedure?

In MySQL 5.0, cursors are available inside stored procedures only.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.4.19. Can I return a cursor as an `OUT` parameter from a stored procedure?

In MySQL 5.0, cursors are available inside stored procedures only. However, if you do not open a cursor on a `SELECT`, the result will be sent directly to the client. You can also `SELECT INTO` variables. See Section 13.2.8, “SELECT Syntax”.

A.4.20. Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You will need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that—for instance—in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.21. Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

A.4.22. Do MySQL 5.0 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in Section 18.6, “Binary Logging of Stored Programs”.

A.4.23. Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. `ALTER` and `DROP` statements for stored procedures and functions are also replicated.

A.4.24. How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

A.4.25. Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the `SUPER` privilege.
2. Alternatively, a DBA can set the `log_bin_trust_function_creators` system variable to 1, which enables anyone with the standard `CREATE ROUTINE` privilege to create stored functions.

A.4.26. What limitations exist for replicating stored procedure and function actions?

Nondeterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Declaring stored functions to be `DETERMINISTIC` or setting the `log_bin_trust_function_creators` system variable to 0 will not allow random-valued operations to be invoked.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, nontransactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the `IGNORE` keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

A.4.27 Do the preceding limitations affect MySQL's ability to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

A.4.28 What is being done to correct the aforementioned limitations?

MySQL 5.1 implements *row-based replication*, which resolves the limitations mentioned earlier.

We do not plan to backport row-based replication to MySQL 5.0. For additional information, see Replication Formats, in the *MySQL 5.1 Manual*.

A.5 MySQL 5.0 FAQ: Triggers

A.5.1. Where can I find the documentation for MySQL 5.0 triggers?

See Section 18.3, “Using Triggers”.

A.5.2. Is there a discussion forum for MySQL Triggers?

Yes. It is available at http://forums.mysql.com/list.php?99.

A.5.3. Does MySQL 5.0 have statement-level or row-level triggers?

In MySQL 5.0, all triggers are `FOR EACH ROW`—that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 5.0 does not support triggers using `FOR EACH STATEMENT`.

A.5.4. Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some `TIMESTAMP` columns, as well as for columns which are defined using `AUTO_INCREMENT`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.5.5. How are triggers managed in MySQL?

In MySQL 5.0, triggers can be created using the `CREATE TRIGGER` statement, and dropped using `DROP TRIGGER`. See Section 13.1.11, “CREATE TRIGGER Syntax”, and Section 13.1.18, “DROP TRIGGER Syntax”, for more about these statements.

Information about triggers can be obtained by querying the `INFORMATION_SCHEMA.TRIGGERS` table. See Section 19.15, “The INFORMATION_SCHEMA TRIGGERS Table”.

A.5.6. Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database `dbname` using a query on the `INFORMATION_SCHEMA.TRIGGERS` table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
    FROM INFORMATION_SCHEMA.TRIGGERS
    WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see Section 19.15, “The INFORMATION_SCHEMA TRIGGERS Table”.

You can also use the `SHOW TRIGGERS` statement, which is specific to MySQL. See Section 13.7.5.35, “SHOW TRIGGERS Syntax”.

A.5.7. Where are triggers stored?

Triggers for a table are currently stored in `.TRG` files, with one such file one per table.

A.5.8. Can a trigger call a stored procedure?

Yes.

A.5.9. Can triggers access tables?

A trigger can access both old and new data in its own table. A trigger can also affect other tables, but it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger. (Before MySQL 5.0.10, a trigger cannot modify other tables.)

A.5.10. Can a table have multiple triggers with the same trigger event and action time?

In MySQL 5.0, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. This limitation is lifted in MySQL 5.7.

A.5.11. Can triggers call an external application through a UDF?

Yes. For example, a trigger could invoke the `sys_exec()` UDF.

A.5.12. Is it possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the `FEDERATED` storage engine. (See Section 14.7, “The FEDERATED Storage Engine”).

A.5.13. Do triggers work with replication?

Triggers and replication in MySQL 5.0 work in the same way as in most other database systems: Actions carried out through triggers on a master are not replicated to a slave server. Instead,

This documentation is for an older version. If you're

triggers that exist on tables that reside on a MySQL master server need to be created on the corresponding tables on any MySQL slave servers so that the triggers activate on the slaves as well as the master.

For more information, see Section 16.4.1.27, “Replication and Triggers”.

A.5.14. How are actions carried out through triggers on a master replicated to a slave?

First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table `EMP` that has an `AFTER` insert trigger, which exists on a master MySQL server. The same `EMP` table and `AFTER` insert trigger exist on the slave server as well. The replication flow would be:

1. An `INSERT` statement is made to `EMP`.
2. The `AFTER` trigger on `EMP` activates.
3. The `INSERT` statement is written to the binary log.
4. The replication slave picks up the `INSERT` statement to `EMP` and executes it.
5. The `AFTER` trigger on `EMP` that exists on the slave activates.

For more information, see Section 16.4.1.27, “Replication and Triggers”.

A.6 MySQL 5.0 FAQ: Views

A.6.1. Where can I find documentation covering MySQL Views?

See Section 18.4, “Using Views”.

A.6.2. Is there a discussion forum for MySQL Views?

Yes. See http://forums.mysql.com/list.php?100

A.6.3. What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the `CHECK TABLE` statement. (See Section 13.7.2.3, “CHECK TABLE Syntax”.)

A.6.4. Does MySQL 5.0 have table snapshots?

No.

A.6.5. Does MySQL 5.0 have materialized views?

No.

A.6.6. Can you insert into views that are based on joins?

This documentation is for an older version. If you're
This documentation is for an older version. If you're

It is possible, provided that your `INSERT` statement has a column list that makes it clear there is only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

A.7 MySQL 5.0 FAQ: INFORMATION_SCHEMA

A.7.1. Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See Chapter 19, *INFORMATION_SCHEMA Tables*

A.7.2. Is there a discussion forum for `INFORMATION_SCHEMA`?

See http://forums.mysql.com/list.php?101.

A.7.3. Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available—such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer—which give a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

A.7.4. What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. MySQL's implementation is more similar to those found in DB2 and SQL Server, which also support `INFORMATION_SCHEMA` as defined in the SQL standard.

A.7.5. Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *we cannot support bugs or other issues which result from modifying `INFORMATION_SCHEMA` tables or data*.

A.8 MySQL 5.0 FAQ: Migration

A.8.1. Where can I find information on how to migrate from MySQL 4.1 to MySQL 5.0?

For detailed upgrade information, see Section 2.19.1, “Upgrading MySQL”. Do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but it will you save time and trouble—if you encounter problems during the upgrade, their origin will be easier to identify, either by you or—if you have a MySQL Enterprise subscription—by MySQL support.

A.8.2. How has storage engine (table type) support changed in MySQL 5.0 from previous versions?

Storage engine support has changed as follows:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Support for `ISAM` tables was removed in MySQL 5.0 and you should now use the `MyISAM` storage engine in place of `ISAM`. To convert a table *`tblname`* from `ISAM` to `MyISAM`, simply issue a statement such as this one:

  ```
  ALTER TABLE tblname ENGINE=MYISAM;
  ```

- Internal `RAID` for `MyISAM` tables was also removed in MySQL 5.0. This was formerly used to allow large tables in file systems that did not support file sizes greater than 2GB. All modern file systems allow for larger tables; in addition, there are now other solutions such as `MERGE` tables and views.

- The `VARCHAR` column type now retains trailing spaces in all storage engines.

- `MEMORY` tables (formerly known as `HEAP` tables) can also contain `VARCHAR` columns.

A.9 MySQL 5.0 FAQ: Security

A.9.1. Where can I find documentation that addresses security issues for MySQL?

The best place to start is Chapter 6, *Security*.

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- Section 6.1.1, “Security Guidelines”.
- Section 6.1.3, “Making MySQL Secure Against Attackers”.
- Section B.5.3.2, “How to Reset the Root Password”.
- Section 6.1.5, “How to Run MySQL as a Normal User”.
- Section 21.2.2.6, “UDF Security Precautions”.
- Section 6.1.4, “Security-Related mysqld Options and Variables”.
- Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.
- Section 2.18, “Postinstallation Setup and Testing”.
- Section 6.3.6, “Using Secure Connections”.

A.9.2. Does MySQL 5.0 have native support for SSL?

Most 5.0 binaries have support for SSL connections between the client and server. See Section 6.3.6, “Using Secure Connections”.

You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see Section 6.3.8, “Connecting to MySQL Remotely from Windows with SSH”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.9.3. Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 5.0 binaries have SSL enabled for client/server connections that are secured, authenticated, or both. See Section 6.3.6, “Using Secure Connections”.

A.9.4. Does MySQL 5.0 have built-in authentication against LDAP directories?

No.

As of MySQL 5.5.16, the Enterprise edition includes a PAM Authentication Plugin that supports authentication against an LDAP directory.

A.9.5. Does MySQL 5.0 include support for Roles Based Access Control (RBAC)?

Not at this time.

A.10 MySQL 5.0 FAQ: MySQL Cluster

In the following section, we answer questions that are frequently asked about MySQL Cluster and the `NDBCLUSTER` storage engine.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.10.1 Which versions of the MySQL software support Cluster? Do I have to compile from source?

MySQL Cluster is supported in all server binaries in the 5.0 release series for operating systems on which MySQL Cluster is available. See Section 4.3.1, “`mysqld` — The MySQL Server”. You can determine whether your server has `NDB` support using either of the statements `SHOW VARIABLES LIKE 'have_%'` or `SHOW ENGINES`.

Note

Linux users: `NDB` is *not* included in the standard MySQL server RPMs. Beginning with MySQL 5.0.4, there are separate RPM packages for the `NDB` storage engine and accompanying management and other tools; see the `NDB` RPM Downloads section of the MySQL 5.0 Downloads page for these. (Prior to 5.0.4, you had to use the `-max` binaries supplied as `.tar.gz` archives. This is still possible, but is not required, so you can use your Linux distribution's RPM manager if you prefer.)

You can also obtain `NDB` support by compiling MySQL from source, but it is not necessary to do so simply to use MySQL Cluster. To download the latest binary, RPM, or source distribution in the MySQL 5.0 series, visit http://dev.mysql.com/downloads/mysql/5.0.html.

You should use MySQL Cluster NDB 7.3 or MySQL Cluster NDB 7.4 for new deployments; if you are currently using an older version of MySQL Cluster, you should upgrade to one of these versions as soon as possible. For an overview of improvements made in MySQL Cluster NDB 7.3 and 7.4, see What is New in MySQL Cluster NDB 7.3, and What is New in MySQL Cluster NDB 7.4, respectively.

A.10.2 What do “NDB” and “NDBCLUSTER” mean?

“NDB” stands for “**N**etwork **D**ata**b**ase”. `NDB` and `NDBCLUSTER` are both names for the storage engine that enables clustering support with MySQL. `NDB` is preferred, but either name is correct.

A.10.3 What is the difference between using MySQL Cluster versus using MySQL Replication?

In traditional MySQL replication, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as `InnoDB` is being used, a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is *asynchronous*, MySQL Cluster is *synchronous*.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

We have implemented (asynchronous) replication for Cluster in MySQL 5.1 and later. *MySQL Cluster Replication* (also sometimes known as “geo-replication”) includes the capability to replicate both between two MySQL Clusters, and from a MySQL Cluster to a non-Cluster MySQL server. However, we do *not* plan to backport this functionality to MySQL 5.0. See MySQL Cluster Replication.

A.10.4. Do I need any special networking to run MySQL Cluster? How do computers in a cluster communicate?

MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting using TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for MySQL Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.

A.10.5. How many computers do I need to run a MySQL Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum *recommended* number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

A.10.6. What do the different computers do in a MySQL Cluster?

A MySQL Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node.** This node provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application `ndb_mgmd`; the management client used to control MySQL Cluster is `ndb_mgm`. See Section 17.4.2, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”, and Section 17.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”, for information about these programs.

- **Data node.** This type of node stores and replicates data. Data node functionality is handled by instances of the `NDB` data node process `ndbd`. For more information, see Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”.

- **SQL node.** This is simply an instance of MySQL Server (`mysqld`) that is built with support for the `NDBCLUSTER` storage engine and started with the `--ndb-cluster` option to enable the engine and the `--ndb-connectstring` option to enable it to connect to a MySQL Cluster management server. For more about these options, see mysqld Command Options for MySQL Cluster.

Note

An *API node* is any application that makes direct use of Cluster data nodes for data storage and retrieval. An SQL node can thus be considered a type of API node that uses a MySQL Server to provide an SQL interface

This documentation is for an older version. If you're
This documentation is for an older version. If you're

to the Cluster. You can write such applications (that do not depend on a MySQL Server) using the NDB API, which supplies a direct, object-oriented transaction and scanning interface to MySQL Cluster data; see MySQL Cluster API Overview: The NDB API, for more information.

A.10.7. With which operating systems can I use MySQL Cluster?

MySQL Cluster is supported on most Unix-like operating systems. Beginning with MySQL Cluster NDB 7.1.3, MySQL Cluster is also supported in production on Microsoft Windows operating systems.

Important

We do not intend to provide any level of support on Windows for MySQL Cluster in MySQL 5.0; you must use MySQL Cluster NDB 7.1.3 or later to obtain GA-level support for MySQL Cluster in a Windows environment. See What is New in MySQL Cluster NDB 7.1, for more information.

For more detailed information concerning the level of support which is offered for MySQL Cluster on various operating system versions, operating system distributions, and hardware platforms, please refer to http://www.mysql.com/support/supportedplatforms/cluster.html.

A.10.8. What are the hardware requirements for running MySQL Cluster?

MySQL Cluster should run on any platform for which `NDB`-enabled binaries are available. For data nodes and API nodes, faster CPUs and more memory are likely to improve performance, and 64-bit CPUs are likely to be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). For a computer which is used only for running the MySQL Cluster management server, the requirements are minimal; a common desktop PC (or the equivalent) is generally sufficient for this task. Nodes can communicate through the standard TCP/IP network and hardware. They can also use the high-speed SCI protocol; however, special networking hardware and software are required to use SCI (see Section 17.3.4, “Using High-Speed Interconnects with MySQL Cluster”).

A.10.9. How much RAM do I need to use MySQL Cluster? Is it possible to use disk memory at all?

In MySQL 5.0, Cluster is in-memory only. This means that all table data (including indexes) is stored in RAM. Therefore, if your data takes up 1 GB of space and you want to replicate it once in the cluster, you need 2 GB of memory to do so (1 GB per replica). This is in addition to the memory required by the operating system and any applications running on the cluster computers.

If a data node's memory usage exceeds what is available in RAM, then the system will attempt to use swap space up to the limit set for `DataMemory`. However, this will at best result in severely degraded performance, and may cause the node to be dropped due to slow response time (missed heartbeats). We do not recommend on relying on disk swapping in a production environment for this reason. In any case, once the `DataMemory` limit is reached, any operations requiring additional memory (such as inserts) will fail.

We have implemented disk data storage for MySQL Cluster in MySQL 5.1 and later but we have no plans to add this capability in MySQL 5.0. See MySQL Cluster Disk Data Tables, for more information.

You can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

This documentation is for an older version. If you're

```
(SizeofDatabase × NumberOfReplicas × 1.1 ) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see Section 11.7, “Data Type Storage Requirements”, for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an `NDBCLUSTER` table requires 21−25 bytes per record. These indexes use `IndexMemory`.
- Each ordered index requires 10 bytes storage per record, using `DataMemory`.
- Creating a primary key or unique index also creates an ordered index, unless this index is created with `USING HASH`. In other words:
  - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
  - However, if the primary key or unique index is created with `USING HASH`, then it requires only 21 to 25 bytes per record.

Creating MySQL Cluster tables with `USING HASH` for all primary keys and unique indexes will generally cause table updates to run more quickly—in some cases by a much as 20 to 30 percent faster than updates on tables where `USING HASH` was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available in recent MySQL 5.0 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. For more information, see Section 17.4.18, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”.

It is especially important to keep in mind that *every MySQL Cluster table must have a primary key*. The `NDB` storage engine creates a primary key automatically if none is defined; this primary key is created without `USING HASH`.

There is no easy way to determine exactly how much memory is being used for storage of MySQL Cluster indexes at any given time; however, warnings are written to the cluster log when 80% of available `DataMemory` or `IndexMemory` is in use, and again when usage reaches 85%, 90%, and so on.

A.10.10 What file systems can I use with MySQL Cluster? What about network file systems or network shares?

Generally, any file system that is native to the host operating system should work well with MySQL Cluster. If you find that a given file system works particularly well (or not so especially well) with MySQL Cluster, we invite you to discuss your findings in the MySQL Cluster Forums.

We do not test MySQL Cluster with `FAT` or `VFAT` file systems on Linux. Because of this, and due to the fact that these are not very useful for any purpose other than sharing disk partitions between Linux and Windows operating systems on multi-boot computers, we do not recommend their use with MySQL Cluster.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MySQL Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network file systems is not supported for MySQL Cluster. This also applies to shared storage devices such as SANs.

A.10.11. Can I run MySQL Cluster nodes inside virtual machines (such as those created by VMWare, Parallels, or Xen)?

This is possible but not recommended for a production environment with MySQL Cluster versions prior to MySQL Cluster NDB 7.2.

For deployment in virtualized environments, you should use MySQL Cluster NDB 7.2 or later.

A.10.12. I am trying to populate a MySQL Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: The table 'my_cluster_table' is full
```

Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the* `NDB` *storage engine and automatically created in the event that the table definition does not include the definition of a primary key*.

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. For example, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote at most 1GB to MySQL Cluster data and indexes.

In some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. You can force `NDB` to create extra partitions for MySQL Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient.

For similar reasons, you can also sometimes encounter problems with data node restarts on nodes that are heavily loaded with data. In MySQL Cluster NDB 7.1 and later, the addition of the `MinFreePct` parameter helps with this issue by reserving a portion (5% by default) of `DataMemory` and `IndexMemory` for use in restarts. This reserved memory is not available for storing `NDB` tables or data.

A.10.13. MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is *very* unlikely that a cluster would perform reliably under such conditions, as MySQL Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet—preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in a MySQL Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you

This documentation is for an older version. If you're
This documentation is for an older version. If you're

should take the same precautions as you would with any other instance of the MySQL server.) For more information, see Section 17.5.10, “MySQL Cluster Security Issues”.

A.10.14. Do I have to learn a new programming or query language to use MySQL Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL statements are required for the following operations:

- Creating, altering, and dropping tables
- Inserting, updating, and deleting table data
- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up a MySQL Cluster—see Section 17.3.3, “MySQL Cluster Configuration Files”, for information about these.

A few simple commands are used in the MySQL Cluster management client (`ndb_mgm`) for tasks such as starting and stopping cluster nodes. See Section 17.5.2, “Commands in the MySQL Cluster Management Client”.

A.10.15. What programming languages and APIs are supported by MySQL Cluster?

MySQL Cluster 5.0 supports the same programming APIs and languages as the standard MySQL Server, including ODBC, .Net, the MySQL C API, and numerous drivers for popular scripting languages such as PHP, Perl, and Python. MySQL Cluster applications written using these APIs behave similarly to other MySQL applications; they transmit SQL statements to a MySQL Server (in the case of MySQL Cluster, an SQL node), and receive responses containing rows of data. For more information about these APIs, see Chapter 20, *Connectors and APIs*.

A.10.16. Does MySQL Cluster include any management tools?

MySQL Cluster includes a command line client for performing basic management functions. See Section 17.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”, and Section 17.5.2, “Commands in the MySQL Cluster Management Client”.

A.10.17. How do I find out what an error or warning message means when using MySQL Cluster?

There are two ways in which this can be done:

- From within the `mysql` client, use `SHOW ERRORS` or `SHOW WARNINGS` immediately upon being notified of the error or warning condition.
- From a system shell prompt, use `perror --ndb error_code`.

A.10.18. Is MySQL Cluster transaction-safe? What isolation levels are supported?

Yes. For tables created with the `NDB` storage engine, transactions are supported. Currently, MySQL Cluster supports only the `READ COMMITTED` transaction isolation level.

A.10.19. What storage engines are supported by MySQL Cluster?

Clustering with MySQL is supported only by the `NDB` storage engine. That is, in order for a table to be shared between nodes in a MySQL Cluster, the table must be created using `ENGINE=NDB` (or the equivalent option `ENGINE=NDBCLUSTER`).

It is possible to create tables using other storage engines (such as `InnoDB` or `MyISAM`) on a MySQL server being used with a MySQL Cluster, but since these tables do not use `NDB`, they do not

This documentation is for an older version. If you're
This documentation is for an older version. If you're

participate in clustering; each such table is strictly local to the individual MySQL server instance on which it is created.

A.10.20: In the event of a catastrophic failure—say, for instance, the whole city loses power *and* my UPS fails—would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

A.10.21: Is it possible to use `FULLTEXT` indexes with MySQL Cluster?

`FULLTEXT` indexing is currently supported only by the `MyISAM` storage engine. See Section 12.9, "Full-Text Search Functions", for more information.

A.10.22: Can I run multiple nodes on a single computer?

It is possible but not advisable. One of the chief reasons to run a cluster is to provide redundancy. To obtain the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. Given that MySQL Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 300 MHz Pentium or equivalent CPU and sufficient RAM for the operating system, plus a small amount of overhead for the `ndb_mgmd` and `ndb_mgm` processes.

It is acceptable to run multiple cluster data nodes on a single host for learning about MySQL Cluster, or for testing purposes; however, this is not generally supported for production use.

A.10.23: Can I add data nodes to a MySQL Cluster without restarting it?

Not in MySQL 5.0. While a rolling restart is all that is required for adding new management or API nodes to a MySQL Cluster (see Section 17.5.5, "Performing a Rolling Restart of a MySQL Cluster"), adding data nodes is more complex, and requires the following steps:

1. Make a complete backup of all Cluster data.
2. Completely shut down the cluster and all cluster node processes.
3. Restart the cluster, using the `--initial` startup option for all instances of `ndbd`.

Warning

Never use the `--initial` when starting `ndbd` except when necessary to clear the data node file system. See Section 17.4.1, "`ndbd` — The MySQL Cluster Data Node Daemon", for information about when this is required.

4. Restore all cluster data from the backup.

Note

Beginning with MySQL Cluster NDB 6.4, it is possible to add new data nodes to a running MySQL Cluster without taking it offline. For more information, see Adding MySQL Cluster Data Nodes Online. However, we do not plan to add this capability in MySQL 5.0.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

A.10.24. Are there any limitations that I should be aware of when using MySQL Cluster?

Limitations on `NDB` tables in MySQL 5.0 include the following:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- `FULLTEXT` indexes are not supported.
- Index prefixes are not supported. Only complete columns may be indexed.
- As of MySQL 5.0.16, MySQL Cluster supports spatial data types. However, spatial indexes are not supported. See Section 11.5, “Extensions for Spatial Data”.
- Only complete rollbacks for transactions are supported. Partial rollbacks and rollbacks to savepoints are not supported. A failed insert due to a duplicate key or similar error causes a transaction to abort; when this occurs, you must issue an explicit `ROLLBACK` and retry the transaction.
- The maximum number of attributes allowed per table is 128, and attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- The maximum size for a table row is 8 kilobytes, not counting `BLOB` values.

  There is no set limit for the number of rows per `NDB` table. Limits on table size depend on a number of factors, in particular on the amount of RAM available to each data node.
- The `NDB` engine does not support foreign key constraints. As with `MyISAM` tables, if these are specified in a `CREATE TABLE` or `ALTER TABLE` statement, they are ignored.

For a complete listing of limitations in MySQL Cluster, see Section 17.1.5, “Known Limitations of MySQL Cluster”. See also Section 17.1.5.10, “Previous MySQL Cluster Issues Resolved in MySQL 5.0”.

A.10.25. Does MySQL Cluster support foreign keys?

MySQL Cluster 5.0 does not support foreign key constraints, and ignores foreign keys in `CREATE TABLE` statements (similarly to how `MyISAM` treats foreign key syntax).

Foreign key support comparable to that found in the `InnoDB` storage engine is provided by `NDB` beginning with MySQL Cluster NDB 7.3. Applications requiring foreign support should use MySQL Cluster NDB 7.3 or later.

A.10.26. How do I import an existing MySQL database into a MySQL Cluster?

You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ, the only other special requirement is that any tables to be included in the cluster must use the `NDB` storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables that use other storage engines to `NDBCLUSTER` using one or more `ALTER TABLE` statement. However, the definition of the table must be compatible with the `NDBCLUSTER` storage engine prior to making the conversion. In MySQL 5.0, an additional workaround is also required; see Section 17.1.5, “Known Limitations of MySQL Cluster”, for details.

A.10.27. How do MySQL Cluster nodes communicate with one another?

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Cluster nodes can communicate through any of three different transport mechanisms: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See Section 17.3.4, “Using High-Speed Interconnects with MySQL Cluster”, for more about using SCI as a transport mechanism for MySQL Cluster.

A.10.28 What is an *arbitrator*?

If one or more data nodes in a cluster fail, it is possible that not all cluster data nodes will be able to “see” one another. In fact, it is possible that two sets of data nodes might become isolated from one another in a network partitioning, also known as a “split-brain” scenario. This type of situation is undesirable because each set of data nodes tries to behave as though it is the entire cluster. An arbitrator is required to decide between the competing sets of data nodes.

When all data nodes in at least one node group are alive, network partitioning is not an issue, because no single subset of the cluster can form a functional cluster on its own. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the `ArbitrationRank` configuration parameter for MySQL Server and management server nodes. For more information about this parameter, see Section 17.3.3.4, “Defining a MySQL Cluster Management Server”.

The role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

A.10.29 What data types are supported by MySQL Cluster?

In MySQL 5.0;, MySQL Cluster supports all of the usual MySQL data types, including (beginning with MySQL 5.0.16) those associated with MySQL's spatial extensions; however, the `NDB` storage engine does not support spatial indexes. (Spatial indexes are supported only by `MyISAM`; see Section 11.5, “Extensions for Spatial Data”, for more information.) In addition, there are some differences with regard to indexes when used with `NDB` tables.

Note

In MySQL 5.0, MySQL Cluster tables (that is, tables created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each record containing a `VARCHAR(255)` column will require space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein. This issue is fixed in MySQL 5.1 and later; however, we do not plan to backport this functionality to MySQL 5.0.

See Section 17.1.5, “Known Limitations of MySQL Cluster”, for more information about these issues.

A.10.30 How do I start and stop MySQL Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node, using the `ndb_mgmd` command.

This documentation is for an older version. If you're

You must include the `-f` or `--config-file` option to tell the management node where its configuration file can be found.

2. Start each data node with the `ndbd` command.

   Each data node must be started with the `-c` or `--ndb-connectstring` option so that the data node knows how to connect to the management server.

3. Start each MySQL Server (SQL node) using your preferred startup script, such as `mysqld_safe`.

   Each MySQL Server must be started with the `--ndbcluster` and `--ndb-connectstring` options. These options cause mysqld to enable `NDBCLUSTER` storage engine support and how to connect to the management server.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine—a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the `NDB` management client `ndb_mgm` on the machine housing the management node and issuing the `SHOW` or `ALL STATUS` command.

To shut down a running cluster, issue the command `SHUTDOWN` in the management client. Alternatively, you may enter the following command in a system shell:

```
shell> ndb_mgm -e "SHUTDOWN"
```

(The quotation marks in this example are optional, since there are no spaces in the command string following the `-e` option; in addition, the `SHUTDOWN` command, like other management client commands, is not case-sensitive.)

Either of these commands causes the `ndb_mgm`, `ndb_mgm`, and any `ndbd` processes to terminate gracefully. MySQL servers running as SQL nodes can be stopped using `mysqladmin shutdown`.

For more information, see Section 17.5.2, “Commands in the MySQL Cluster Management Client”, and Section 17.2.5, “Safe Shutdown and Restart of MySQL Cluster”.

A.10.31. What happens to MySQL Cluster data when the MySQL Cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

A.10.32. Is it a good idea to have more than one management node for a MySQL Cluster?

It can be helpful as a fail-safe. Only one management node controls the cluster at any given time, but it is possible to configure one management node as primary, and one or more additional management nodes to take over in the event that the primary management node fails.

See Section 17.3.3, “MySQL Cluster Configuration Files”, for information on how to configure MySQL Cluster management nodes.

A.10.33. Can I mix different kinds of hardware and operating systems in one MySQL Cluster?

Yes, as long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian).

This documentation is for an older version. If you're

It is also possible to use software from different MySQL Cluster releases on different nodes. However, we support this only as part of a rolling upgrade procedure (see Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”).

A.10.34. Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port. However, in MySQL 5.0, running more than one cluster node of a given type per machine is generally not encouraged or supported for production use.

We also advise against running data nodes and SQL nodes together on the same host, since the `ndbd` and `mysqld` processes may compete for memory.

A.10.35. Can I use host names with MySQL Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, you should use fixed (numeric) IP addresses, since making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional potential points of failure.

A.10.36. How do I handle MySQL users in a MySQL Cluster having multiple MySQL servers?

MySQL user accounts and privileges are *not* automatically propagated between different MySQL servers accessing the same MySQL Cluster. Therefore, you must make sure that these are copied between the SQL nodes yourself. You can do this manually, or automate the task with scripts.

Warning

Do not attempt to work around this issue by converting the MySQL system tables to use the `NDBCLUSTER` storage engine. Only the `MyISAM` storage engine is supported for these tables.

A.10.37. How do I continue to send queries in the event that one of the SQL nodes fails?

MySQL Cluster does not provide any sort of automatic failover between SQL nodes. Your application must be prepared to handle the loss of SQL nodes and to fail over between them.

A.10.38. How do I back up and restore a MySQL Cluster?

You can use the NDB native backup and restore functionality in the MySQL Cluster management client and the `ndb_restore` program. See Section 17.5.3, “Online Backup of MySQL Cluster”, and Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”.

You can also use the traditional functionality provided for this purpose in `mysqldump` and the MySQL server. See Section 4.5.4, “`mysqldump` — A Database Backup Program”, for more information.

A.10.39. What is an “angel process”?

This process monitors and, if necessary, attempts to restart the data node process. If you check the list of active processes on your system after starting `ndbd`, you can see that there are actually 2 processes running by that name, as shown here (we omit the output from `ndb_mgmd` and `ndbd` for brevity):

```
shell> ./ndb_mgmd
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> ps aux | grep ndb
me      23002  0.0  0.0 122948  3104 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23025  0.0  0.0   5284   820 pts/2    S+   14:14   0:00 grep ndb

shell> ./ndbd -c 127.0.0.1 --initial

shell> ps aux | grep ndb
me      23002  0.0  0.0 123080  3356 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23096  0.0  0.0  35876  2036 ?        Ss   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23097  1.0  2.4 524116 91096 ?        Sl   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23168  0.0  0.0   5284   812 pts/2    R+   14:15   0:00 grep ndb
```

The `ndbd` process showing 0 memory and CPU usage is the angel process. It actually does use a very small amount of each, of course. It simply checks to see if the main `ndbd` process (the primary data node process that actually handles the data) is running. If permitted to do so (for example, if the `StopOnError` configuration parameter is set to false—see Section 17.3.2.1, “MySQL Cluster Data Node Configuration Parameters”), the angel process tries to restart the primary data node process.

A.11 MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

A.11.1 What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the `gb18030` character set was not supported prior to MySQL 5.7.4. However, since the name of the applicable language appears in the `DESCRIPTION` column for every entry in the `INFORMATION_SCHEMA.CHARACTER_SETS` table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
    -> FROM INFORMATION_SCHEMA.CHARACTER_SETS
    -> WHERE DESCRIPTION LIKE '%Chin%'
    -> OR DESCRIPTION LIKE '%Japanese%'
    -> OR DESCRIPTION LIKE '%Korean%'
    -> ORDER BY CHARACTER_SET_NAME;
+--------------------+---------------------------------+
| CHARACTER_SET_NAME | DESCRIPTION                     |
+--------------------+---------------------------------+
| big5               | Big5 Traditional Chinese        |
| cp932              | SJIS for Windows Japanese       |
| eucjpms            | UJIS for Windows Japanese       |
| euckr              | EUC-KR Korean                   |
| gb18030            | China National Standard GB18030 |
| gb2312             | GB2312 Simplified Chinese       |
| gbk                | GBK Simplified Chinese          |
| sjis               | Shift-JIS Japanese              |
| ujis               | EUC-JP Japanese                 |
+--------------------+---------------------------------+
9 rows in set (0.01 sec)
```

(See Section 19.1, “The INFORMATION_SCHEMA CHARACTER_SETS Table”, for more information.)

MySQL supports three variants of the *GB* (*Guojia Biaozhun*, or *National Standard*, or *Simplified Chinese*) character sets which are official in the People's Republic of China: `gb2312`, `gbk`, and `gb18030` (added in MySQL 5.7.4).

Sometimes people try to insert `gbk` characters into `gb2312`, and it works most of the time because `gbk` is a superset of `gb2312`—but eventually they try to insert a rarer Chinese character and it doesn't work. (See Bug #16072 for an example).

Here, we try to clarify exactly what characters are legitimate in `gb2312` or `gbk`, with reference to the official documents. Please check these references before reporting `gb2312` or `gbk` bugs.

- For a complete listing of the `gb2312` characters, ordered according to the `gb2312_chinese_ci` collation: gb2312
- MySQL's `gbk` is in reality “Microsoft code page 936”. This differs from the official `gbk` for characters `A1A4` (middle dot), `A1AA` (em dash), `A6E0-A6F5`, and `A8BB-A8C0`.
- For a listing of `gbk`/Unicode mappings, see http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT.
- For MySQL's listing of `gbk` characters, see gbk.

A.11.2 I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?

This problem is usually due to a setting in MySQL that doesn't match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using*.

  Use the statement `SELECT VERSION();` to determine this.

- *Make sure that the database is actually using the desired character set*.

This documentation is for an older version. If you're

People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of `SHOW CREATE TABLE tablename` or—better yet—by using this statement:

```
SELECT character_set_name, collation_name
    FROM information_schema.columns
    WHERE table_schema = your_database_name
        AND table_name = your_table_name
        AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly*.

  You can obtain this information for a column `column_name` in the table `table_name` using the following query:

  ```
  SELECT HEX(column_name)
  FROM table_name;
  ```

  `3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip possible—that is, when you select `literal` (or `_introducer hexadecimal-value`), you obtain `literal` as a result*.

  For example, the Japanese Katakana character *Pe* (`ペ`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

  ```
  SELECT 'ペ' AS `ペ`;         /* or SELECT _ucs2 0x30da; */
  ```

  If the result is not also `ペ`, then the round trip has failed.

  For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ペ');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL*.

  Use the `mysql` client program (on Windows: `mysql.exe`) to accomplish this task. If `mysql` displays correctly but your application doesn't, then your problem is probably due to system settings.

  To find out what your settings are, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

  ```
  mysql> SHOW VARIABLES LIKE 'char%';
  +--------------------------+----------------------------------------+
  | Variable_name            | Value                                  |
  +--------------------------+----------------------------------------+
  | character_set_client     | utf8                                   |
  | character_set_connection | utf8                                   |
  | character_set_database   | latin1                                 |
  | character_set_filesystem | binary                                 |
  ```


This documentation is for an older version. If you're

```
| character_set_results    | utf8                                     |
| character_set_server     | latin1                                   |
| character_set_system     | utf8                                     |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+------------------------------------------+
8 rows in set (0.03 sec)
```

These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set and a default for MySQL).

Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it is often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is suitable.

If you cannot control the server settings, and you have no idea what your underlying computer is, then try changing to a common character set for the country that you're in (`euckr` = Korea; `gb18030`, `gb2312` or `gbk` = People's Republic of China; `big5` = Taiwan; `sjis`, `ujis`, `cp932`, or `eucjpms` = Japan; `ucs2` or `utf8` = anywhere). Usually it is necessary to change only the client and connection and results settings. There is a simple statement which changes all three at once: `SET NAMES`. For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing `my.cnf` or `my.ini`. For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...?* for more information.

A.11.3. What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). MySQL's `big5` is in reality Microsoft code page 950, which is very similar to the original `big5` character set. We changed to this character set starting with MySQL version 4.1.16 / 5.0.16 (as a result of Bug #12476). For example, the following statements work in current versions of MySQL, but not in old versions:

```
mysql> CREATE TABLE big5 (BIG5 CHAR(1) CHARACTER SET BIG5);
Query OK, 0 rows affected (0.13 sec)

mysql> INSERT INTO big5 VALUES (0xf9dc);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM big5;
+------+
| big5 |
+------+
| 嬪   |
+------+
1 row in set (0.02 sec)
```


This documentation is for an older version. If you're

A feature request for adding HKSCS extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

A.11.4. Why do Japanese character set conversions fail?

MySQL supports the sjis, ujis, cp932, and eucjpms character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with sjis or ujis) and a Windows client (typically with cp932).

In the following conversion table, the ucs2 column represents the source, and the sjis, cp932, ujis, and eucjpms columns represent the destinations—that is, the last 4 columns provide the hexadecimal result when we use CONVERT(ucs2) or we assign a ucs2 column containing the value to an sjis, cp932, ujis, or eucjpms column.

Character Name	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH ""	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider the following portion of the table.

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark ("?")—this is what is always used when the conversion cannot be performed.

A.11.5. What should I do if I want to convert SJIS `81CA` to `cp932`?

Our answer is: "?". There are serious complaints about this: many people would prefer a "loose" conversion, so that `81CA (NOT SIGN)` in `sjis` becomes `81CA (FULLWIDTH NOT SIGN)` in `cp932`. We are considering a change to this behavior.

A.11.6. How does MySQL represent the Yen (¥) sign?

A problem arises because some versions of Japanese character sets (both `sjis` and `euc`) treat `5C` as a *reverse solidus* (`\`—also known as a backslash), and others treat it as a yen sign (¥).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, *`5C` is always the reverse solidus (`\`)*.

A.11.7. Does MySQL plan to make a separate character set where `5C` is the Yen sign, as at least one other major DBMS does?

This is one possible solution to the Yen sign issue; however, this will not happen in MySQL 5.1 or 6.0.

A.11.8. Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the `euckr` (*Extended Unix Code Korea*) character set, only one problem has been noted.

We use the "ASCII" variant of EUC-KR, in which the code point `0x5c` is REVERSE SOLIDUS, that is `\`, instead of the "KS-Roman" variant of EUC-KR, in which the code point `0x5c` is `WON SIGN`(₩). This means that you cannot convert Unicode `U+20A9` to `euckr`:

```
mysql> SELECT
    ->     CONVERT('₩' USING euckr) AS euckr,
    ->     HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-------+----------+
| euckr | hexeuckr |
+-------+----------+
| ?     | 3F       |
+-------+----------+
1 row in set (0.00 sec)
```

MySQL's graphic Korean chart is here: euckr.

A.11.9. Why do I get `Incorrect string value` error messages?

For illustration, we'll create a table with one Unicode (`ucs2`) column and one Chinese (`gb2312`) column.

```
mysql> CREATE TABLE ch
    -> (ucs2 CHAR(3) CHARACTER SET ucs2,
    -> gb2312 CHAR(3) CHARACTER SET gb2312);
Query OK, 0 rows affected (0.05 sec)
```

We'll try to place the rare character 𠜎 in both columns.

This documentation is for an older version. If you're

```
mysql> INSERT INTO ch VALUES ('A訓B','A訓B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

Ah, there is a warning. Use the following statement to see what it is:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
1 row in set (0.00 sec)
```

So it is a warning about the `gb2312` column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-------+--------------+--------+-------------+
| ucs2  | HEX(ucs2)    | gb2312 | HEX(gb2312) |
+-------+--------------+--------+-------------+
| A訓B | 00416C4C0042 | A?B    | 413F42      |
+-------+--------------+--------+-------------+
1 row in set (0.00 sec)
```

Several things need explanation here:

1. The fact that it is a “warning” rather than an “error” is characteristic of MySQL. We like to try to do what we can, to get the best fit, rather than give up.
2. The 訓 character is not in the `gb2312` character set. We described that problem earlier.
3. If you are using an old version of MySQL, you will probably see a different message.
4. With `sql_mode=TRADITIONAL`, there would be an error message, rather than a warning.

A.11.10 Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the `mysql` client (Windows: `mysql.exe`), and try the same query there. If `mysql` responds correctly, then the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%';`. If you are using Access, then you are most likely connecting with Connector/ODBC. In this case, you should check Configuring Connector/ODBC. If, for instance, you use `big5`, you would enter `SET NAMES 'big5'`. (Note that no `;` is required in this case). If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
              & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/Net, then you must specify the character set in the connection string. See Connecting to MySQL Using Connector/Net, for more information.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you are using PHP, try this:

```
<?php
  $link = new mysqli($host, $usr, $pwd, $db);

  if( mysqli_connect_errno() )
  {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
  }

  $link->query("SET NAMES 'utf8'");
?>
```

In this case, we used `SET NAMES` to change `character_set_client` and `character_set_connection` and `character_set_results`.

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, you should include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` of the HTML page.

If you are using Connector/J, see Using Character Sets and Unicode.

A.11.11. I've upgraded to MySQL 5.0. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single “global” character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a “handshake”, as described in Section 10.1.4, “Connection Character Sets and Collations”:

> When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some of our Asian customers have said that they prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set that it wants to use—however, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1` (unlikely in a CJK area, but this is the default value). Suppose further that the client uses `utf8` because this is what the client's operating system supports. Now, start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

This documentation is for an older version. If you're

```
mysql --default-character-set=utf8
```

The current settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | utf8                                   |
| character_set_connection | utf8                                   |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | utf8                                   |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
8 rows in set (0.01 sec)
```

Now stop the client, and then stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the current settings:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | latin1                                 |
| character_set_connection | latin1                                 |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | latin1                                 |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
8 rows in set (0.01 sec)
```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` is used.

A.11.12 Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

There is a very simple problem with `LIKE` searches on `BINARY` and `BLOB` columns: we need to know the end of a character. With multibyte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but `ペ` requires three bytes, as shown here:

```
+-------------------------+--------------------------+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'ペ') |
+-------------------------+--------------------------+
|                       1 |                        3 |
+-------------------------+--------------------------+
1 row in set (0.00 sec)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If we don't know where the first character ends, then we don't know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a regular CJK character set in the first place, or to convert to a CJK character set before comparing.

This is one reason why MySQL cannot allow encodings of nonexistent characters. If it is not strict about rejecting bad input, then it has no way of knowing where characters end.

For `FULLTEXT` searches, we need to know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary—the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

A.11.13. How do I know whether character *X* is available in all character sets?

The majority of simplified Chinese and basic nonhalfwidth Japanese Kana characters appear in all CJK character sets. This stored procedure accepts a `UCS-2` Unicode character, converts it to all other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
             (ucs2 CHAR(1) character set ucs2,
              utf8 CHAR(1) character set utf8,
              big5 CHAR(1) character set big5,
              cp932 CHAR(1) character set cp932,
              eucjpms CHAR(1) character set eucjpms,
              euckr CHAR(1) character set euckr,
              gb2312 CHAR(1) character set gb2312,
              gbk CHAR(1) character set gbk,
              sjis CHAR(1) character set sjis,
              ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
              big5=ucs2,
              cp932=ucs2,
              eucjpms=ucs2,
              euckr=ucs2,
              gb2312=ucs2,
              gbk=ucs2,
              sjis=ucs2,
              ujis=ucs2;

/* If there is a conversion problem, UPDATE will produce a warning. */

SELECT hex(ucs2) AS ucs2,
       hex(utf8) AS utf8,
       hex(big5) AS big5,
       hex(cp932) AS cp932,
       hex(eucjpms) AS eucjpms,
       hex(euckr) AS euckr,
       hex(gb2312) AS gb2312,
       hex(gbk) AS gbk,
       hex(sjis) AS sjis,
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
      hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//
```

The input can be any single `ucs2` character, or it can be the code point value (hexadecimal representation) of that character. For example, from Unicode's list of `ucs2` encodings and names (http://www.unicode.org/Public/UNIDATA/UnicodeData.txt), we know that the Katakana character *Pe* appears in all CJK character sets, and that its code point value is `0x30da`. If we use this value as the argument to `p_convert()`, the result is as shown here:

```
mysql> CALL p_convert(0x30da)//
+------+--------+------+-------+---------+-------+--------+------+------+------+
| ucs2 | utf8   | big5 | cp932 | eucjpms | euckr | gb2312 | gbk  | sjis | ujis |
+------+--------+------+-------+---------+-------+--------+------+------+------+
| 30DA | E3839A | C772 | 8379  | A5DA    | ABDA  | A5DA   | A5DA | 8379 | A5DA |
+------+--------+------+-------+---------+-------+--------+------+------+------+
1 row in set (0.04 sec)
```

Since none of the column values is `3F`—that is, the question mark character (?)—we know that every conversion worked.

A.11.14. Why do CJK strings sort incorrectly in Unicode? (I)

Sometimes people observe that the result of a `utf8_unicode_ci` or `ucs2_unicode_ci` search, or of an `ORDER BY` sort is not what they think a native would expect. Although we never rule out the possibility that there is a bug, we have found in the past that many people do not read correctly the standard table of weights for the Unicode Collation Algorithm. MySQL uses the table found at http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. This is not the first table you will find by navigating from the `unicode.org` home page, because MySQL uses the older 4.0.0 “allkeys” table, rather than the more recent 4.1.0 table. (The newer `'520'` collations in MySQL 5.6 use the 5.2 “allkeys” table.) This is because we are very wary about changing ordering which affects indexes, lest we bring about situations such as that reported in Bug #16526, illustrated as follows:

```
mysql< CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_unicode_ci);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO tj VALUES ('が'),('か');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM tj WHERE s1 = 'か';
+------+
| s1   |
+------+
| が   |
| か   |
+------+
2 rows in set (0.00 sec)
```

The character in the first result row is not the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the `ucs2` version of the characters:

```
mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
+------+-----------------------------+
| s1   | HEX(CONVERT(s1 USING ucs2)) |
```


This documentation is for an older version. If you're

```
+------+----------------------------+
| が   | 304C                       |
| か   | 304B                       |
+------+----------------------------+
2 rows in set (0.03 sec)
```

Now we search for `304B` and `304C` in the `4.0.0 allkeys` table, and find these lines:

```
304B  ; [.1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C  ; [.1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETTER GA; QQCM
```

The official Unicode names (following the "#" mark) tell us the Japanese syllabary (Hiragana), the informal classification (letter, digit, or punctuation mark), and the Western identifier (`KA` or `GA`, which happen to be voiced and unvoiced components of the same letter pair). More importantly, the *primary weight* (the first hexadecimal number inside the square brackets) is `1E57` on both lines. For comparisons in both searching and sorting, MySQL pays attention to the primary weight only, ignoring all the other numbers. This means that we are sorting が and か correctly according to the Unicode specification. If we wanted to distinguish them, we'd have to use a non-UCA (Unicode Collation Algorithm) collation (`utf8_bin` or `utf8_general_ci`), or to compare the `HEX()` values, or use `ORDER BY CONVERT(s1 USING sjis)`. Being correct "according to Unicode" isn't enough, of course: the person who submitted the bug was equally correct. We plan to add another collation for Japanese according to the JIS X 4061 standard, in which voiced/unvoiced letter pairs like `KA`/`GA` are distinguishable for ordering purposes.

A.11.15 Why do CJK strings sort incorrectly in Unicode? (II)

If you are using Unicode (`ucs2` or `utf8`), and you know what the Unicode sort order is (see Section A.11, "MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets"), but MySQL still seems to sort your table incorrectly, then you should first verify the table character set:

```
mysql> SHOW CREATE TABLE t\G
******************** 1. row ******************
Table: t
Create Table: CREATE TABLE `t` (
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Since the character set appears to be correct, let's see what information the `INFORMATION_SCHEMA.COLUMNS` table can provide about this column:

```
mysql> SELECT COLUMN_NAME, CHARACTER_SET_NAME, COLLATION_NAME
    -> FROM INFORMATION_SCHEMA.COLUMNS
    -> WHERE COLUMN_NAME = 's1'
    -> AND TABLE_NAME = 't';
+-------------+--------------------+-----------------+
| COLUMN_NAME | CHARACTER_SET_NAME | COLLATION_NAME  |
+-------------+--------------------+-----------------+
| s1          | ucs2               | ucs2_general_ci |
+-------------+--------------------+-----------------+
1 row in set (0.01 sec)
```

(See Section 19.4, "The INFORMATION_SCHEMA COLUMNS Table", for more information.)

You can see that the collation is `ucs2_general_ci` instead of `ucs2_unicode_ci`. The reason why this is so can be found using `SHOW CHARSET`, as shown here:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+--------+--------------+------------------+--------+
| Charset | Description   | Default collation | Maxlen |
+--------+--------------+------------------+--------+
| ucs2    | UCS-2 Unicode | ucs2_general_ci   |      2 |
+--------+--------------+------------------+--------+
1 row in set (0.00 sec)
```

For `ucs2` and `utf8`, the default collation is “general”. To specify a Unicode collation, use `COLLATE ucs2_unicode_ci`.

A.11.16. Why are my supplementary characters rejected by MySQL?

Before MySQL 5.5.3, MySQL does not support supplementary characters—that is, characters which need more than 3 bytes—for `UTF-8`. We support only what Unicode calls the *Basic Multilingual Plane / Plane 0*. Only a few very rare Han characters are supplementary; support for them is uncommon. This has led to reports such as that found in Bug #12600, which we rejected as “not a bug”. With `utf8`, we must truncate an input string when we encounter bytes that we don't understand. Otherwise, we wouldn't know how long the bad multibyte character is.

One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks; however, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

As of MySQL 5.5.3, Unicode support is extended to include supplementary characters by means of additional Unicode character sets: `utf16`, `utf32`, and 4-byte `utf8mb4`. These character sets support supplementary Unicode characters outside the Basic Multilingual Plane (BMP).

A.11.17. Shouldn't it be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL has no plan to support the old Vietnamese script using Han characters. MySQL does of course support the modern Vietnamese script with Western characters.

As of MySQL 5.6, there are Vietnamese collations for Unicode character sets, as described in Section 10.1.13.1, “Unicode Character Sets”.

A.11.18. Does MySQL allow CJK characters to be used in database and table names?

This issue is fixed in MySQL 5.1, by automatically rewriting the names of the corresponding directories and files.

For example, if you create a database named 楮 on a server whose operating system does not support CJK in directory names, MySQL creates a directory named `@0w@00a5@00ae`, which is just a fancy way of encoding `E6A5AE`—that is, the Unicode hexadecimal representation for the 楮 character. However, if you run a `SHOW DATABASES` statement, you can see that the database is listed as 楮.

A.11.19. Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at https://wikis.oracle.com/display/mysql/List+of+MySQL+User+Groups.
- View feature requests relating to character set issues at http://tinyurl.com/y6xcuf.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Visit the MySQL Character Sets, Collation, Unicode Forum. http://forums.mysql.com/ also provides foreign-language forums.

A.12 MySQL 5.0 FAQ: Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- Section 20.6.14, “Common Questions and Problems When Using the C API”
- Common Problems with MySQL and PHP
- Connector/ODBC Notes and Tips
- Connector/Net Programming
- MySQL Connector/J 5.1 Developer Guide

A.13 MySQL 5.0 FAQ: Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication.

A.13.1 Must the slave be connected to the master all the time?

No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in synchrony with the master unless you take some special measures.

To ensure that catchup can occur for a slave that has been disconnected, you must not remove binary log files from the master that contain information that has not yet been replicated to the

This documentation is for an older version. If you're
This documentation is for an older version. If you're

slaves. Asynchronous replication can work only if the slave is able to continue reading the binary log from the point where it last read events.

A.13.2. Must I enable networking on my master and slave to enable replication?

Yes, networking must be enabled on the master and slave. If networking is not enabled, the slave cannot connect to the master and transfer the binary log. Check that the `skip-networking` option has not been enabled in the configuration file for either server.

A.13.3. How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

Check the `Seconds_Behind_Master` column in the output from `SHOW SLAVE STATUS`. See Section 16.1.3.1, “Checking Replication Status”.

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see large `Time` values such as 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old. See Section 16.2.1, “Replication Implementation Details”.

A.13.4. How do I force the master to block updates until the slave catches up?

Use the following procedure:

1. On the master, execute these statements:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

   Record the replication coordinates (the current binary log file name and position) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);
```

   The `SELECT` statement blocks until the slave reaches the specified log file and position. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to enable the master to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

A.13.5. What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than

This documentation is for an older version. If you're
This documentation is for an older version. If you're

it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

A.13.6. How can I use replication to improve performance of my system?

Set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb`, `--low-priority-updates`, and `--delay-key-write=ALL` options to get speed improvements on the slave end. In this case, the slave uses nontransactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

A.13.7. What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, Section 16.3.3, “Using Replication for Scale-Out”.

A.13.8. When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you must know your query patterns, and determine empirically by benchmarking the relationship between the throughput for reads and writes on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system. Let `reads` and `writes` denote the number of reads and writes per second, respectively.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is 1200 - 2 * `writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Suppose that the master and each slave have the same capacity, and that we have one master and *N* slaves. Then we have for each server (master or slave):

`reads` = 1200 - 2 * `writes`

`reads` = 9 * `writes` / (*N* + 1) (reads are split, but writes replicated to all slaves)

9 * `writes` / (*N* + 1) + 2 * `writes` = 1200

`writes` = 1200 / (2 + 9/(*N* + 1))

The last equation indicates the maximum number of writes for *N* slaves, given a maximum possible read rate of 1,200 per second and a ratio of nine reads per write.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This analysis yields the following conclusions:

- If *N* = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If *N* = 1, we get up to 184 writes per second.
- If *N* = 8, we get up to 400 writes per second.
- If *N* = 17, we get up to 480 writes per second.
- Eventually, as *N* approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

These computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add *N* replication slaves. However, answering the following questions should help you decide whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

A.13.9. How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the slave.

To handle the process manually, you should be able to switch from a failed master to a pre-configured slave by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see Section 16.3.6, “Switching Masters During Failover”.

A.13.10. How do I tell whether a master server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value shown will be one of `STATEMENT`, `ROW`, or `MIXED`. For `MIXED` mode, statement-based logging is used by default but replication switches automatically to row-based logging under certain conditions, such as unsafe statements. For information about when this may occur, see Mixed Binary Logging Format.

A.13.11. How do I tell a slave to use row-based replication?

Slaves automatically know which format to use.

A.13.12. How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?

This documentation is for an older version. If you're

Start the server with the `--replicate-wild-ignore-table=mysql.%` option to ignore replication for tables in the `mysql` database.

A.13.13. Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on OS X and Windows)?

Yes.

A.13.14. Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Yes.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Appendix B Errors, Error Codes, and Common Problems

Table of Contents

This appendix lists common problems and errors that may occur and potential resolutions, in addition to listing the errors that may appear when you call MySQL from any host language. The first section covers problems and resolutions. Detailed information on errors is provided: One list displays server error messages. Another list displays client program messages.

B.1 Sources of Error Information

There are several sources of error information in MySQL:

- Each SQL statement executed results in an error code, an SQLSTATE value, and an error message, as described in Section B.2, “Types of Error Values”. These errors are returned from the server side; see Section B.3, “Server Error Codes and Messages”.
- Errors can occur on the client side, usually involving problems communicating with the server; see Section B.4, “Client Error Codes and Messages”.
- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` system variable indicates the number of errors, warnings, and notes. The `error_count` system variable indicates the number of errors. Its value excludes warnings and notes.
- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on the slave side.
- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.
- The `perror` program provides information from the command line about error numbers. See Section 4.8.1, “`perror` — Explain Error Codes”.

Descriptions of server and client errors are provided later in this Appendix. For information about errors related to `InnoDB`, see Section 14.2.12, “InnoDB Error Handling”.

B.2 Types of Error Values

When an error occurs in MySQL, the server returns two types of error values:

- A MySQL-specific error code. This value is numeric. It is not portable to other database systems.

This documentation is for an older version. If you're

This documentation is for an older version. If you're

- An SQLSTATE value. The value is a five-character string (for example, `'42S02'`). The values are taken from ANSI SQL and ODBC and are more standardized.

A message string that provides a textual description of the error is also available.

When an error occurs, the MySQL error code, SQLSTATE value, and message string are available using C API functions:

- MySQL error code: Call `mysql_errno()`
- SQLSTATE value: Call `mysql_sqlstate()`
- Error message: Call `mysql_error()`

For prepared statements, the corresponding error functions are `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()`. All error functions are described in Section 20.6, “MySQL C API”.

The number of errors, warnings, and notes for the previous statement can be obtained by calling `mysql_warning_count()`. See Section 20.6.7.72, “mysql_warning_count()”.

The first two characters of an SQLSTATE value indicate the error class:

- Class = `'00'` indicates success.
- Class = `'01'` indicates a warning.
- Class = `'02'` indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
- Class > `'02'` indicates an exception.

B.3 Server Error Codes and Messages

MySQL programs have access to several types of error information when the server returns an error. For example, the `mysql` client program displays errors using the following format:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

The message displayed contains three types of information:

- A numeric error code (`1146`). This number is MySQL-specific and is not portable to other database systems.
- A five-character SQLSTATE value (`'42S02'`). The values are taken from ANSI SQL and ODBC and are more standardized. Not all MySQL error numbers have corresponding SQLSTATE values. In these cases, `'HY000'` (general error) is used.
- A message string that provides a textual description of the error.

For error checking, use error codes, not error messages. Error messages do not change often, but it is possible. Also if the database administrator changes the language setting, that affects the language of error messages.

Error codes are stable across GA releases of a given MySQL series. Before a series reaches GA status, new codes may still be under development and subject to change.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Server error information comes from the following source files. For details about the way that error information is defined, see the MySQL Internals Manual.

- Error message information is listed in the `share/errmsg.txt` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the Message values when they are displayed.
- The Error values listed in `share/errmsg.txt` are used to generate the definitions in the `include/mysqld_error.h` and `include/mysqld_ername.h` MySQL source files.
- The SQLSTATE values listed in `share/errmsg.txt` are used to generate the definitions in the `include/sql_state.h` MySQL source file.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: `1000` SQLSTATE: `HY000` (`ER_HASHCHK`)

  Message: hashchk

  Unused.

- Error: `1001` SQLSTATE: `HY000` (`ER_NISAMCHK`)

  Message: isamchk

  Unused.

- Error: `1002` SQLSTATE: `HY000` (`ER_NO`)

  Message: NO

  Used in the construction of other messages.

- Error: `1003` SQLSTATE: `HY000` (`ER_YES`)

  Message: YES

  Used in the construction of other messages.

  Extended `EXPLAIN` format generates Note messages. `ER_YES` is used in the `Code` column for these messages in subsequent `SHOW WARNINGS` output.

- Error: `1004` SQLSTATE: `HY000` (`ER_CANT_CREATE_FILE`)

  Message: Can't create file '%s' (errno: %d)

  Occurs for failure to copy an `.frm` file to a new location, during execution of a `CREATE TABLE` *`dst`* `LIKE` *`src`* statement when the server tries to copy the source table `.frm` file to the destination table `.frm` file.

  Possible causes: Permissions problem for source `.frm` file; destination `.frm` file already exists but is not writeable.

- Error: `1005` SQLSTATE: `HY000` (`ER_CANT_CREATE_TABLE`)

  Message: Can't create table '%s' (errno: %d)

  `InnoDB` reports this error when a table cannot be created. If the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers

This documentation is for an older version. If you're
This documentation is for an older version. If you're

to error −1, table creation probably failed because the table includes a column name that matched the name of an internal `InnoDB` table.

- Error: `1006` SQLSTATE: `HY000` (`ER_CANT_CREATE_DB`)

  Message: Can't create database '%s' (errno: %d)

- Error: `1007` SQLSTATE: `HY000` (`ER_DB_CREATE_EXISTS`)

  Message: Can't create database '%s'; database exists

  An attempt to create a database failed because the database already exists.

  Drop the database first if you really want to replace an existing database, or add an `IF NOT EXISTS` clause to the `CREATE DATABASE` statement if to retain an existing database without having the statement produce an error.

- Error: `1008` SQLSTATE: `HY000` (`ER_DB_DROP_EXISTS`)

  Message: Can't drop database '%s'; database doesn't exist

- Error: `1009` SQLSTATE: `HY000` (`ER_DB_DROP_DELETE`)

  Message: Error dropping database (can't delete '%s', errno: %d)

- Error: `1010` SQLSTATE: `HY000` (`ER_DB_DROP_RMDIR`)

  Message: Error dropping database (can't rmdir '%s', errno: %d)

- Error: `1011` SQLSTATE: `HY000` (`ER_CANT_DELETE_FILE`)

  Message: Error on delete of '%s' (errno: %d)

- Error: `1012` SQLSTATE: `HY000` (`ER_CANT_FIND_SYSTEM_REC`)

  Message: Can't read record in system table

  Returned by `InnoDB` for attempts to access `InnoDB` `INFORMATION_SCHEMA` tables when `InnoDB` is unavailable.

- Error: `1013` SQLSTATE: `HY000` (`ER_CANT_GET_STAT`)

  Message: Can't get status of '%s' (errno: %d)

- Error: `1014` SQLSTATE: `HY000` (`ER_CANT_GET_WD`)

  Message: Can't get working directory (errno: %d)

- Error: `1015` SQLSTATE: `HY000` (`ER_CANT_LOCK`)

  Message: Can't lock file (errno: %d)

- Error: `1016` SQLSTATE: `HY000` (`ER_CANT_OPEN_FILE`)

  Message: Can't open file: '%s' (errno: %d)

  `InnoDB` reports this error when the table from the `InnoDB` data files cannot be found, even though the `.frm` file for the table exists. See Section 14.2.13.3, “Troubleshooting InnoDB Data Dictionary Operations”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND)

  Message: Can't find file: '%s' (errno: %d)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

  Message: Can't read dir of '%s' (errno: %d)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

  Message: Can't change dir to '%s' (errno: %d)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD)

  Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL)

  Message: Disk full (%s); waiting for someone to free some space...

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY)

  Message: Can't write; duplicate key in table '%s'

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

  Message: Error on close of '%s' (errno: %d)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

  Message: Error reading file '%s' (errno: %d)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

  Message: Error on rename of '%s' to '%s' (errno: %d)

  InnoDB reports this error if you attempt to drop the last index that can enforce a particular referential constraint. As of MySQL 5.5, this error message is replaced by ERROR 1553.

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)

  Message: Error writing file '%s' (errno: %d)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED)

  Message: '%s' is locked against change

- Error: 1028 SQLSTATE: HY000 (ER_FILSORT_ABORT)

  Message: Sort aborted

- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND)

  Message: View '%s' doesn't exist for '%s'

- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO)

  Message: Got error %d from storage engine

  Check the %d value to see what the OS error means. For example, 28 indicates that you have run out of disk space.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL_HA)

  Message: Table storage engine for '%s' doesn't have this option

- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND)

  Message: Can't find record in '%s'

- Error: 1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE)

  Message: Incorrect information in file: '%s'

- Error: 1034 SQLSTATE: HY000 (ER_NOT_KEYFILE)

  Message: Incorrect key file for table '%s'; try to repair it

- Error: 1035 SQLSTATE: HY000 (ER_OLD_KEYFILE)

  Message: Old key file for table '%s'; repair it!

- Error: 1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY)

  Message: Table '%s' is read only

- Error: 1037 SQLSTATE: HY001 (ER_OUTOFMEMORY)

  Message: Out of memory; restart server and try again (needed %d bytes)

- Error: 1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY)

  Message: Out of sort memory; increase server sort buffer size

- Error: 1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF)

  Message: Unexpected EOF found when reading file '%s' (errno: %d)

- Error: 1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR)

  Message: Too many connections

- Error: 1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES)

  Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error: 1042 SQLSTATE: 08S01 (ER_BAD_HOST_ERROR)

  Message: Can't get hostname for your address

- Error: 1043 SQLSTATE: 08S01 (ER_HANDSHAKE_ERROR)

  Message: Bad handshake

- Error: 1044 SQLSTATE: 42000 (ER_DBACCESS_DENIED_ERROR)

  Message: Access denied for user '%s'@'%s' to database '%s'

- Error: 1045 SQLSTATE: 28000 (ER_ACCESS_DENIED_ERROR)

  Message: Access denied for user '%s'@'%s' (using password: %s)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1046 SQLSTATE: 3D000 (ER_NO_DB_ERROR)

  Message: No database selected

- Error: 1047 SQLSTATE: 08S01 (ER_UNKNOWN_COM_ERROR)

  Message: Unknown command

- Error: 1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR)

  Message: Column '%s' cannot be null

- Error: 1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR)

  Message: Unknown database '%s'

- Error: 1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR)

  Message: Table '%s' already exists

- Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)

  Message: Unknown table '%s'

- Error: 1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR)

  Message: Column '%s' in %s is ambiguous

  ```
  %s = column name
  %s = location of column (for example, "field list")
  ```

  Likely cause: A column appears in a query without appropriate qualification, such as in a select list or ON clause.

  Examples:

  ```
  mysql> SELECT i FROM t INNER JOIN t AS t2;
  ERROR 1052 (23000): Column 'i' in field list is ambiguous

  mysql> SELECT * FROM t LEFT JOIN t AS t2 ON i = i;
  ERROR 1052 (23000): Column 'i' in on clause is ambiguous
  ```

  Resolution:

  - Qualify the column with the appropriate table name:

    ```
    mysql> SELECT t2.i FROM t INNER JOIN t AS t2;
    ```

  - Modify the query to avoid the need for qualification:

    ```
    mysql> SELECT * FROM t LEFT JOIN t AS t2 USING (i);
    ```

- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

  Message: Server shutdown in progress

- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Unknown column '%s' in '%s'

- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

  Message: '%s' isn't in GROUP BY

- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD)

  Message: Can't group on '%s'

- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT)

  Message: Statement has sum functions and columns in same statement

- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT)

  Message: Column count doesn't match value count

- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT)

  Message: Identifier name '%s' is too long

- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME)

  Message: Duplicate column name '%s'

- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME)

  Message: Duplicate key name '%s'

- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY)

  Message: Duplicate entry '%s' for key %d

- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC)

  Message: Incorrect column specifier for column '%s'

- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR)

  Message: %s near '%s' at line %d

- Error: 1065 SQLSTATE: 42000 (ER_EMPTY_QUERY)

  Message: Query was empty

- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE)

  Message: Not unique table/alias: '%s'

- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT)

  Message: Invalid default value for '%s'

- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY)

  Message: Multiple primary key defined

- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Too many keys specified; max %d keys allowed

- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS)

  Message: Too many key parts specified; max %d parts allowed

- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

  Message: Specified key was too long; max key length is %d bytes

- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS)

  Message: Key column '%s' doesn't exist in table

- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

  Message: BLOB column '%s' can't be used in key specification with the used table type

- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

  Message: Column length too big for column '%s' (max = %lu); use BLOB or TEXT instead

- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

  Message: Incorrect table definition; there can be only one auto column and it must be defined as a key

- Error: 1076 SQLSTATE: HY000 (ER_READY)

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d

- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)

  Message: %s: Normal shutdown

- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)

  Message: %s: Got signal %d. Aborting!

- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)

  Message: %s: Shutdown complete

- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)

  Message: %s: Forcing close of thread %ld user: '%s'

- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)

  Message: Can't create IP socket

- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)

  Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table

- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)

  Message: Field separator argument is not what is expected; check the manual

- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'

- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)

  Message: The file '%s' must be in the database directory or be readable by all

- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

  Message: File '%s' already exists

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO)

  Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO)

  Message: Records: %ld Duplicates: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

  Message: Incorrect sub part key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique sub keys

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

  Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

  Message: Can't DROP '%s'; check that column/key exists

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO)

  Message: Records: %ld Duplicates: %ld Warnings: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

  Message: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

  Message: Unknown thread id: %lu

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

  Message: You are not owner of thread %lu

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)

  Message: No tables used

- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)

  Message: Too many strings for column %s and SET

- Error: 1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE)

  Message: Can't generate a unique log-filename %s.(1-999)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE)

  Message: Table '%s' was locked with a READ lock and can't be updated

- Error: 1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED)

  Message: Table '%s' was not locked with LOCK TABLES

- Error: 1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT)

  Message: BLOB/TEXT column '%s' can't have a default value

- Error: 1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME)

  Message: Incorrect database name '%s'

- Error: 1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME)

  Message: Incorrect table name '%s'

- Error: 1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT)

  Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay

- Error: 1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR)

  Message: Unknown error

- Error: 1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE)

  Message: Unknown procedure '%s'

- Error: 1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE)

  Message: Incorrect parameter count to procedure '%s'

- Error: 1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE)

  Message: Incorrect parameters to procedure '%s'

- Error: 1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE)

  Message: Unknown table '%s' in %s

- Error: 1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE)

  Message: Column '%s' specified twice

- Error: 1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE)

  Message: Invalid use of group function

- Error: 1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION)

  Message: Table '%s' uses an extension that doesn't exist in this MySQL version

- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)

  Message: A table must have at least 1 column

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)

  Message: The table '%s' is full

  InnoDB reports this error when the system tablespace runs out of free space. Reconfigure the system tablespace to add a new data file.

- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)

  Message: Unknown character set: '%s'

- Error: 1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES)

  Message: Too many tables; MySQL can only use %d tables in a join

- Error: 1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS)

  Message: Too many columns

- Error: 1118 SQLSTATE: 42000 (ER_TOO_BIG_ROWSIZE)

  Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs

- Error: 1119 SQLSTATE: HY000 (ER_STACK_OVERRUN)

  Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed

- Error: 1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN)

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

- Error: 1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX)

  Message: Column '%s' is used with UNIQUE or INDEX but is not defined as NOT NULL

- Error: 1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF)

  Message: Can't load function '%s'

- Error: 1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF)

  Message: Can't initialize function '%s'; %s

- Error: 1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS)

  Message: No paths allowed for shared library

- Error: 1125 SQLSTATE: HY000 (ER_UDF_EXISTS)

  Message: Function '%s' already exists

- Error: 1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY)

  Message: Can't open shared library '%s' (errno: %d %s)

- Error: 1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY)

  Message: Can't find function '%s' in library

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1128 SQLSTATE: HY000 (ER_FUNCTION_NOT_DEFINED)

  Message: Function '%s' is not defined

- Error: 1129 SQLSTATE: HY000 (ER_HOST_IS_BLOCKED)

  Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error: 1130 SQLSTATE: HY000 (ER_HOST_NOT_PRIVILEGED)

  Message: Host '%s' is not allowed to connect to this MySQL server

- Error: 1131 SQLSTATE: 42000 (ER_PASSWORD_ANONYMOUS_USER)

  Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords

- Error: 1132 SQLSTATE: 42000 (ER_PASSWORD_NOT_ALLOWED)

  Message: You must have privileges to update tables in the mysql database to be able to change passwords for others

- Error: 1133 SQLSTATE: 42000 (ER_PASSWORD_NO_MATCH)

  Message: Can't find any matching row in the user table

- Error: 1134 SQLSTATE: HY000 (ER_UPDATE_INFO)

  Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error: 1135 SQLSTATE: HY000 (ER_CANT_CREATE_THREAD)

  Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error: 1136 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT_ON_ROW)

  Message: Column count doesn't match value count at row %ld

- Error: 1137 SQLSTATE: HY000 (ER_CANT_REOPEN_TABLE)

  Message: Can't reopen table: '%s'

- Error: 1138 SQLSTATE: 22004 (ER_INVALID_USE_OF_NULL)

  Message: Invalid use of NULL value

- Error: 1139 SQLSTATE: 42000 (ER_REGEXP_ERROR)

  Message: Got error '%s' from regexp

- Error: 1140 SQLSTATE: 42000 (ER_MIX_OF_GROUP_FUNC_AND_FIELDS)

  Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: 1141 SQLSTATE: 42000 (ER_NONEXISTING_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s'

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1142 SQLSTATE: 42000 (ER_TABLEACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for table '%s'

- Error: 1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: 1144 SQLSTATE: 42000 (ER_ILLEGAL_GRANT_FOR_TABLE)

  Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error: 1145 SQLSTATE: 42000 (ER_GRANT_WRONG_HOST_OR_USER)

  Message: The host or user argument to GRANT is too long

- Error: 1146 SQLSTATE: 42S02 (ER_NO_SUCH_TABLE)

  Message: Table '%s.%s' doesn't exist

- Error: 1147 SQLSTATE: 42000 (ER_NONEXISTING_TABLE_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: 1148 SQLSTATE: 42000 (ER_NOT_ALLOWED_COMMAND)

  Message: The used command is not allowed with this MySQL version

- Error: 1149 SQLSTATE: 42000 (ER_SYNTAX_ERROR)

  Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error: 1150 SQLSTATE: HY000 (ER_DELAYED_CANT_CHANGE_LOCK)

  Message: Delayed insert thread couldn't get requested lock for table %s

- Error: 1151 SQLSTATE: HY000 (ER_TOO_MANY_DELAYED_THREADS)

  Message: Too many delayed threads in use

- Error: 1152 SQLSTATE: 08S01 (ER_ABORTING_CONNECTION)

  Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: 1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE)

  Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: 1154 SQLSTATE: 08S01 (ER_NET_READ_ERROR_FROM_PIPE)

  Message: Got a read error from the connection pipe

- Error: 1155 SQLSTATE: 08S01 (ER_NET_FCNTL_ERROR)

  Message: Got an error from fcntl()

- Error: 1156 SQLSTATE: 08S01 (ER_NET_PACKETS_OUT_OF_ORDER)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Got packets out of order

- Error: 1157 SQLSTATE: 08S01 (ER_NET_UNCOMPRESS_ERROR)

  Message: Couldn't uncompress communication packet

- Error: 1158 SQLSTATE: 08S01 (ER_NET_READ_ERROR)

  Message: Got an error reading communication packets

- Error: 1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED)

  Message: Got timeout reading communication packets

- Error: 1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE)

  Message: Got an error writing communication packets

- Error: 1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED)

  Message: Got timeout writing communication packets

- Error: 1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING)

  Message: Result string is longer than 'max_allowed_packet' bytes

- Error: 1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB)

  Message: The used table type doesn't support BLOB/TEXT columns

- Error: 1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT)

  Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: 1165 SQLSTATE: HY000 (ER_DELAYED_INSERT_TABLE_LOCKED)

  Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES

- Error: 1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME)

  Message: Incorrect column name '%s'

- Error: 1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN)

  Message: The used storage engine can't index column '%s'

- Error: 1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE)

  Message: Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist

- Error: 1169 SQLSTATE: 23000 (ER_DUP_UNIQUE)

  Message: Can't write, because of unique constraint, to table '%s'

- Error: 1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH)

  Message: BLOB/TEXT column '%s' used in key specification without a key length

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL)

  Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: 1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS)

  Message: Result consisted of more than one row

- Error: 1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY)

  Message: This table type requires a primary key

  InnoDB reports this error when you attempt to drop an implicit clustered index (the first UNIQUE NOT NULL index) if the table did not contain a PRIMARY KEY.

  InnoDB should no longer report this error as of MySQL 5.5. For tables without an explicit PRIMARY KEY, InnoDB creates an implicit clustered index using the first columns of the table that are declared UNIQUE and NOT NULL. When you drop such an index, InnoDB now automatically copies the table and rebuilds the index using a different UNIQUE NOT NULL group of columns or a system-generated key. Since this operation changes the primary key, it uses the slow method of copying the table and re-creating the index, rather than the Fast Index Creation technique from Implementation Details of Fast Index Creation.

- Error: 1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED)

  Message: This version of MySQL is not compiled with RAID support

- Error: 1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE)

  Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

- Error: 1176 SQLSTATE: HY000 (ER_KEY_DOES_NOT_EXITS)

  Message: Key '%s' doesn't exist in table '%s'

- Error: 1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE)

  Message: Can't open table

- Error: 1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED)

  Message: The storage engine for the table doesn't support %s

- Error: 1179 SQLSTATE: 25000 (ER_CANT_DO_THIS_DURING_AN_TRANSACTION)

  Message: You are not allowed to execute this command in a transaction

- Error: 1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT)

  Message: Got error %d during COMMIT

- Error: 1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK)

  Message: Got error %d during ROLLBACK

- Error: 1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Got error %d during FLUSH_LOGS

- Error: 1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT)

  Message: Got error %d during CHECKPOINT

- Error: 1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION)

  Message: Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)

- Error: 1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED)

  Message: The storage engine for the table does not support binary table dump

- Error: 1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED)

  Message: Binlog closed, cannot RESET MASTER

- Error: 1187 SQLSTATE: HY000 (ER_INDEX_REBUILD)

  Message: Failed rebuilding the index of dumped table '%s'

- Error: 1188 SQLSTATE: HY000 (ER_MASTER)

  Message: Error from master: '%s'

- Error: 1189 SQLSTATE: 08S01 (ER_MASTER_NET_READ)

  Message: Net error reading from master

- Error: 1190 SQLSTATE: 08S01 (ER_MASTER_NET_WRITE)

  Message: Net error writing to master

- Error: 1191 SQLSTATE: HY000 (ER_FT_MATCHING_KEY_NOT_FOUND)

  Message: Can't find FULLTEXT index matching the column list

- Error: 1192 SQLSTATE: HY000 (ER_LOCK_OR_ACTIVE_TRANSACTION)

  Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: 1193 SQLSTATE: HY000 (ER_UNKNOWN_SYSTEM_VARIABLE)

  Message: Unknown system variable '%s'

- Error: 1194 SQLSTATE: HY000 (ER_CRASHED_ON_USAGE)

  Message: Table '%s' is marked as crashed and should be repaired

- Error: 1195 SQLSTATE: HY000 (ER_CRASHED_ON_REPAIR)

  Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: 1196 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK)

  Message: Some non-transactional changed tables couldn't be rolled back

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1197 SQLSTATE: HY000 (ER_TRANS_CACHE_FULL)

  Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: 1198 SQLSTATE: HY000 (ER_SLAVE_MUST_STOP)

  Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: 1199 SQLSTATE: HY000 (ER_SLAVE_NOT_RUNNING)

  Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: 1200 SQLSTATE: HY000 (ER_BAD_SLAVE)

  Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: 1201 SQLSTATE: HY000 (ER_MASTER_INFO)

  Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: 1202 SQLSTATE: HY000 (ER_SLAVE_THREAD)

  Message: Could not create slave thread; check system resources

- Error: 1203 SQLSTATE: 42000 (ER_TOO_MANY_USER_CONNECTIONS)

  Message: User %s already has more than 'max_user_connections' active connections

- Error: 1204 SQLSTATE: HY000 (ER_SET_CONSTANTS_ONLY)

  Message: You may only use constant expressions with SET

- Error: 1205 SQLSTATE: HY000 (ER_LOCK_WAIT_TIMEOUT)

  Message: Lock wait timeout exceeded; try restarting transaction

  InnoDB reports this error when lock wait timeout expires. The statement that waited too long was rolled back (not the entire transaction). You can increase the value of the innodb_lock_wait_timeout configuration option if SQL statements should wait longer for other transactions to complete, or decrease it if too many long-running transactions are causing locking problems and reducing concurrency on a busy system.

- Error: 1206 SQLSTATE: HY000 (ER_LOCK_TABLE_FULL)

  Message: The total number of locks exceeds the lock table size

  InnoDB reports this error when the total number of locks exceeds the amount of memory devoted to managing locks. To avoid this error, increase the value of innodb_buffer_pool_size. Within an individual application, a workaround may be to break a large operation into smaller pieces. For example, if the error occurs for a large INSERT, perform several smaller INSERT operations.

- Error: 1207 SQLSTATE: 25000 (ER_READ_ONLY_TRANSACTION)

  Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: 1208 SQLSTATE: HY000 (ER_DROP_DB_WITH_READ_LOCK)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: 1209 SQLSTATE: HY000 (ER_CREATE_DB_WITH_READ_LOCK)

  Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: 1210 SQLSTATE: HY000 (ER_WRONG_ARGUMENTS)

  Message: Incorrect arguments to %s

- Error: 1211 SQLSTATE: 42000 (ER_NO_PERMISSION_TO_CREATE_USER)

  Message: '%s'@'%s' is not allowed to create new users

- Error: 1212 SQLSTATE: HY000 (ER_UNION_TABLES_IN_DIFFERENT_DIR)

  Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: 1213 SQLSTATE: 40001 (ER_LOCK_DEADLOCK)

  Message: Deadlock found when trying to get lock; try restarting transaction

  InnoDB reports this error when a transaction encounters a deadlock and is automatically rolled back so that your application can take corrective action. To recover from this error, run all the operations in this transaction again. A deadlock occurs when requests for locks arrive in inconsistent order between transactions. The transaction that was rolled back released all its locks, and the other transaction can now get all the locks it requested. Thus, when you re-run the transaction that was rolled back, it might have to wait for other transactions to complete, but typically the deadlock does not recur. If you encounter frequent deadlocks, make the sequence of locking operations (LOCK TABLES, SELECT ... FOR UPDATE, and so on) consistent between the different transactions or applications that experience the issue. See Section 14.2.8.9, “How to Cope with Deadlocks” for details.

- Error: 1214 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_FT)

  Message: The used table type doesn't support FULLTEXT indexes

- Error: 1215 SQLSTATE: HY000 (ER_CANNOT_ADD_FOREIGN)

  Message: Cannot add foreign key constraint

- Error: 1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW)

  Message: Cannot add or update a child row: a foreign key constraint fails

  InnoDB reports this error when you try to add a row but there is no parent row, and a foreign key constraint fails. Add the parent row first.

- Error: 1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED)

  Message: Cannot delete or update a parent row: a foreign key constraint fails

  InnoDB reports this error when you try to delete a parent row that has children, and a foreign key constraint fails. Delete the children first.

- Error: 1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER)

  Message: Error connecting to master: %s

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER)

  Message: Error running query on master: %s

- Error: 1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND)

  Message: Error when executing command %s: %s

- Error: 1221 SQLSTATE: HY000 (ER_WRONG_USAGE)

  Message: Incorrect usage of %s and %s

- Error: 1222 SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT)

  Message: The used SELECT statements have a different number of columns

- Error: 1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK)

  Message: Can't execute the query because you have a conflicting read lock

- Error: 1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED)

  Message: Mixing of transactional and non-transactional tables is disabled

- Error: 1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT)

  Message: Option '%s' used twice in statement

- Error: 1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED)

  Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: 1227 SQLSTATE: 42000 (ER_SPECIFIC_ACCESS_DENIED_ERROR)

  Message: Access denied; you need the %s privilege for this operation

- Error: 1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE)

  Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

  Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL

- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

  Message: Variable '%s' doesn't have a default value

- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

  Message: Variable '%s' can't be set to the value of '%s'

- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

  Message: Incorrect argument type to variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

  Message: Variable '%s' can only be set, not read

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

  Message: Incorrect usage/placement of '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

  Message: This version of MySQL doesn't yet support '%s'

- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG)

  Message: Got fatal error %d: '%s' from master when reading data from binary log

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

  Message: Slave SQL thread ignored the query because of replicate-*-table rules

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

  Message: Variable '%s' is a %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

  Message: Incorrect foreign key definition for '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

  Message: Key reference and table reference don't match

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)

  Message: Operand should contain %d column(s)

- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)

  Message: Subquery returns more than 1 row

- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)

  Message: Unknown prepared statement handler (%.*s) given to %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)

  Message: Help database is corrupt or does not exist

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)

  Message: Cyclic reference on subqueries

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)

  Message: Converting column '%s' from %s to %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)

  Message: Reference '%s' not supported (%s)

- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)

  Message: Every derived table must have its own alias

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)

  Message: Select %u was reduced during optimization

- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)

  Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

  Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

  Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

  Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

  Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

  Message: Slave already has been stopped

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

  Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

  Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

  Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: 1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR)

  Message: ZLIB: Input data corrupted

- Error: 1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT)

  Message: %d line(s) were cut by GROUP_CONCAT()

- Error: 1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS)

  Message: Row %ld doesn't contain data for all columns

- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS)

  Message: Row %ld was truncated; it contained more data than there were input columns

- Error: 1263 SQLSTATE: 22004 (ER_WARN_NULL_TO_NOTNULL)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Column was set to data type implicit default; NULL supplied for NOT NULL column '%s' at row %ld

- Error: 1264 SQLSTATE: 22003 (ER_WARN_DATA_OUT_OF_RANGE)

  Message: Out of range value adjusted for column '%s' at row %ld

- Error: 1265 SQLSTATE: 01000 (WARN_DATA_TRUNCATED)

  Message: Data truncated for column '%s' at row %ld

- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER)

  Message: Using storage engine %s for table '%s'

- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS)

  Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER)

  Message: Cannot drop one or more of the requested users

- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS)

  Message: Can't revoke all privileges for one or more of the requested users

- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS)

  Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS)

  Message: Illegal mix of collations for operation '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT)

  Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

  Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

  Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

  Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

  Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

  Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)

  Message: SQL thread is not to be started so UNTIL options are ignored

- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)

  Message: Incorrect index name '%s'

- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)

  Message: Incorrect catalog name '%s'

- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)

  Message: Query cache failed to set size %lu; new query cache size is %lu

- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)

  Message: Column '%s' cannot be part of FULLTEXT index

- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)

  Message: Unknown key cache '%s'

- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)

  Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work

- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)

  Message: Unknown table engine '%s'

- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)

  Message: '%s' is deprecated; use '%s' instead

- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)

  Message: The target table %s of the %s is not updatable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

  Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

  Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

  Message: Column '%s' has duplicated value '%s' in %s

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1292 SQLSTATE: 22007 (ER_TRUNCATED_WRONG_VALUE)

  Message: Truncated incorrect %s value: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS)

  Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE)

  Message: Invalid ON UPDATE clause for '%s' column

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS)

  Message: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG)

  Message: Got error %d '%s' from %s

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG)

  Message: Got temporary error %d '%s' from %s

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE)

  Message: Unknown or incorrect time zone: '%s'

- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP)

  Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING)

  Message: Invalid %s character string: '%s'

- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED)

  Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS)

  Message: Conflicting declarations: '%s%s' and '%s%s'

- Error: 1303 SQLSTATE: 2F003 (ER_SP_NO_RECURSIVE_CREATE)

  Message: Can't create a %s from within another stored routine

- Error: 1304 SQLSTATE: 42000 (ER_SP_ALREADY_EXISTS)

  Message: %s %s already exists

- Error: 1305 SQLSTATE: 42000 (ER_SP_DOES_NOT_EXIST)

  Message: %s %s does not exist

- Error: 1306 SQLSTATE: HY000 (ER_SP_DROP_FAILED)

  Message: Failed to DROP %s %s

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1307 SQLSTATE: HY000 (ER_SP_STORE_FAILED)

  Message: Failed to CREATE %s %s

- Error: 1308 SQLSTATE: 42000 (ER_SP_LILABEL_MISMATCH)

  Message: %s with no matching label: %s

- Error: 1309 SQLSTATE: 42000 (ER_SP_LABEL_REDEFINE)

  Message: Redefining label %s

- Error: 1310 SQLSTATE: 42000 (ER_SP_LABEL_MISMATCH)

  Message: End-label %s without match

- Error: 1311 SQLSTATE: 01000 (ER_SP_UNINIT_VAR)

  Message: Referring to uninitialized variable %s

- Error: 1312 SQLSTATE: 0A000 (ER_SP_BADSELECT)

  Message: PROCEDURE %s can't return a result set in the given context

- Error: 1313 SQLSTATE: 42000 (ER_SP_BADRETURN)

  Message: RETURN is only allowed in a FUNCTION

- Error: 1314 SQLSTATE: 0A000 (ER_SP_BADSTATEMENT)

  Message: %s is not allowed in stored procedures

- Error: 1315 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_IGNORED)

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored

- Error: 1316 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_TRANSLATED)

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN

- Error: 1317 SQLSTATE: 70100 (ER_QUERY_INTERRUPTED)

  Message: Query execution was interrupted

- Error: 1318 SQLSTATE: 42000 (ER_SP_WRONG_NO_OF_ARGS)

  Message: Incorrect number of arguments for %s %s; expected %u, got %u

- Error: 1319 SQLSTATE: 42000 (ER_SP_COND_MISMATCH)

  Message: Undefined CONDITION: %s

- Error: 1320 SQLSTATE: 42000 (ER_SP_NORETURN)

  Message: No RETURN found in FUNCTION %s

- Error: 1321 SQLSTATE: 2F005 (ER_SP_NORETURNEND)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: FUNCTION %s ended without RETURN

- Error: 1322 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_QUERY)

  Message: Cursor statement must be a SELECT

- Error: 1323 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_SELECT)

  Message: Cursor SELECT must not have INTO

- Error: 1324 SQLSTATE: 42000 (ER_SP_CURSOR_MISMATCH)

  Message: Undefined CURSOR: %s

- Error: 1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN)

  Message: Cursor is already open

- Error: 1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN)

  Message: Cursor is not open

- Error: 1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR)

  Message: Undeclared variable: %s

- Error: 1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS)

  Message: Incorrect number of FETCH variables

- Error: 1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)

  Message: No data - zero rows fetched, selected, or processed

- Error: 1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM)

  Message: Duplicate parameter: %s

- Error: 1331 SQLSTATE: 42000 (ER_SP_DUP_VAR)

  Message: Duplicate variable: %s

- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

  Message: Duplicate condition: %s

- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

  Message: Duplicate cursor: %s

- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

  Message: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

  Message: Subselect value not supported

- Error: 1336 SQLSTATE: 42000 (ER_SP_NO_USE)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: USE is not allowed in a stored procedure

In 5.0.12: ER_SP_NO_USE was renamed to ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG.

ER_SP_NO_USE was removed after 5.0.11.

- Error: 1336 SQLSTATE: 0A000 (ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG)

  Message: %s is not allowed in stored function or trigger

  In 5.0.12: ER_SP_NO_USE was renamed to ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG.

  ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG was added in 5.0.12.

- Error: 1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

  Message: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

  Message: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

  Message: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

  Message: Configuration file '%s' is too big

- Error: 1341 SQLSTATE: HY000 (ER_FPARSER_BAD_HEADER)

  Message: Malformed file type header in file '%s'

- Error: 1342 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_COMMENT)

  Message: Unexpected end of file while parsing comment '%s'

- Error: 1343 SQLSTATE: HY000 (ER_FPARSER_ERROR_IN_PARAMETER)

  Message: Error while parsing parameter '%s' (line: '%s')

- Error: 1344 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER)

  Message: Unexpected end of file while skipping unknown parameter '%s'

- Error: 1345 SQLSTATE: HY000 (ER_VIEW_NO_EXPLAIN)

  Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error: 1346 SQLSTATE: HY000 (ER_FRM_UNKNOWN_TYPE)

  Message: File '%s' has unknown type '%s' in its header

- Error: 1347 SQLSTATE: HY000 (ER_WRONG_OBJECT)

  Message: '%s.%s' is not %s

- Error: 1348 SQLSTATE: HY000 (ER_NONUPDATEABLE_COLUMN)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Column '%s' is not updatable

- Error: 1349 SQLSTATE: HY000 (ER_VIEW_SELECT_DERIVED)

  Message: View's SELECT contains a subquery in the FROM clause

- Error: 1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE)

  Message: View's SELECT contains a '%s' clause

- Error: 1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE)

  Message: View's SELECT contains a variable or parameter

- Error: 1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE)

  Message: View's SELECT refers to a temporary table '%s'

- Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

  Message: View's SELECT and view's field list have different column counts

- Error: 1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE)

  Message: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error: 1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY)

  Message: View being updated does not have complete key of underlying table in it

- Error: 1356 SQLSTATE: HY000 (ER_VIEW_INVALID)

  Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

- Error: 1357 SQLSTATE: HY000 (ER_SP_NO_DROP_SP)

  Message: Can't drop or alter a %s from within another stored routine

- Error: 1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR)

  Message: GOTO is not allowed in a stored procedure handler

- Error: 1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS)

  Message: Trigger already exists

- Error: 1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST)

  Message: Trigger does not exist

- Error: 1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE)

  Message: Trigger's '%s' is view or temporary table

- Error: 1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW)

  Message: Updating of %s row is not allowed in %strigger

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG)

  Message: There is no %s row in %s trigger

- Error: 1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD)

  Message: Field '%s' doesn't have a default value

- Error: 1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO)

  Message: Division by 0

- Error: 1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD)

  Message: Incorrect %s value: '%s' for column '%s' at row %ld

- Error: 1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE)

  Message: Illegal %s '%s' value found during parsing

- Error: 1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK)

  Message: CHECK OPTION on non-updatable view '%s.%s'

- Error: 1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED)

  Message: CHECK OPTION failed '%s.%s'

- Error: 1370 SQLSTATE: 42000 (ER_PROCACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for routine '%s'

- Error: 1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL)

  Message: Failed purging old relay logs: %s

- Error: 1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH)

  Message: Password hash should be a %d-digit hexadecimal number

- Error: 1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG)

  Message: Target log not found in binlog index

- Error: 1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ)

  Message: I/O error reading log index file

- Error: 1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED)

  Message: Server configuration does not permit binlog purge

- Error: 1376 SQLSTATE: HY000 (ER_FSEEK_FAIL)

  Message: Failed on fseek()

- Error: 1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR)

  Message: Fatal error during log purge

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1378 SQLSTATE: HY000 (ER_LOG_IN_USE)

  Message: A purgeable log is in use, will not purge

- Error: 1379 SQLSTATE: HY000 (ER_LOG_PURGE_UNKNOWN_ERR)

  Message: Unknown error during log purge

- Error: 1380 SQLSTATE: HY000 (ER_RELAY_LOG_INIT)

  Message: Failed initializing relay log position: %s

- Error: 1381 SQLSTATE: HY000 (ER_NO_BINARY_LOGGING)

  Message: You are not using binary logging

- Error: 1382 SQLSTATE: HY000 (ER_RESERVED_SYNTAX)

  Message: The '%s' syntax is reserved for purposes internal to the MySQL server

- Error: 1383 SQLSTATE: HY000 (ER_WSAS_FAILED)

  Message: WSAStartup Failed

- Error: 1384 SQLSTATE: HY000 (ER_DIFF_GROUPS_PROC)

  Message: Can't handle procedures with different groups yet

- Error: 1385 SQLSTATE: HY000 (ER_NO_GROUP_FOR_PROC)

  Message: Select must have a group with this procedure

- Error: 1386 SQLSTATE: HY000 (ER_ORDER_WITH_PROC)

  Message: Can't use ORDER clause with this procedure

- Error: 1387 SQLSTATE: HY000 (ER_LOGGING_PROHIBIT_CHANGING_OF)

  Message: Binary logging and replication forbid changing the global server %s

- Error: 1388 SQLSTATE: HY000 (ER_NO_FILE_MAPPING)

  Message: Can't map file: %s, errno: %d

- Error: 1389 SQLSTATE: HY000 (ER_WRONG_MAGIC)

  Message: Wrong magic in %s

- Error: 1390 SQLSTATE: HY000 (ER_PS_MANY_PARAM)

  Message: Prepared statement contains too many placeholders

- Error: 1391 SQLSTATE: HY000 (ER_KEY_PART_0)

  Message: Key part '%s' length cannot be 0

- Error: 1392 SQLSTATE: HY000 (ER_VIEW_CHECKSUM)

  Message: View text checksum failed

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1393 SQLSTATE: HY000 (ER_VIEW_MULTIUPDATE)

  Message: Can not modify more than one base table through a join view '%s.%s'

- Error: 1394 SQLSTATE: HY000 (ER_VIEW_NO_INSERT_FIELD_LIST)

  Message: Can not insert into join view '%s.%s' without fields list

- Error: 1395 SQLSTATE: HY000 (ER_VIEW_DELETE_MERGE_VIEW)

  Message: Can not delete from join view '%s.%s'

- Error: 1396 SQLSTATE: HY000 (ER_CANNOT_USER)

  Message: Operation %s failed for %s

- Error: 1397 SQLSTATE: XAE04 (ER_XAER_NOTA)

  Message: XAER_NOTA: Unknown XID

- Error: 1398 SQLSTATE: XAE05 (ER_XAER_INVAL)

  Message: XAER_INVAL: Invalid arguments (or unsupported command)

- Error: 1399 SQLSTATE: XAE07 (ER_XAER_RMFAIL)

  Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state

- Error: 1400 SQLSTATE: XAE09 (ER_XAER_OUTSIDE)

  Message: XAER_OUTSIDE: Some work is done outside global transaction

- Error: 1401 SQLSTATE: XAE03 (ER_XAER_RMERR)

  Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency

- Error: 1402 SQLSTATE: XA100 (ER_XA_RBROLLBACK)

  Message: XA_RBROLLBACK: Transaction branch was rolled back

- Error: 1403 SQLSTATE: 42000 (ER_NONEXISTING_PROC_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'

- Error: 1404 SQLSTATE: HY000 (ER_PROC_AUTO_GRANT_FAIL)

  Message: Failed to grant EXECUTE and ALTER ROUTINE privileges

- Error: 1405 SQLSTATE: HY000 (ER_PROC_AUTO_REVOKE_FAIL)

  Message: Failed to revoke all privileges to dropped routine

- Error: 1406 SQLSTATE: 22001 (ER_DATA_TOO_LONG)

  Message: Data too long for column '%s' at row %ld

- Error: 1407 SQLSTATE: 42000 (ER_SP_BAD_SQLSTATE)

  Message: Bad SQLSTATE: '%s'

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1408 SQLSTATE: HY000 (ER_STARTUP)

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s

- Error: 1409 SQLSTATE: HY000 (ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR)

  Message: Can't load value from file with fixed size rows to variable

- Error: 1410 SQLSTATE: 42000 (ER_CANT_CREATE_USER_WITH_GRANT)

  Message: You are not allowed to create a user with GRANT

- Error: 1411 SQLSTATE: HY000 (ER_WRONG_VALUE_FOR_TYPE)

  Message: Incorrect %s value: '%s' for function %s

- Error: 1412 SQLSTATE: HY000 (ER_TABLE_DEF_CHANGED)

  Message: Table definition has changed, please retry transaction

- Error: 1413 SQLSTATE: 42000 (ER_SP_DUP_HANDLER)

  Message: Duplicate handler declared in the same block

- Error: 1414 SQLSTATE: 42000 (ER_SP_NOT_VAR_ARG)

  Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger

- Error: 1415 SQLSTATE: 0A000 (ER_SP_NO_RETSET_IN_FUNC)

  Message: Not allowed to return a result set from a function

  In 5.0.12: ER_SP_NO_RETSET_IN_FUNC was renamed to ER_SP_NO_RETSET.

  ER_SP_NO_RETSET_IN_FUNC was removed after 5.0.11.

- Error: 1415 SQLSTATE: 0A000 (ER_SP_NO_RETSET)

  Message: Not allowed to return a result set from a %s

  In 5.0.12: ER_SP_NO_RETSET_IN_FUNC was renamed to ER_SP_NO_RETSET.

  ER_SP_NO_RETSET was added in 5.0.12.

- Error: 1416 SQLSTATE: 22003 (ER_CANT_CREATE_GEOMETRY_OBJECT)

  Message: Cannot get geometry object from data you send to the GEOMETRY field

- Error: 1417 SQLSTATE: HY000 (ER_FAILED_ROUTINE_BREAK_BINLOG)

  Message: A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes

- Error: 1418 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_ROUTINE)

  Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1419 SQLSTATE: HY000 (ER_BINLOG_CREATE_ROUTINE_NEED_SUPER)

  Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error: 1420 SQLSTATE: HY000 (ER_EXEC_STMT_WITH_OPEN_CURSOR)

  Message: You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.

- Error: 1421 SQLSTATE: HY000 (ER_STMT_HAS_NO_OPEN_CURSOR)

  Message: The statement (%lu) has no open cursor.

- Error: 1422 SQLSTATE: HY000 (ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG)

  Message: Explicit or implicit commit is not allowed in stored function or trigger.

- Error: 1423 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_VIEW_FIELD)

  Message: Field of view '%s.%s' underlying table doesn't have a default value

  ER_NO_DEFAULT_FOR_VIEW_FIELD was added in 5.0.9.

- Error: 1424 SQLSTATE: HY000 (ER_SP_NO_RECURSION)

  Message: Recursive stored functions and triggers are not allowed.

  ER_SP_NO_RECURSION was added in 5.0.9.

- Error: 1425 SQLSTATE: 42000 (ER_TOO_BIG_SCALE)

  Message: Too big scale %lu specified for column '%s'. Maximum is %d.

  ER_TOO_BIG_SCALE was added in 5.0.10.

- Error: 1426 SQLSTATE: 42000 (ER_TOO_BIG_PRECISION)

  Message: Too big precision %lu specified for column '%s'. Maximum is %lu.

  ER_TOO_BIG_PRECISION was added in 5.0.10.

- Error: 1427 SQLSTATE: 42000 (ER_SCALE_BIGGER_THAN_PRECISION)

  Message: Scale may not be larger than the precision (column '%s').

  In 5.0.14: ER_SCALE_BIGGER_THAN_PRECISION was renamed to ER_M_BIGGER_THAN_D.

  ER_SCALE_BIGGER_THAN_PRECISION was added in 5.0.10, removed after 5.0.13.

- Error: 1427 SQLSTATE: 42000 (ER_M_BIGGER_THAN_D)

  Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

  In 5.0.14: ER_SCALE_BIGGER_THAN_PRECISION was renamed to ER_M_BIGGER_THAN_D.

  ER_M_BIGGER_THAN_D was added in 5.0.14.

- Error: 1428 SQLSTATE: HY000 (ER_WRONG_LOCK_OF_SYSTEM_TABLE)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: You can't combine write-locking of system '%s.%s' table with other tables

ER_WRONG_LOCK_OF_SYSTEM_TABLE was added in 5.0.10.

- Error: 1429 SQLSTATE: HY000 (ER_CONNECT_TO_FOREIGN_DATA_SOURCE)

  Message: Unable to connect to foreign data source: %s

  ER_CONNECT_TO_FOREIGN_DATA_SOURCE was added in 5.0.10.

- Error: 1430 SQLSTATE: HY000 (ER_QUERY_ON_FOREIGN_DATA_SOURCE)

  Message: There was a problem processing the query on the foreign data source. Data source error: %s

  ER_QUERY_ON_FOREIGN_DATA_SOURCE was added in 5.0.10.

- Error: 1431 SQLSTATE: HY000 (ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST)

  Message: The foreign data source you are trying to reference does not exist. Data source error: %s

  ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST was added in 5.0.10.

- Error: 1432 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE)

  Message: Can't create federated table. The data source connection string '%s' is not in the correct format

  ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE was added in 5.0.10.

- Error: 1433 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID)

  Message: The data source connection string '%s' is not in the correct format

  ER_FOREIGN_DATA_STRING_INVALID was added in 5.0.10.

- Error: 1434 SQLSTATE: HY000 (ER_CANT_CREATE_FEDERATED_TABLE)

  Message: Can't create federated table. Foreign data src error: %s

  ER_CANT_CREATE_FEDERATED_TABLE was added in 5.0.10.

- Error: 1435 SQLSTATE: HY000 (ER_TRG_IN_WRONG_SCHEMA)

  Message: Trigger in wrong schema

  ER_TRG_IN_WRONG_SCHEMA was added in 5.0.10.

- Error: 1436 SQLSTATE: HY000 (ER_STACK_OVERRUN_NEED_MORE)

  Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld -O thread_stack=#' to specify a bigger stack.

  ER_STACK_OVERRUN_NEED_MORE was added in 5.0.11.

- Error: 1437 SQLSTATE: 42000 (ER_TOO_LONG_BODY)

  Message: Routine body for '%s' is too long

  ER_TOO_LONG_BODY was added in 5.0.11.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Error: 1438 SQLSTATE: HY000 (ER_WARN_CANT_DROP_DEFAULT_KEYCACHE)

  Message: Cannot drop default keycache

  ER_WARN_CANT_DROP_DEFAULT_KEYCACHE was added in 5.0.12.

- Error: 1439 SQLSTATE: 42000 (ER_TOO_BIG_DISPLAYWIDTH)

  Message: Display width out of range for column '%s' (max = %lu)

  ER_TOO_BIG_DISPLAYWIDTH was added in 5.0.12.

- Error: 1440 SQLSTATE: XAE08 (ER_XAER_DUPID)

  Message: XAER_DUPID: The XID already exists

  ER_XAER_DUPID was added in 5.0.12.

- Error: 1441 SQLSTATE: 22008 (ER_DATETIME_FUNCTION_OVERFLOW)

  Message: Datetime function: %s field overflow

  ER_DATETIME_FUNCTION_OVERFLOW was added in 5.0.12.

- Error: 1442 SQLSTATE: HY000 (ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG)

  Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.

  ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG was added in 5.0.12.

- Error: 1443 SQLSTATE: HY000 (ER_VIEW_PREVENT_UPDATE)

  Message: The definition of table '%s' prevents operation %s on table '%s'.

  ER_VIEW_PREVENT_UPDATE was added in 5.0.13.

- Error: 1444 SQLSTATE: HY000 (ER_PS_NO_RECURSION)

  Message: The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner

  ER_PS_NO_RECURSION was added in 5.0.13.

- Error: 1445 SQLSTATE: HY000 (ER_SP_CANT_SET_AUTOCOMMIT)

  Message: Not allowed to set autocommit from a stored function or trigger

  ER_SP_CANT_SET_AUTOCOMMIT was added in 5.0.13.

- Error: 1446 SQLSTATE: HY000 (ER_NO_VIEW_USER)

  Message: View definer is not fully qualified

  In 5.0.17: ER_NO_VIEW_USER was renamed to ER_MALFORMED_DEFINER.

  ER_NO_VIEW_USER was added in 5.0.13, removed after 5.0.16.

- Error: 1446 SQLSTATE: HY000 (ER_MALFORMED_DEFINER)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Definer is not fully qualified

In 5.0.17: ER_NO_VIEW_USER was renamed to ER_MALFORMED_DEFINER.

ER_MALFORMED_DEFINER was added in 5.0.17.

- Error: 1447 SQLSTATE: HY000 (ER_VIEW_FRM_NO_USER)

  Message: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!

  ER_VIEW_FRM_NO_USER was added in 5.0.13.

- Error: 1448 SQLSTATE: HY000 (ER_VIEW_OTHER_USER)

  Message: You need the SUPER privilege for creation view with '%s'@'%s' definer

  ER_VIEW_OTHER_USER was added in 5.0.13.

- Error: 1449 SQLSTATE: HY000 (ER_NO_SUCH_USER)

  Message: There is no '%s'@'%s' registered

  ER_NO_SUCH_USER was added in 5.0.13.

- Error: 1450 SQLSTATE: HY000 (ER_FORBID_SCHEMA_CHANGE)

  Message: Changing schema from '%s' to '%s' is not allowed.

  ER_FORBID_SCHEMA_CHANGE was added in 5.0.14.

- Error: 1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2)

  Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)

  ER_ROW_IS_REFERENCED_2 was added in 5.0.14.

- Error: 1452 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW_2)

  Message: Cannot add or update a child row: a foreign key constraint fails (%s)

  ER_NO_REFERENCED_ROW_2 was added in 5.0.14.

- Error: 1453 SQLSTATE: 42000 (ER_SP_BAD_VAR_SHADOW)

  Message: Variable '%s' must be quoted with `...`, or renamed

  ER_SP_BAD_VAR_SHADOW was added in 5.0.15.

- Error: 1454 SQLSTATE: HY000 (ER_TRG_NO_DEFINER)

  Message: No definer attribute for trigger '%s'.'%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.

  ER_TRG_NO_DEFINER was added in 5.0.17.

- Error: 1455 SQLSTATE: HY000 (ER_OLD_FILE_FORMAT)

  Message: '%s' has an old format, you should re-create the '%s' object(s)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ER_OLD_FILE_FORMAT was added in 5.0.17.

- Error: 1456 SQLSTATE: HY000 (ER_SP_RECURSION_LIMIT)

  Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

  ER_SP_RECURSION_LIMIT was added in 5.0.17.

- Error: 1457 SQLSTATE: HY000 (ER_SP_PROC_TABLE_CORRUPT)

  Message: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)

  ER_SP_PROC_TABLE_CORRUPT was added in 5.0.17.

- Error: 1458 SQLSTATE: 42000 (ER_SP_WRONG_NAME)

  Message: Incorrect routine name '%s'

  ER_SP_WRONG_NAME was added in 5.0.19.

- Error: 1459 SQLSTATE: HY000 (ER_TABLE_NEEDS_UPGRADE)

  Message: Table upgrade required. Please do "REPAIR TABLE `%s`" to fix it!

  ER_TABLE_NEEDS_UPGRADE was added in 5.0.19.

- Error: 1460 SQLSTATE: 42000 (ER_SP_NO_AGGREGATE)

  Message: AGGREGATE is not supported for stored functions

  ER_SP_NO_AGGREGATE was added in 5.0.19.

- Error: 1461 SQLSTATE: 42000 (ER_MAX_PREPARED_STMT_COUNT_REACHED)

  Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

  ER_MAX_PREPARED_STMT_COUNT_REACHED was added in 5.0.21.

- Error: 1462 SQLSTATE: HY000 (ER_VIEW_RECURSIVE)

  Message: `%s`.`%s` contains view recursion

  ER_VIEW_RECURSIVE was added in 5.0.21.

- Error: 1463 SQLSTATE: 42000 (ER_NON_GROUPING_FIELD_USED)

  Message: non-grouping field '%s' is used in %s clause

  ER_NON_GROUPING_FIELD_USED was added in 5.0.23.

- Error: 1464 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_SPKEYS)

  Message: The used table type doesn't support SPATIAL indexes

  ER_TABLE_CANT_HANDLE_SPKEYS was added in 5.0.23.

- Error: 1465 SQLSTATE: HY000 (ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Triggers can not be created on system tables

ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA was added in 5.0.23.

- Error: 1466 SQLSTATE: HY000 (ER_REMOVED_SPACES)

  Message: Leading spaces are removed from name '%s'

  ER_REMOVED_SPACES was added in 5.0.25.

- Error: 1467 SQLSTATE: HY000 (ER_AUTOINC_READ_FAILED)

  Message: Failed to read auto-increment value from storage engine

  ER_AUTOINC_READ_FAILED was added in 5.0.26.

- Error: 1468 SQLSTATE: HY000 (ER_USERNAME)

  Message: user name

  ER_USERNAME was added in 5.0.24.

- Error: 1469 SQLSTATE: HY000 (ER_HOSTNAME)

  Message: host name

  ER_HOSTNAME was added in 5.0.24.

- Error: 1470 SQLSTATE: HY000 (ER_WRONG_STRING_LENGTH)

  Message: String '%s' is too long for %s (should be no longer than %d)

  ER_WRONG_STRING_LENGTH was added in 5.0.24.

- Error: 1471 SQLSTATE: HY000 (ER_NON_INSERTABLE_TABLE)

  Message: The target table %s of the %s is not insertable-into

  ER_NON_INSERTABLE_TABLE was added in 5.0.26.

- Error: 1472 SQLSTATE: HY000 (ER_ADMIN_WRONG_MRG_TABLE)

  Message: Table '%s' is differently defined or of non-MyISAM type or doesn't exist

  ER_ADMIN_WRONG_MRG_TABLE was added in 5.0.46.

- Error: 1473 SQLSTATE: HY000 (ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT)

  Message: Too high level of nesting for select

  ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT was added in 5.0.48.

- Error: 1474 SQLSTATE: HY000 (ER_NAME_BECOMES_EMPTY)

  Message: Name '%s' has become ''

  ER_NAME_BECOMES_EMPTY was added in 5.0.50.

- Error: 1475 SQLSTATE: HY000 (ER_AMBIGUOUS_FIELD_TERM)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY

`ER_AMBIGUOUS_FIELD_TERM` was added in 5.0.52.

- Error: `1476` SQLSTATE: `HY000` (`ER_LOAD_DATA_INVALID_COLUMN`)

  Message: Invalid column reference (%s) in LOAD DATA

  `ER_LOAD_DATA_INVALID_COLUMN` was added in 5.0.60.

- Error: `1477` SQLSTATE: `HY000` (`ER_LOG_PURGE_NO_FILE`)

  Message: Being purged log %s was not found

  `ER_LOG_PURGE_NO_FILE` was added in 5.0.60.

- Error: `1478` SQLSTATE: `XA106` (`ER_XA_RBTIMEOUT`)

  Message: XA_RBTIMEOUT: Transaction branch was rolled back: took too long

  `ER_XA_RBTIMEOUT` was added in 5.0.72.

- Error: `1479` SQLSTATE: `XA102` (`ER_XA_RBDEADLOCK`)

  Message: XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected

  `ER_XA_RBDEADLOCK` was added in 5.0.72.

- Error: `1480` SQLSTATE: `HY000` (`ER_TOO_MANY_CONCURRENT_TRXS`)

  Message: Too many active concurrent transactions

  `ER_TOO_MANY_CONCURRENT_TRXS` was added in 5.0.85.

B.4 Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the `include/errmsg.h` MySQL source file.
- The Message values correspond to the error messages that are listed in the `libmysql/errmsg.c` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: `2000` (`CR_UNKNOWN_ERROR`)

  Message: Unknown MySQL error

- Error: `2001` (`CR_SOCKET_CREATE_ERROR`)

  Message: Can't create UNIX socket (%d)

- Error: `2002` (`CR_CONNECTION_ERROR`)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: 2003 (CR_CONN_HOST_ERROR)

  Message: Can't connect to MySQL server on '%s' (%d)

- Error: 2004 (CR_IPSOCK_ERROR)

  Message: Can't create TCP/IP socket (%d)

- Error: 2005 (CR_UNKNOWN_HOST)

  Message: Unknown MySQL server host '%s' (%d)

- Error: 2006 (CR_SERVER_GONE_ERROR)

  Message: MySQL server has gone away

- Error: 2007 (CR_VERSION_ERROR)

  Message: Protocol mismatch; server version = %d, client version = %d

- Error: 2008 (CR_OUT_OF_MEMORY)

  Message: MySQL client ran out of memory

- Error: 2009 (CR_WRONG_HOST_INFO)

  Message: Wrong host info

- Error: 2010 (CR_LOCALHOST_CONNECTION)

  Message: Localhost via UNIX socket

- Error: 2011 (CR_TCP_CONNECTION)

  Message: %s via TCP/IP

- Error: 2012 (CR_SERVER_HANDSHAKE_ERR)

  Message: Error in server handshake

- Error: 2013 (CR_SERVER_LOST)

  Message: Lost connection to MySQL server during query

- Error: 2014 (CR_COMMANDS_OUT_OF_SYNC)

  Message: Commands out of sync; you can't run this command now

- Error: 2015 (CR_NAMEDPIPE_CONNECTION)

  Message: Named pipe: %s

- Error: 2016 (CR_NAMEDPIPEWAIT_ERROR)

  Message: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error: 2017 (CR_NAMEDPIPEOPEN_ERROR)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: 2018 (CR_NAMEDPIPESETSTATE_ERROR)

  Message: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error: 2019 (CR_CANT_READ_CHARSET)

  Message: Can't initialize character set %s (path: %s)

- Error: 2020 (CR_NET_PACKET_TOO_LARGE)

  Message: Got packet bigger than 'max_allowed_packet' bytes

- Error: 2021 (CR_EMBEDDED_CONNECTION)

  Message: Embedded server

- Error: 2022 (CR_PROBE_SLAVE_STATUS)

  Message: Error on SHOW SLAVE STATUS:

- Error: 2023 (CR_PROBE_SLAVE_HOSTS)

  Message: Error on SHOW SLAVE HOSTS:

- Error: 2024 (CR_PROBE_SLAVE_CONNECT)

  Message: Error connecting to slave:

- Error: 2025 (CR_PROBE_MASTER_CONNECT)

  Message: Error connecting to master:

- Error: 2026 (CR_SSL_CONNECTION_ERROR)

  Message: SSL connection error

- Error: 2027 (CR_MALFORMED_PACKET)

  Message: Malformed packet

- Error: 2028 (CR_WRONG_LICENSE)

  Message: This client library is licensed only for use with MySQL servers having '%s' license

- Error: 2029 (CR_NULL_POINTER)

  Message: Invalid use of null pointer

- Error: 2030 (CR_NO_PREPARE_STMT)

  Message: Statement not prepared

- Error: 2031 (CR_PARAMS_NOT_BOUND)

  Message: No data supplied for parameters in prepared statement

- Error: 2032 (CR_DATA_TRUNCATED)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Data truncated

- Error: 2033 (CR_NO_PARAMETERS_EXISTS)

  Message: No parameters exist in the statement

- Error: 2034 (CR_INVALID_PARAMETER_NO)

  Message: Invalid parameter number

  The column number for `mysql_stmt_fetch_column()` was invalid.

  The parameter number for `mysql_stmt_send_long_data()` was invalid.

- Error: 2035 (CR_INVALID_BUFFER_USE)

  Message: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error: 2036 (CR_UNSUPPORTED_PARAM_TYPE)

  Message: Using unsupported buffer type: %d (parameter: %d)

- Error: 2037 (CR_SHARED_MEMORY_CONNECTION)

  Message: Shared memory: %s

- Error: 2038 (CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR)

  Message: Can't open shared memory; client could not create request event (%lu)

- Error: 2039 (CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR)

  Message: Can't open shared memory; no answer event received from server (%lu)

- Error: 2040 (CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR)

  Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: 2041 (CR_SHARED_MEMORY_CONNECT_MAP_ERROR)

  Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: 2042 (CR_SHARED_MEMORY_FILE_MAP_ERROR)

  Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: 2043 (CR_SHARED_MEMORY_MAP_ERROR)

  Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: 2044 (CR_SHARED_MEMORY_EVENT_ERROR)

  Message: Can't open shared memory; client could not create %s event (%lu)

- Error: 2045 (CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR)

  Message: Can't open shared memory; no answer from server (%lu)

- Error: 2046 (CR_SHARED_MEMORY_CONNECT_SET_ERROR)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Message: Can't open shared memory; cannot send request event to server (%lu)

- Error: 2047 (CR_CONN_UNKNOW_PROTOCOL)

  Message: Wrong or unknown protocol

- Error: 2048 (CR_INVALID_CONN_HANDLE)

  Message: Invalid connection handle

- Error: 2049 (CR_SECURE_AUTH)

  Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: 2050 (CR_FETCH_CANCELED)

  Message: Row retrieval was canceled by mysql_stmt_close() call

- Error: 2051 (CR_NO_DATA)

  Message: Attempt to read column without prior row fetch

- Error: 2052 (CR_NO_STMT_METADATA)

  Message: Prepared statement contains no metadata

- Error: 2053 (CR_NO_RESULT_SET)

  Message: Attempt to read a row while there is no result set associated with the statement

- Error: 2054 (CR_NOT_IMPLEMENTED)

  Message: This feature is not implemented yet

- Error: 2055 (CR_SERVER_LOST_EXTENDED)

  Message: Lost connection to MySQL server at '%s', system error: %d

  CR_SERVER_LOST_EXTENDED was added in 5.0.32.

B.5 Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

B.5.1 How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:

  - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

  - The mouse pointer does not move.
  - The machine does not answer to a remote machine's pings.
  - Other programs that are not related to MySQL do not behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

  In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

  It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.4, “MySQL Server Logs”.
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system does not work.” This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

This documentation is for an older version. If you're
This documentation is for an older version. If you're

When sending a bug report, you should follow the outline described in Section 1.7, “How to Report Bugs or Problems”.

B.5.2 Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

B.5.2.1 Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See Section 6.2, “The MySQL Access Privilege System”, and Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”.

B.5.2.2 Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the file system (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you do not specify a host name or if you specify the special host name `localhost`.

If the MySQL server is running on Windows, you can connect using TCP/IP. If the server is started with the `--enable-named-pipe` option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you do not give a host name when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that does not work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the host name.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) `Can't connect to MySQL server on 'server' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.18.2, “Starting the Server”.

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. `host_ip` represents the IP address of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotation marks with the `hostname` command; these cause the output of `hostname` (that is, the current host name) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the host name of

This documentation is for an older version. If you're
This documentation is for an older version. If you're

your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking`, it will not accept TCP/IP connections at all. If the server was started with `--bind-address=127.0.0.1`, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running on the local host. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: Connection to MySQL Server Failing on Windows.
- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You have started the `mysqld` server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

  See Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill` or with the `mysql_zap` script) before you can restart the MySQL server. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”.
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet:`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`Unable to connect to remote host: Connection refused`, then there is no server running on the given port.

- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of `TIME_WAIT` is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The `TIME_WAIT` delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce `TIME_WAIT` greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

Windows permits ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. The port will not be available again until this time expires. The default range of port numbers depends on the version of Windows, with a more limited number of ports in older versions:

- Windows through Server 2003: Ports in range 1025–5000
- Windows Vista, Server 2008, and newer: Ports in range 49152–65535

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

Important

The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore it if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: http://support.microsoft.com/kb/256986/EN-US/.

1. Start Registry Editor (`Regedt32.exe`).
2. Locate the following key in the registry:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

3. On the Edit menu, click Add Value, and then add the following registry value:

```
Value Name: MaxUserPort
Data Type: REG_DWORD
Value: 65534
```

   This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the Edit menu, click Add Value, and then add the following registry value:

```
Value Name: TcpTimedWaitDelay
Data Type: REG_DWORD
Value: 30
```

   This sets the number of seconds to hold a TCP port connection in TIME_WAIT state before closing. The valid range is between 30 and 300 decimal, although you may wish to check with Microsoft for the latest permitted values. The default value is 0x78 (120 decimal).

5. Quit Registry Editor.

6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

B.5.2.3 Lost connection to MySQL server

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing net_read_timeout from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your connect_timeout value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using SHOW GLOBAL STATUS LIKE 'Aborted_connects'. It will increase by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with BLOB values that are larger than max_allowed_packet, which can cause this error with some clients. Sometime you may see an ER_NET_PACKET_TOO_LARGE error, and that confirms that you need to increase max_allowed_packet.

B.5.2.4 Client does not support authentication protocol

The current implementation of the authentication protocol uses a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. Attempts to connect to a 4.1 or newer server with an older client may fail with the following message:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To deal with this problem, the preferred solution is to upgrade all client programs to use a 4.1.1 or newer client library. If that is not possible, use one of the following approaches:

- To connect to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the `SET PASSWORD` statement and the `OLD_PASSWORD()` function:

  ```
  mysql> SET PASSWORD FOR
      -> 'some_user'@'some_host' = OLD_PASSWORD('new_password');
  ```

  Substitute the password you want to use for "*new_password*" in the preceding example. MySQL cannot tell you what the original password was, so you'll need to pick a new one.
- Tell the server to use the older password hashing algorithm by default:

  1. Start `mysqld` with the `old_passwords` system variable set to 1.
  2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

     ```
     mysql> SELECT Host, User, Password FROM mysql.user
         -> WHERE LENGTH(Password) > 16;
     ```

     For each account record displayed by the query, use the `Host` and `User` values and assign a password using one of the methods described previously.

The `Client does not support authentication protocol` error also can occur if multiple versions of MySQL are installed but client programs are dynamically linked and link to an older library. Make sure that clients use the most recent library version with which they are compatible. The procedure to do this will depend on your system.

Note

The PHP `mysql` extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or newer, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysqli` extension (stands for "MySQL, Improved"; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the `mysqli` extension, see http://php.net/mysqli.

For additional background on password hashing and authentication, see Section 6.1.2.4, "Password Hashing in MySQL".

B.5.2.5 Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

B.5.2.6 Host 'host_name' is blocked

If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The value of the `max_connect_errors` system variable determines how many successive interrupted connection requests are permitted. (See Section 5.1.4, “Server System Variables”.) After `max_connect_errors` failed requests without a successful connection, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you issue a `FLUSH HOSTS` statement or execute a `mysqladmin flush-hosts` command.

By default, `mysqld` blocks a host after 10 connection errors. You can adjust the value by setting `max_connect_errors` at server startup:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

The value can also be set at runtime:

```
mysql> SET GLOBAL max_connect_errors=10000;
```

If you get the `Host 'host_name' is blocked` error message for a given host, you should first verify that there is nothing wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` variable.

B.5.2.7 Too many connections

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that all available connections are in use by other clients.

The number of connections permitted is controlled by the `max_connections` system variable. Its default value is 100. If you need to support more connections, you should set a larger value for this variable.

`mysqld` actually permits `max_connections+1` clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the `SUPER` privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.5.27, “SHOW PROCESSLIST Syntax”.

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform, the amount of RAM available, how much RAM is used for each connection, the workload from each connection, and the desired response time. Linux or Solaris should be able to support at 500 to

This documentation is for an older version. If you're
This documentation is for an older version. If you're

1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding. Windows is limited to (open tables × 2 + open connections) < 2048 due to the Posix compatibility layer used on that platform.

Increasing `open-files-limit` may be necessary. Also see Section 2.20.1.4, “Linux Postinstallation Notes”, for how to raise the operating system limit on how many handles can be used by MySQL.

B.5.2.8 Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

B.5.2.9 MySQL server has gone away

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

Error Code	Description
`CR_SERVER_GONE_ERROR`	The client couldn't send a question to the server.
`CR_SERVER_LOST`	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start `mysqld`. See Section 5.1.4, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.

- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).

- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

  The problem on Windows is that in some cases MySQL does not get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

  Prior to MySQL 5.0.19, even if the `reconnect` flag in the `MYSQL` structure is equal to 1, MySQL does not automatically reconnect and re-issue the query as it doesn't know if the server did get the original query or not.

  The solution to this is to either do a `mysql_ping()` on the connection if there has been a long time since the last query (this is what Connector/ODBC does) or set `wait_timeout` on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section B.5.2.10, “Packet Too Large”.

  An `INSERT` or `REPLACE` statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per `INSERT` or `REPLACE`.

- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.

- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

  You may also see the `MySQL server has gone away` error if MySQL is started with the `--skip-networking` option.

  Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.

- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can get more information about the lost connections by starting `mysqld` with the `--log-warnings=2` option. This logs some of the disconnected errors in the `hostname.err` file. See Section 5.4.1, “The Error Log”.

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See Section B.5.3.3, “What to Do If MySQL Keeps Crashing”.
- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See Section 21.3, “Debugging and Porting MySQL”.
- What is the value of the `wait_timeout` system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)
- Have you tried to run `mysqld` with the general query log enabled to determine whether the problem query appears in the log? (See Section 5.4.2, “The General Query Log”.)

See also Section B.5.2.11, “Communication Errors and Aborted Connections”, and Section 1.7, “How to Report Bugs or Problems”.

B.5.2.10 Packet Too Large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

The largest possible packet that can be transmitted to or from a MySQL 5.0 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` bytes, it issues an `ER_NET_PACKET_TOO_LARGE` error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

B.5.2.11 Communication Errors and Aborted Connections

If connection problems occur such as communication errors or aborted connections, use these sources of information to diagnose problems:

- The error log. See Section 5.4.1, “The Error Log”.
- The general query log. See Section 5.4.2, “The General Query Log”.
- The `Aborted_xxx` status variables. See Section 5.1.6, “Server Status Variables”.

If you start the server with the `--log-warnings` option, you might find messages like this in your error log:

```
Aborted connection 854 to db: 'employees' user: 'josh'
```

If a client is unable even to connect, the server increments the `Aborted_connects` status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client attempts to access a database but has no privileges for it.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout` seconds to obtain a connect packet. See Section 5.1.4, “Server System Variables”.

If these kinds of things happen, it might indicate that someone is trying to break into your server! If the general query log is enabled, messages for these types of problems are logged to it.

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients` status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See Section 5.1.4, “Server System Variables”.
- The client program ended abruptly in the middle of a data transfer.

Other reasons for problems with aborted connections or aborted clients:

- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See Section B.5.2.10, “Packet Too Large”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Use of Ethernet protocol with Linux, both half and full duplex. Some Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also Section B.5.2.9, “MySQL server has gone away”.

B.5.2.12 The table is full

If a table-full error occurs, it may be that the disk is full or that the table has reached its maximum size. The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. See Section C.7.3, “Limits on Table Size”.

This error can occur sometimes for MySQL Cluster tables even when there appears to be more than sufficient data memory available. See the documentation for the `DataMemory` MySQL Cluster data node configuration parameter, as well as Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”, for more information.

B.5.2.13 Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See Section 4.2.6, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
shell> perror 28
OS error code  28:  No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Provided that the write error is to a test file, the error is not serious and can be safely ignored.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.5.2.14 Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

B.5.2.15 Ignoring user

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

`Found wrong password for user '`*`some_user`*`'@'`*`some_host`*`'; ignoring user`

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see whether the `Password` column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The account has an old password (eight characters long). Update the account in the `user` table to have a new password.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use `mysql` to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` function:

```
mysql> UPDATE user SET Password=PASSWORD('new_password')
    -> WHERE User='some_user' AND Host='some_host';
```

B.5.2.16 Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See Section 13.7.5, “SHOW Syntax”.

B.5.2.17 Can't initialize character set

This documentation is for an older version. If you're

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multibyte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `configure` with the `--with-charset=charset_name` or `--with-extra-charsets=charset_name` option. See Section 2.17.3, “MySQL Source-Configuration Options”.

  All standard MySQL binaries are compiled with `--with-extra-charsets=complex` or (for Windows) `--with-extra-charsets=complex`, which enables support for all multibyte character sets. See Section 2.17.3, “MySQL Source-Configuration Options”.

- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.17.3, “MySQL Source-Configuration Options”.
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
  - Copy the character definition files to the path where the client expects them to be.

B.5.2.18 File Not Found and Similar Errors

If you get `ERROR 'file_name' not found (errno: 23)`, `Can't open file: file_name (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you have not allocated enough file descriptors for the MySQL server. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
OS error code  23:  File table overflow
shell> perror 24
OS error code  24:  Too many open files
shell> perror 11
OS error code  11:  Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_cache` system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in Section 8.4.3.1, “How MySQL Opens and Closes Tables”. Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` option to `mysqld_safe` or set the `open_files_limit` system variable. See Section 5.1.4, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.2.6, “Using Option Files”. If you have an old version of `mysqld` that does not support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the script. You can remove the "#" character to uncomment this line, and change the number 256 to set the number of file descriptors to be made available to mysqld.

--open-files-limit and ulimit can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a "hard" limit that can be overridden only if you start mysqld_safe or mysqld as root (just remember that you also need to start the server with the --user option in this case so that it does not continue to run as root after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the tcsh shell, ulimit does not work! tcsh also reports incorrect values when you ask for the current limits. In this case, you should start mysqld_safe using sh.

B.5.2.19 Table-Corruption Issues

If you have started mysqld with --myisam-recover, MySQL automatically checks and tries to repair MyISAM tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the hostname.err file 'Warning: Checking table ...' which is followed by Warning: Repairing table if the table needs to be repaired. If you get a lot of these errors, without mysqld having died unexpectedly just before, then something is wrong and needs to be investigated further.

See also Section 5.1.3, "Server Command Options", and Section 21.3.1.7, "Making a Test Case If You Experience Table Corruption".

B.5.3 Administration-Related Issues

B.5.3.1 Problems with File Permissions

If you have problems with file permissions, the UMASK or UMASK_DIR environment variable might be set incorrectly when mysqld starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default UMASK and UMASK_DIR values are 0660 and 0700, respectively. MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero. For example, setting UMASK=0600 is equivalent to UMASK=384 because 0600 octal is 384 decimal.

To change the default UMASK value, start mysqld_safe as follows:

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database and RAID directories with an access permission value of 0700. To modify this behavior, set the UMASK_DIR variable. If you set its value, new directories are created with the combined UMASK and UMASK_DIR values. For example, to give group access to all new directories, start mysqld_safe as follows:

```
shell> UMASK_DIR=504  # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

For additional details, see Section 2.21, “Environment Variables”.

B.5.3.2 How to Reset the Root Password

If you have never assigned a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning passwords, see Section 2.18.4, “Securing the Initial MySQL Accounts”.

If you know the `root` password and want to change it, see Section 13.7.1.6, “SET PASSWORD Syntax”.

If you assigned a `root` password previously but have forgotten it, you can assign a new password. The following sections provide instructions for Windows and Unix and Unix-like systems, as well as generic instructions that apply to any system.

Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list and stop it.

   If your server is not running as a service, you may need to use the Task Manager to force it to stop.
3. Create a text file containing the following statement on a single line. Replace the password with the password that you want to use.

   ```
   SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');
   ```
4. Save the file. This example assumes that you name the file `C:\mysql-init.txt`.
5. Open a console window to get to the command prompt: From the **Start** menu, select **Run**, then enter `cmd` as the command to be run.
6. Start the MySQL server with the special `--init-file` option (notice that the backslash in the option value is doubled):

   ```
   C:\> cd "C:\Program Files\MySQL\MySQL Server 5.0\bin"
   C:\> mysqld-nt --init-file=C:\\mysql-init.txt
   ```

   If you installed MySQL to a different location, adjust the `cd` command accordingly.

   The server executes the contents of the file named by the `--init-file` option at startup, changing the `'root'@'localhost'` account password.

   To have server output to appear in the console window rather than in a log file, add the `--console` option to the `mysqld` command.

   If you installed MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option. For example:

   ```
   C:\> mysqld-nt
   ```


This documentation is for an older version. If you're

```
        --defaults-file="C:\\Program Files\\MySQL\\MySQL Server 5.0\\my.ini"
        --init-file=C:\\mysql-init.txt
```

The appropriate `--defaults-file` setting can be found using the Services Manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server and restart it normally. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

Resetting the Root Password: Unix and Unix-Like Systems

On Unix, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

The instructions assume that you will start the MySQL server from the Unix login account that you normally use for running it. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` option. If you start the server as `root` without using `--user=mysql`, the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you will need to either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the MySQL server runs as (for example, `mysql`).

2. Stop the MySQL server if it is running. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

   Stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process. Use the actual path name of the `.pid` file in the following command:

   ```
   shell> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file containing the following statement on a single line. Replace the password with the password that you want to use.

   ```
   SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');
   ```

4. Save the file. This example assumes that you name the file `/home/me/mysql-init`. The file contains the password, so do not save it where it can be read by other users. If you are not logged in as `mysql` (the user the server runs as), make sure that the file has permissions that permit `mysql` to read it.

5. Start the MySQL server with the special `--init-file` option:

   ```
   shell> mysqld_safe --init-file=/home/me/mysql-init &
   ```


This documentation is for an older version. If you're

The server executes the contents of the file named by the `--init-file` option at startup, changing the `'root'@'localhost'` account password.

6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions specifically for Windows and Unix and Unix-like systems. Alternatively, on any platform, you can reset the password using the `mysql` client (but this approach is less secure):

1. Stop the MySQL server if necessary, then restart it with the `--skip-grant-tables` option. This enables anyone to connect without a password and with all privileges, and disables account-management statements such as `SET PASSWORD`. Because this is insecure, you might want to use `--skip-grant-tables` in conjunction with `--skip-networking` to prevent remote clients from connecting.

2. Connect to the MySQL server using the `mysql` client; no password is necessary because the server was started with `--skip-grant-tables`:

```
shell> mysql
```

3. In the `mysql` client, tell the server to reload the grant tables so that account-management statements work:

```
mysql> FLUSH PRIVILEGES;
```

Then change the `'root'@'localhost'` account password. Replace the password with the password that you want to use. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

```
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');
```

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally (without the `--skip-grant-tables` and `--skip-networking` options).

B.5.3.3 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See Section 5.4.1, “The Error Log”.

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `resolve_stack_dump` program. See Section 21.3, “Debugging and Porting MySQL”. Note that the variable values written in the error log may not always be 100% correct.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `--delay-key-write`, in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all `MyISAM` tables, and restart `mysqld`. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.
- Start `mysqld` with the general query log enabled (see Section 5.4.2, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.4.2, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See Section 1.7, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section 21.3, “Debugging and Porting MySQL”.
- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See Section 21.1.2, “The MySQL Test Suite”. They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening. Reconfigure MySQL with the `--with-debug` or `--with-debug=full` option to `configure` and then recompile. See Section 21.3, “Debugging and Porting MySQL”.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while you run your other queries.
- Try the following:

  1. Start `mysqld` from `gdb` (or another debugger). See Section 21.3, “Debugging and Porting MySQL”.
  2. Run your test scripts.
  3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where *`N`* is the thread ID.
- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See Section 1.7, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to your memory and disk subsystems when troubleshooting hardware.

B.5.3.4 How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

B.5.3.5 Where MySQL Stores Temporary Files

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files. If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, Netware and OS2, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `mysqld --tmpdir` option to specify a directory in a file system where you have enough space. On replication slaves, you can use `--slave-load-tmpdir` to specify a separate directory for holding temporary files when replicating `LOAD DATA INFILE` statements.

The `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (“`:`”) on Unix and semicolon characters (“`;`”) on Windows, NetWare, and OS/2.

Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should be sure to set `--slave-load-tmpdir` not to point to a directory that is on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so

This documentation is for an older version. If you're
This documentation is for an older version. If you're

that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the slave temporary file directory are lost when the server restarts, replication fails.

MySQL arranges that temporary files are removed if `mysqld` is terminated. On platforms that support it (such as Unix), this is done by unlinking the file after opening it. The disadvantage of this is that the name does not appear in directory listings and you do not see a big temporary file that fills up the file system in which the temporary file directory is located. (In such cases, `lsof +L1` may be helpful in identifying large files associated with `mysqld`.)

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some `SELECT` queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form `SQL_*`.

`ALTER TABLE` creates a temporary copy of the original table in the same directory as the original table.

B.5.3.6 How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On some versions of Unix, anyone can delete files in the `/tmp` directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your `/tmp` directory so that files can be deleted only by their owners or the superuser (`root`). To do this, set the `sticky` bit on the `/tmp` directory by logging in as `root` and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission character is `t`, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in `/etc/my.cnf`:

  ```
  [mysqld]
  socket=/path/to/socket

  [client]
  socket=/path/to/socket
  ```

  See Section 4.2.6, “Using Option Files”.

- Specify a `--socket` option on the command line to `mysqld_safe` and when you run client programs.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Set the MYSQL_UNIX_PORT environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the --with-unix-socket-path option when you run configure. See Section 2.17.3, “MySQL Source-Configuration Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

B.5.3.7 Time Zone Problems

If you have a problem with SELECT NOW() returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if UNIX_TIMESTAMP() returns the wrong value. This should be done for the environment in which the server runs; for example, in mysqld_safe or mysql.server. See Section 2.21, “Environment Variables”.

You can set the time zone for the server with the --timezone=*timezone_name* option to mysqld_safe. You can also set it by setting the TZ environment variable before you start mysqld.

The permissible values for --timezone or TZ are system dependent. Consult your operating system documentation to see what values are acceptable.

B.5.4 Query-Related Issues

B.5.4.1 Case Sensitivity in String Searches

For nonbinary strings (CHAR, VARCHAR, TEXT), string searches use the collation of the comparison operands. For binary strings (BINARY, VARBINARY, BLOB), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons will be case sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (>=, >, =, <, <=, sorting, and grouping) are based on each character's “sort value.” Characters with the same sort value are treated as the same character. For example, if “e” and “é” have the same sort value in a given collation, they compare as equal.

The default character set and collation are latin1 and latin1_swedish_ci, so nonbinary string comparisons are case insensitive by default. This means that if you search with *col_name* LIKE 'a%', you get all column values that start with A or a. To make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the latin1 character set, you can use the COLLATE operator to cause either operand to have the latin1_general_cs or latin1_bin collation:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See Section 13.1.10, “CREATE TABLE Syntax”.

To cause a case-sensitive comparison of nonbinary strings to be case insensitive, use COLLATE to name a case-insensitive collation. The strings in the following example normally are case sensitive, but COLLATE changes the comparison to be case insensitive:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SET @s1 = 'MySQL' COLLATE latin1_bin,
    ->     @s2 = 'mysql' COLLATE latin1_bin;
mysql> SELECT @s1 = @s2;
+-----------+
| @s1 = @s2 |
+-----------+
|         0 |
+-----------+
mysql> SELECT @s1 COLLATE latin1_swedish_ci = @s2;
+-------------------------------------+
| @s1 COLLATE latin1_swedish_ci = @s2 |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

A binary string is case sensitive in comparisons. To compare the string as case insensitive, convert it to a nonbinary string and use `COLLATE` to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+--------------+
| @s = 'mysql' |
+--------------+
|            0 |
+--------------+
mysql> SELECT CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql';
+-------------------------------------------------------------------+
| CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql' |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

To determine whether a value will compare as a nonbinary or binary string, use the `COLLATION()` function. This example shows that `VERSION()` returns a string that has a case-insensitive collation, so comparisons are case insensitive:

```
mysql> SELECT COLLATION(VERSION());
+----------------------+
| COLLATION(VERSION()) |
+----------------------+
| utf8_general_ci      |
+----------------------+
```

For binary strings, the collation value is `binary`, so comparisons will be case sensitive. One context in which you will see `binary` is for compression and encryption functions, which return binary strings as a general rule: string:

```
mysql> SELECT COLLATION(ENCRYPT('x')), COLLATION(SHA1('x'));
+--------------------------+----------------------+
| COLLATION(ENCRYPT('x')) | COLLATION(SHA1('x')) |
+--------------------------+----------------------+
| binary                   | binary               |
+--------------------------+----------------------+
```

B.5.4.2 Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is permitted. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a `WHERE` clause that compares a date to a `DATE`, `DATETIME`, or `TIMESTAMP` column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent. MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any comparison method other than those just listed, such as `IN` or `STRCMP()`.

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special “zero” date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When a `'0000-00-00'` date is used through Connector/ODBC, it is automatically converted to `NULL` because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that `idate` is a `DATE` column):

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

However, the following statement does not work:

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in `'YYYY-MM-DD'` format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If you enable the `ALLOW_INVALID_DATES` SQL mode, MySQL permits you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12. This makes MySQL very convenient for Web applications where you obtain year,

This documentation is for an older version. If you're
This documentation is for an older version. If you're

month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

MySQL permits you to store dates where the day or month and day are zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` SQL mode.

MySQL permits you to store a "zero" value of `'0000-00-00'` as a "dummy date." This is in some cases more convenient than using `NULL` values. If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores `'0000-00-00'`. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` SQL mode.

To have MySQL check all dates and accept only legal dates (unless overridden by `IGNORE`), set the `sql_mode` system variable to `"NO_ZERO_IN_DATE,NO_ZERO_DATE"`.

Date handling in MySQL 5.0.1 and earlier works like MySQL 5.0.2 with the `ALLOW_INVALID_DATES` SQL mode enabled.

B.5.4.3 Problems with NULL Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as "phone number is not known" and the meaning of the second can be regarded as "the person is known to have no phone, and thus no phone number."

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

In SQL, the `NULL` value is never true in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

To search for column values that are `NULL`, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` test. The following statements show how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See Section 3.3.4.6, "Working with NULL Values", for additional information and examples.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

You can add an index on a column that can have `NULL` values if you are using the `MyISAM`, `InnoDB`, or `BDB`, or `MEMORY` storage engine. Otherwise, you must declare an indexed column `NOT NULL`, and you cannot insert `NULL` into the column.

When reading data with `LOAD DATA INFILE`, empty or missing columns are updated with `''`. To load a `NULL` value into a column, use `\N` in the data file. The literal word "`NULL`" may also be used under some circumstances. See Section 13.2.6, "LOAD DATA INFILE Syntax".

When using `DISTINCT`, `GROUP BY`, or `ORDER BY`, all `NULL` values are regarded as equal.

When using `ORDER BY`, `NULL` values are presented first, or last if you specify `DESC` to sort in descending order.

Aggregate (summary) functions such as `COUNT()`, `MIN()`, and `SUM()` ignore `NULL` values. The exception to this is `COUNT(*)`, which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-`NULL` values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles `NULL` values specially. If you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted. If you insert `NULL` into an integer or floating-point column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted.

B.5.4.4 Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in `GROUP BY`, `ORDER BY`, or `HAVING` clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
  GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
  GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a `WHERE` clause. This restriction is imposed because when the `WHERE` clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
  WHERE cnt > 0 GROUP BY id;
```

The `WHERE` clause determines which rows should be included in the `GROUP BY` clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the `GROUP BY`.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias `` `a` ``:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY `a`;
```

But this statement groups by the literal string `'a'` and will not work as expected:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY 'a';
```

B.5.4.5 Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 13.7.5.33, “SHOW TABLE STATUS Syntax”, and Section 13.7.5.9, “SHOW CREATE TABLE Syntax”.

You can check which storage engines your `mysqld` server supports by using this statement:

```
SHOW ENGINES;
```

You can also use the following statement, and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the `InnoDB` storage engine is available, check the value of the `have_innodb` variable.

See Section 13.7.5.13, “SHOW ENGINES Syntax”, and Section 13.7.5.36, “SHOW VARIABLES Syntax”.

B.5.4.6 Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` system variable), you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

B.5.4.7 Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See Section 13.8.2, “EXPLAIN Syntax”.

2. Select only those columns that are used in the `WHERE` clause.

3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.

4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.

5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See Section B.5.4.8, “Problems with Floating-Point Values”.

   Similar problems may be encountered when comparing `DECIMAL` values prior to MySQL 5.0.3.

6. If you still cannot figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

   Verify that the test file demonstrates the problem by executing these commands:

   ```
   shell> mysqladmin create test2
   shell> mysql test2 < query.sql
   ```

   Attach the test file to a bug report, which you can file using the instructions in Section 1.7, “How to Report Bugs or Problems”.

B.5.4.8 Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. Before MySQL 5.0.3, `DECIMAL` comparison operations are approximate as well.

Prior to MySQL 5.0.3, `DECIMAL` columns store values with exact precision because they are represented as strings, but calculations on `DECIMAL` values are done using floating-point operations. As of 5.0.6, MySQL performs `DECIMAL` operations with a precision of 65 decimal digits (64 digits from 5.0.3 to 5.0.5), which should solve most common inaccuracy problems when it comes to `DECIMAL` columns. (If your server is from MySQL 5.0.3 or higher, but you have `DECIMAL` columns in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer `DECIMAL` format, dump them with `mysqldump` and reload them.)

The following example (for versions of MySQL older than 5.0.3) demonstrates the problem. It shows that even for older `DECIMAL` columns, calculations that are done using floating-point operations are subject to

This documentation is for an older version. If you're
This documentation is for an older version. If you're

floating-point error. (Were you to replace the `DECIMAL` columns with `FLOAT`, similar problems would occur for all versions of MySQL.)

```
mysql> CREATE TABLE t1 (i INT, d1 DECIMAL(9,2), d2 DECIMAL(9,2));
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+-------+
| i    | a      | b     |
+------+--------+-------+
|    1 |  21.40 | 21.40 |
|    2 |  76.80 | 76.80 |
|    3 |   7.40 |  7.40 |
|    4 |  15.40 | 15.40 |
|    5 |   7.20 |  7.20 |
|    6 | -51.40 |  0.00 |
+------+--------+-------+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

As of MySQL 5.0.3, you will get only the last row in the above result.

The problem cannot be solved by using `ROUND()` or similar functions, because the result is still a floating-point number:

```
mysql> SELECT i, ROUND(SUM(d1), 2) AS a, ROUND(SUM(d2), 2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+-------+
| i    | a      | b     |
+------+--------+-------+
|    1 |  21.40 | 21.40 |
|    2 |  76.80 | 76.80 |
|    3 |   7.40 |  7.40 |
|    4 |  15.40 | 15.40 |
|    5 |   7.20 |  7.20 |
|    6 | -51.40 |  0.00 |
+------+--------+-------+
```

This is what the numbers in column `a` look like when displayed with more decimal places:

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1.0000000000000000 AS a,
    -> ROUND(SUM(d2), 2) AS b FROM t1 GROUP BY i HAVING a <> b;
+------+----------------------+-------+
| i    | a                    | b     |
+------+----------------------+-------+
|    1 |  21.3999999999999986 | 21.40 |
|    2 |  76.7999999999999972 | 76.80 |
|    3 |   7.4000000000000004 |  7.40 |
|    4 |  15.4000000000000004 | 15.40 |
|    5 |   7.2000000000000002 |  7.20 |
|    6 | -51.3999999999999986 |  0.00 |
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
+------+----------------------+-------+
```

Depending on your computer architecture, you may or may not see similar results. For example, on some machines you may get the “correct” results by multiplying both arguments by 1, as the following example shows.

Warning

Never use this method in your applications. It is not an example of a trustworthy method!

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1 AS a, ROUND(SUM(d2), 2)*1 AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+------+
| i    | a      | b    |
+------+--------+------+
|    6 | -51.40 | 0.00 |
+------+--------+------+
```

The reason that the preceding example seems to work is that on the particular machine where the test was done, CPU floating-point arithmetic happens to round the numbers to the same value. However, there is no rule that any CPU should do so, so this method cannot be trusted.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+--------+------+
| i    | a      | b    |
+------+--------+------+
|    6 | -51.40 | 0.00 |
+------+--------+------+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+-------+-------+
| i    | a     | b     |
+------+-------+-------+
|    1 | 21.40 | 21.40 |
|    2 | 76.80 | 76.80 |
|    3 |  7.40 |  7.40 |
|    4 | 15.40 | 15.40 |
|    5 |  7.20 |  7.20 |
+------+-------+-------+
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','-1e+52');
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
SELECT * FROM t1;
```

On some platforms, the `SELECT` statement returns `inf` and `-inf`. On others, it returns `0` and `-0`.

An implication of the preceding issues is that if you attempt to create a replication slave by dumping table contents with `mysqldump` on the master and reloading the dump file into the slave, tables containing floating-point columns might differ between the two hosts.

B.5.5 Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` is discussed in more detail in Section 13.8.2, “EXPLAIN Syntax”.

- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.

- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

  `USE INDEX` and `IGNORE INDEX` may also be useful. See Section 8.9.2, “Index Hints”.

- Global and table-level `STRAIGHT_JOIN`. See Section 13.2.8, “SELECT Syntax”.

- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.4, “Server System Variables”.

B.5.6 Table Definition-Related Issues

B.5.6.1 Problems with ALTER TABLE

If you get a duplicate-key error when using `ALTER TABLE` to change the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table.

If `ALTER TABLE` dies with the following error, the problem may be that MySQL crashed during an earlier `ALTER TABLE` operation and there is an old table named `A-xxx` or `B-xxx` lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with `A-` or `B-`. (You may want to move them elsewhere instead of deleting them.)

This documentation is for an older version. If you're
This documentation is for an older version. If you're

`ALTER TABLE` works in the following way:

- Create a new table named `A-xxx` with the requested structural changes.
- Copy all rows from the original table to `A-xxx`.
- Rename the original table to `B-xxx`.
- Rename `A-xxx` to your original table name.
- Delete `B-xxx`.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as `B-xxx`. A simple rename of the table files at the system level should get your data back.

If you use `ALTER TABLE` on a transactional table or if you are using Windows or OS/2, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

B.5.6.2 TEMPORARY Table Problems

The following list indicates limitations on the use of `TEMPORARY` tables:

- A `TEMPORARY` table can only be of type `MEMORY`, `MyISAM`, `MERGE`, or `InnoDB`.

  Temporary tables are not supported for MySQL Cluster.

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

  This error also occurs if you refer to a temporary table multiple times in a stored function under different aliases, even if the references occur in different statements within the function.

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

- There are known issues in using temporary tables with replication. See Section 16.4.1, “Replication Features and Issues”, for more information.

B.5.7 Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in Section 2.20, “Operating System-Specific Notes”, and Section 21.3, “Debugging and Porting MySQL”.

The following problems are known:

- Subquery optimization for `IN` is not as effective as for `=`.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint does not work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` does not work inside `GROUP_CONCAT()` if you do not use all and only those columns that are in the `DISTINCT` list.
- If one user has a long-running transaction and another user drops a table that is updated in the transaction, there is small chance that the binary log may contain the `DROP TABLE` statement before the table is used in the transaction itself.
- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- `FLUSH TABLES WITH READ LOCK` does not block `COMMIT` if the server is running without binary logging, which may cause a problem (of consistency between tables) when doing a full backup.
- `ANALYZE TABLE` on a `BDB` table may in some cases make the table unusable until you restart `mysqld`. If this happens, look for errors of the following form in the MySQL error file:

  ```
  001207 22:07:56  bdb:  log_flush: LSN past current end-of-log
  ```

- Do not execute `ALTER TABLE` on a `BDB` table on which you are running multiple-statement transactions until all those transactions complete. (The transaction might be ignored.)
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on tables for which you are using `INSERT DELAYED`.
- Performing `LOCK TABLE ...` and `FLUSH TABLES ...` does not guarantee that there isn't a half-finished transaction in progress on the table.
- `BDB` tables are relatively slow to open. If you have many `BDB` tables in a database, it takes a long time to use the `mysql` client on the database if you are not using the `-A` option or if you are using `rehash`. This is especially noticeable when you have a large table cache.
- Replication uses query-level logging: The master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases.

  It is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

  For example:

  - `CREATE TABLE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
  - `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
  - `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If and only if the preceding queries have no `ORDER BY` clause guaranteeing a deterministic order.

For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the master and slave.

A query is optimized differently on the master and slave only if:

- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use `InnoDB` on the master, but `MyISAM` on the slave if the slave has less available disk space.)
- MySQL buffer sizes (`key_buffer_size`, and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using `mysqlbinlog|mysql`.

The easiest way to avoid this problem is to add an `ORDER BY` clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order.

- Log file names are based on the server host name if you do not specify a file name with the startup option. To retain the same log file names if you change your host name to something else, you must explicitly use options such as `--log-bin=`*`old_host_name`*`-bin`. See Section 5.1.3, “Server Command Options”. Alternatively, rename the old files to reflect your host name change. If these are binary logs, you must edit the binary log index file and fix the binary log file names there as well. (The same is true for the relay logs on a slave server.)
- `mysqlbinlog` does not delete temporary files left after a `LOAD DATA INFILE` statement. See Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”.
- `RENAME` does not work with `TEMPORARY` tables or tables used in a `MERGE` table.
- Due to the way table format (`.frm`) files are stored, you cannot use character 255 (`CHAR(255)`) in table names, column names, or enumerations.
- When using `SET CHARACTER SET`, you cannot use translated characters in database, table, and column names.
- You cannot use “_” or “%” with `ESCAPE` in `LIKE ... ESCAPE`.
- The server uses only the first `max_sort_length` bytes when comparing data values. This means that values cannot reliably be used in `GROUP BY`, `ORDER BY`, or `DISTINCT` if they differ only after the first `max_sort_length` bytes. To work around this, increase the variable value. The default value of `max_sort_length` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` and `ELT()` with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.
- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- `mysqld_safe` redirects all messages from `mysqld` to the `mysqld` log. One problem with this is that if you execute `mysqladmin refresh` to close and reopen the log, `stdout` and `stderr` are still redirected to the old log. If you use the general query log extensively, you should edit `mysqld_safe` to log to *`host_name`*`.err` instead of *`host_name`*`.log` so that you can easily reclaim the space for the old log by deleting it and executing `mysqladmin refresh`.

- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by `2`, **not** `1`:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns do not participate in the `DISTINCT` comparison.

  An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads
       WHERE userid = 9 ORDER BY id DESC;
```

  and

```
SELECT DISTINCT band_downloads.mp3id
       FROM band_downloads,band_mp3
       WHERE band_downloads.userid = 9
       AND band_mp3.id = band_downloads.mp3id
       ORDER BY band_downloads.id DESC;
```

  In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

  Note that this happens only for queries that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.

- Creation of a table of type `MERGE` does not check whether the underlying tables are compatible types.

- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Appendix C Restrictions and Limits

Table of Contents

The discussion here describes restrictions that apply to the use of MySQL features such as subqueries or views.

C.1 Restrictions on Stored Programs

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. Some of these restrictions apply to stored functions but not to stored procedures.

The restrictions for stored functions also apply to triggers.

Stored routines cannot contain arbitrary SQL statements. The following statements are not permitted:

- The table-maintenance statements `CHECK TABLE` and `OPTIMIZE TABLE`. This restriction is lifted beginning with MySQL 5.0.17.
- The locking statements `LOCK TABLES` and `UNLOCK TABLES`.
- `ALTER VIEW`. (Before MySQL 5.0.46, this restriction is enforced only for stored functions.)
- `LOAD DATA` and `LOAD TABLE`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`). Implication: You cannot use dynamic SQL within stored routines (where you construct dynamically statements as strings and then execute them). This restriction is lifted as of MySQL 5.0.13 for stored procedures; it still applies to stored functions and triggers.

  Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. For a list of statements supported as prepared statements, see Section 13.5, “SQL Syntax for Prepared Statements”.
- Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, `SELECT ... INTO local_var`

This documentation is for an older version. If you're
This documentation is for an older version. If you're

cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Syntax”.

- Inserts cannot be delayed. `INSERT DELAYED` syntax is accepted but the statement is handled as a normal `INSERT`.
- Within stored programs (stored procedures and functions, and triggers), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. Begin a transaction in this context with `START TRANSACTION` instead.

For stored functions (but not stored procedures), the following additional statements or operations are not permitted:

- Statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.
- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO *var_list*` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. A function can process a result set either with `SELECT ... INTO *var_list*` or by using a cursor and `FETCH` statements. See Section 13.2.8.1, “SELECT ... INTO Syntax”, and Section 13.6.6, “Cursors”.
- `FLUSH` statements.
- Before MySQL 5.0.10, stored functions created with `CREATE FUNCTION` must not contain references to tables, with limited exceptions. They may include some `SET` statements that contain table references, for example `SET a:= (SELECT MAX(id) FROM t)`, and `SELECT` statements that fetch values directly into variables, for example `SELECT i INTO var1 FROM t`.
- Stored functions cannot be used recursively.
- Within a stored function or trigger, it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.
- If you refer to a temporary table multiple times in a stored function under different aliases, a `Can't reopen table: '*tbl_name*'` error occurs, even if the references occur in different statements within the function.
- A stored function acquires table locks before executing, to avoid inconsistency in the binary log due to mismatch of the order in which statements execute and when they appear in the log. Statements that invoke a function are recorded rather than the statements executed within the function. Consequently, stored functions that update the same underlying tables do not execute in parallel. In contrast, stored procedures do not acquire table-level locks. All statements executed within stored procedures are written to the binary log. See Section 18.6, “Binary Logging of Stored Programs”.

Although some restrictions normally apply to stored functions and triggers but not to stored procedures, those restrictions do apply to stored procedures if they are invoked from within a stored function or trigger. For example, if you use `FLUSH` in a stored procedure, that stored procedure cannot be called from a stored function or trigger.

It is possible for the same identifier to be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column
- A routine parameter takes precedence over a table column
- A local variable in an inner block takes precedence over a local variable in an outer block

The behavior that variables take precedence over table columns is nonstandard.

Use of stored routines can cause replication problems. This issue is discussed further in Section 18.6, “Binary Logging of Stored Programs”.

`INFORMATION_SCHEMA` does not have a `PARAMETERS` table until MySQL 5.5, so applications that need to acquire routine parameter information at runtime must use workarounds such as parsing the output of `SHOW CREATE` statements or the `param_list` column of the `mysql.proc` table. `param_list` contents can be processed from within a stored routine, unlike the output from `SHOW`.

The `--replicate-wild-do-table=`*`db_name.tbl_name`* option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

There are no stored routine debugging facilities.

Before MySQL 5.0.17, `CALL` statements cannot be prepared. This true both for server-side prepared statements and for SQL prepared statements.

MySQL does not support `UNDO` handlers.

MySQL does not support `FOR` loops.

To prevent problems of interaction between server threads, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. This means that while the statement executes, it will not see changes to routines performed by other threads.

For triggers, the following additional restrictions apply:

- Triggers are not activated by foreign key actions.
- The `RETURN` statement is not permitted in triggers, which cannot return a value. To exit a trigger immediately, use the `LEAVE` statement.
- Triggers are not permitted on tables in the `mysql` database.
- The trigger cache does not detect when metadata of the underlying objects has changed. If a trigger uses a table and the table has changed since the trigger was loaded into the cache, the trigger operates using the outdated metadata.

C.2 Restrictions on Server-Side Cursors

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Server-side cursors are implemented beginning with the C API in MySQL 5.0.2 using the `mysql_stmt_attr_set()` function. A server-side cursor enables a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into an internal temporary table. Initially, this is a `MEMORY` table, but is converted to a `MyISAM` table when its size exceeds the minimum value of the `max_heap_table_size` and `tmp_table_size` system variables. The same restrictions apply to internal temporary tables created to hold the result set for a cursor as for other uses of internal temporary tables. See Section 8.4.4, “Internal Temporary Table Use in MySQL”. (Beginning with MySQL 5.0.14, the same temporary-table implementation also is used for cursors in stored routines.) One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read only; you cannot use a cursor to update rows.

`UPDATE WHERE CURRENT OF` and `DELETE WHERE CURRENT OF` are not implemented, because updatable cursors are not supported.

Cursors are nonholdable (not held open after a commit).

Cursors are asensitive.

Cursors are nonscrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as `CHECK TABLE`, `HANDLER READ`, and `SHOW BINLOG EVENTS`.

C.3 Restrictions on Subqueries

- In MySQL 5.0 before 5.0.36, if you compare a `NULL` value to a subquery using `ALL`, `ANY`, or `SOME`, and the subquery returns an empty result, the comparison might evaluate to the nonstandard result of `NULL` rather than to `TRUE` or `FALSE`.
- Subquery optimization for `IN` is not as effective as for the `=` operator or for the `IN(value_list)` operator.

  A typical case for poor `IN` subquery performance is when the subquery returns a small number of rows but the outer query returns a large number of rows to be compared to the subquery result.

  The problem is that, for a statement that uses an `IN` subquery, the optimizer rewrites it as a correlated subquery. Consider the following statement that uses an uncorrelated subquery:

  ```
  SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
  ```

  The optimizer rewrites the statement to a correlated subquery:

  ```
  SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
  ```

  If the inner and outer queries return *M* and *N* rows, respectively, the execution time becomes on the order of O(*M*×*N*), rather than O(*M*+*N*) as it would be for an uncorrelated subquery.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

An implication is that an `IN` subquery can be much slower than a query written using an `IN(value_list)` operator that lists the same values that the subquery would return.

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

Exception: The preceding prohibition does not apply if you are using a subquery for the modified table in the `FROM` clause. Example:

```
UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS _t ...);
```

Here the result from the subquery in the `FROM` clause is stored as a temporary table, so the relevant rows in `t` have already been selected by the time the update to `t` takes place.

- Row comparison operations are only partially supported:
  - For *`expr [NOT] IN subquery`*, *`expr`* can be an *n*-tuple (specified using row constructor syntax) and the subquery can return rows of *n*-tuples. The permitted syntax is therefore more specifically expressed as *`row_constructor [NOT] IN table_subquery`*
  - For *`expr op {ALL|ANY|SOME} subquery`*, *`expr`* must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

  In other words, for a subquery that returns rows of *n*-tuples, this is supported:

```
(expr_1, ..., expr_n) [NOT] IN table_subquery
```

But this is not supported:

```
(expr_1, ..., expr_n) op {ALL|ANY|SOME} subquery
```

The reason for supporting row comparisons for `IN` but not for the others is that `IN` is implemented by rewriting it as a sequence of `=` comparisons and `AND` operations. This approach cannot be used for `ALL`, `ANY`, or `SOME`.

- Prior to MySQL 5.0.26, row constructors were not well optimized; of the following two equivalent expressions, only the second could be optimized:

```
(col1, col2, ...) = (val1, val2, ...)
col1 = val1 AND col2 = val2 AND ...
```

In MySQL 5.0.26 and later, all row equalities are converted into conjunctions of equalities between row elements, and handled by the optimizer in the same way. (Bug #16081)

- Subqueries in the `FROM` clause cannot be correlated subqueries. They are materialized in whole (evaluated to produce a result set) before evaluating the outer query, so they cannot be evaluated per row of the outer query.
- MySQL does not support `LIMIT` in subqueries for certain subquery operators:

This documentation is for an older version. If you're
This documentation is for an older version. If you're

```
mysql> SELECT * FROM t1
    ->   WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
ERROR 1235 (42000): This version of MySQL doesn't yet support
 'LIMIT & IN/ALL/ANY/SOME subquery'
```

- The optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

  An exception occurs for the case where an `IN` subquery can be rewritten as a `SELECT DISTINCT` join. Example:

  ```
  SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
  ```

  That statement can be rewritten as follows:

  ```
  SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
  ```

  But in this case, the join requires an extra `DISTINCT` operation and is not more efficient than the subquery.

- MySQL permits a subquery to refer to a stored function that has data-modifying side effects such as inserting rows into a table. For example, if `f()` inserts rows, the following query can modify data:

  ```
  SELECT ... WHERE x IN (SELECT f() ...);
  ```

  This behavior is an extension to the SQL standard. In MySQL, it can produce indeterminate results because `f()` might be executed a different number of times for different executions of a given query depending on how the optimizer chooses to handle it.

  For replication, one implication of this indeterminism is that such a query can produce different results on the master and its slaves.

C.4 Restrictions on Views

View processing is not optimized:

- It is not possible to create an index on a view.
- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

Subqueries cannot be used in the `FROM` clause of a view.

There is a general principle that you cannot modify a table and select from the same table in a subquery. See Section C.3, “Restrictions on Subqueries”.

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view will be stored in a temporary table and thus

This documentation is for an older version. If you're
This documentation is for an older version. If you're

you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying `ALGORITHM = TEMPTABLE` in the view definition.)

You can use `DROP TABLE` or `ALTER TABLE` to drop or alter a table that is used in a view definition. No warning results from the `DROP` or `ALTER` operation, even though this invalidates the view. Instead, an error occurs later, when the view is used. `CHECK TABLE` can be used to check for views that have been invalidated by `DROP` or `ALTER` operations.

A view definition is “frozen” by certain statements:

- If a statement prepared by `PREPARE` refers to a view, the view definition seen each time the statement is executed later will be the definition of the view at the time it was prepared. This is true even if the view definition is changed after the statement is prepared and before it is executed. Example:

  ```
  CREATE VIEW v AS SELECT RAND();
  PREPARE s FROM 'SELECT * FROM v';
  ALTER VIEW v AS SELECT NOW();
  EXECUTE s;
  ```

  The result returned by the `EXECUTE` statement is a random number, not the current date and time.

- If a statement in a stored routine refers to a view, the view definition seen by the statement are its definition the first time that statement is executed. For example, this means that if the statement is executed in a loop, further iterations of the statement see the same view definition, even if the definition is changed later in the loop. Example:

  ```
  CREATE VIEW v AS SELECT 1;
  delimiter //
  CREATE PROCEDURE p ()
  BEGIN
    DECLARE i INT DEFAULT 0;
    WHILE i < 5 DO
      SELECT * FROM v;
      SET i = i + 1;
      ALTER VIEW v AS SELECT 2;
    END WHILE;
  END;
  //
  delimiter ;
  CALL p();
  ```

  When the procedure `p()` is called, the `SELECT` returns 1 each time through the loop, even though the view definition is changed within the loop.

  As of MySQL 5.0.46, `ALTER VIEW` is prohibited within stored routines, so this restriction does not apply.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. This includes views that have `UNION` in their definition. Not all views that are theoretically updatable can be updated. The initial view implementation was deliberately written this way to get usable, updatable views into MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist:

- Updatable views with subqueries anywhere other than in the `WHERE` clause. Some views that have subqueries in the `SELECT` list may be updatable.
- You cannot use `UPDATE` to update more than one underlying table of a view that is defined as a join.
- You cannot use `DELETE` to update a view that is defined as a join.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the `CREATE VIEW` and `SELECT` privileges), that user will be unable to call `SHOW CREATE VIEW` on that object unless the user is also granted the `SHOW VIEW` privilege.

That shortcoming can lead to problems backing up a database with `mysqldump`, which may fail due to insufficient privileges. This problem is described in Bug #22062.

The workaround to the problem is for the administrator to manually grant the `SHOW VIEW` privilege to users who are granted `CREATE VIEW`, since MySQL doesn't grant it implicitly when views are created.

Views do not have indexes, so index hints do not apply. Use of index hints when selecting from a view is not permitted.

`SHOW CREATE VIEW` displays view definitions using an `AS alias_name` clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. As of MySQL 5.0.52, aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of `SHOW CREATE VIEW` fail if any column alias exceeds 64 characters. This can cause problems in the following circumstances for views with too-long aliases:

- View definitions fail to replicate to newer slaves that enforce the column-length restriction.
- Dump files created with `mysqldump` cannot be loaded into servers that enforce the column-length restriction.

A workaround for either problem is to modify each problematic view definition to use aliases that provide shorter column names. Then the view will replicate properly, and can be dumped and reloaded without causing an error. To modify the definition, drop and create the view again with `DROP VIEW` and `CREATE VIEW`, or replace the definition with `CREATE OR REPLACE VIEW`.

For problems that occur when reloading view definitions in dump files, another workaround is to edit the dump file to modify its `CREATE VIEW` statements. However, this does not change the original view definitions, which may cause problems for subsequent dump operations.

C.5 Restrictions on XA Transactions

XA transaction support is limited to the `InnoDB` storage engine.

For “external XA,” a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. For “Internal XA”, storage engines within a MySQL server act as RMs, and the server itself acts as a TM. Internal XA support is limited by the capabilities of individual storage engines. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for `InnoDB`.

For `XA START`, the `JOIN` and `RESUME` clauses are not supported.

For `XA END`, the `SUSPEND [FOR MIGRATE]` clause is not supported.

The requirement that the *`bqual`* part of the *`xid`* value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

If an XA transaction has reached the `PREPARED` state and the MySQL server is killed (for example, with `kill -9` on Unix) or shuts down abnormally, the transaction can be continued after the server restarts. However, if the client reconnects and commits the transaction, the transaction will be absent from the binary log even though it has been committed. This means the data and the binary log have gone out of synchrony. An implication is that XA cannot be used safely together with replication.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

It is possible that the server will roll back a pending XA transaction, even one that has reached the `PREPARED` state. This happens if a client connection terminates and the server continues to run, or if clients are connected and the server shuts down gracefully. (In the latter case, the server marks each connection to be terminated, and then rolls back the `PREPARED` XA transaction associated with it.) It should be possible to commit or roll back a `PREPARED` XA transaction, but this cannot be done without changes to the binary logging mechanism.

C.6 Restrictions on Character Sets

- Identifiers are stored in `mysql` database tables (`user`, `db`, and so forth) using `utf8`, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.
- The `ucs2` character sets has the following restrictions:
  - It cannot be used as a client character set, which means that it does not work for `SET NAMES` or `SET CHARACTER SET`. (See Section 10.1.4, “Connection Character Sets and Collations”.)
  - It is currently not possible to use `LOAD DATA INFILE` to load data files that use this character set.
  - `FULLTEXT` indexes cannot be created on a column that this character set. However, you can perform `IN BOOLEAN MODE` searches on the column without an index.
- The `REGEXP` and `RLIKE` operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

C.7 Limits in MySQL

This section lists current limits in MySQL 5.0.

C.7.1 Limits on Joins

The maximum number of tables that can be referenced in a single join is 61. This also applies to the number of tables that can be referenced in the definition of a view.

C.7.2 Limits on Number of Databases and Tables

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of tables. Individual storage engines may impose engine-specific constraints. `InnoDB` permits up to 4 billion tables.

C.7.3 Limits on Table Size

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	File-size Limit
Win32 w/ FAT/FAT32	2GB/4GB

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Operating System	File-size Limit
Win32 w/ NTFS	2TB (possibly larger)
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 file system) 4TB
Solaris 9/10	16TB
MacOS X w/ HFS+	2TB
NetWare w/NSS file system	8TB

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

On Linux 2.2, you can get `MyISAM` tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 file system. Most current Linux distributions are based on kernel 2.4 or higher and include all the required LFS patches. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). With JFS and XFS, petabyte and larger files are possible on Linux.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

If you do encounter a full-table error, there are several reasons why it might have occurred:

- The disk might be full.
- The `InnoDB` storage engine maintains `InnoDB` tables within a tablespace that can be created from several files. This enables a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which permits extremely large tables. The maximum tablespace size is 64TB.

  If you are using `InnoDB` tables and run out of room in the `InnoDB` tablespace. In this case, the solution is to extend the `InnoDB` tablespace. See Section 14.2.4, “Changing the Number or Size of InnoDB Redo Log Files”.
- You are using `MyISAM` tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a `MyISAM` table and the space required for the table exceeds what is permitted by the internal pointer size. `MyISAM` creates data and index table files to permit up to 4GB by default (256TB as of MySQL 5.0.6), but this limit can be changed up to the maximum permissible size of 65,536TB ($256^7 - 1$ bytes).

  If you need a `MyISAM` table that is larger than the default limit and your operating system supports large files, the `CREATE TABLE` statement supports `AVG_ROW_LENGTH` and `MAX_ROWS` options. See Section 13.1.10, “CREATE TABLE Syntax”. The server uses these options to determine how large a table to permit.

  If the pointer size is too small for an existing table, you can change the options with `ALTER TABLE` to increase a table's maximum permissible size. See Section 13.1.4, “ALTER TABLE Syntax”.

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

  You have to specify `AVG_ROW_LENGTH` only for tables with `BLOB` or `TEXT` columns; in this case, MySQL can't optimize the space required based only on the number of rows.

  To change the default size limit for `MyISAM` tables, set the `myisam_data_pointer_size`, which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new

This documentation is for an older version. If you're
This documentation is for an older version. If you're

tables if you do not specify the `MAX_ROWS` option. The value of `myisam_data_pointer_size` can be from 2 to 7. A value of 4 permits tables up to 4GB; a value of 6 permits tables up to 256TB.

You can check the maximum data and index sizes by using this statement:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

You also can use `myisamchk -dv /path/to/table-index-file`. See Section 13.7.5, “SHOW Syntax”, or Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

Other ways to work around file-size limits for `MyISAM` tables are as follows:

- If your large table is read only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.
- MySQL includes a `MERGE` library that enables you to handle a collection of `MyISAM` tables that have identical structure as a single `MERGE` table. See Section 14.3, “The MERGE Storage Engine”.

- You are using the `NDB` storage engine, in which case you need to increase the values for the `DataMemory` and `IndexMemory` configuration parameters in your `config.ini` file. See Section 17.3.2.1, “MySQL Cluster Data Node Configuration Parameters”.
- You are using the `MEMORY` (`HEAP`) storage engine; in this case you need to increase the value of the `max_heap_table_size` system variable. See Section 5.1.4, “Server System Variables”.

C.7.4 Limits on Table Column Count and Row Size

There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact limit depends on several interacting factors.

- Every table (regardless of storage engine) has a maximum row size of 65,535 bytes. Storage engines may place additional constraints on this limit, reducing the effective maximum row size.

  The maximum row size constrains the number (and possibly size) of columns because the total length of all columns cannot exceed this size. For example, `utf8` characters require up to three bytes per character, so for a `CHAR(255) CHARACTER SET utf8` column, the server must allocate 255 × 3 = 765 bytes per value. Consequently, a table cannot contain more than 65,535 / 765 = 85 such columns.

  Storage for variable-length columns includes length bytes, which are assessed against the row size. For example, a `VARCHAR(255) CHARACTER SET utf8` column takes two bytes to store the length of the value, so each value can take up to 767 bytes.

  `BLOB` and `TEXT` columns count from one to four plus eight bytes each toward the row-size limit because their contents are stored separately from the rest of the row.

  Declaring columns `NULL` can reduce the maximum number of columns permitted. For `MyISAM` tables, `NULL` columns require additional space in the row to record whether their values are `NULL`. Each `NULL` column takes one bit extra, rounded up to the nearest byte. The maximum row length in bytes can be calculated as follows:

```
row length = 1
             + (sum of column lengths)
             + (number of NULL columns + delete_flag + 7)/8
             + (number of variable-length columns)
```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

delete_flag is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *delete_flag* is 0 for dynamic tables because the flag is stored in the dynamic row header. For information about MyISAM table formats, see Section 14.1.3, "MyISAM Table Storage Formats".

For InnoDB tables, storage size is the same for NULL and NOT NULL columns, so the preceding calculations do not apply.

The following statement to create table t1 succeeds because the columns require 32,765 + 2 bytes and 32,766 + 2 bytes, which falls within the maximum row size of 65,535 bytes:

```
mysql> CREATE TABLE t1
    -> (c1 VARCHAR(32765) NOT NULL, c2 VARCHAR(32766) NOT NULL)
    -> ENGINE = MyISAM CHARACTER SET latin1;
Query OK, 0 rows affected (0.02 sec)
```

The following statement to create table t2 fails because the columns are NULL and MyISAM requires additional space that causes the row size to exceed 65,535 bytes:

```
mysql> CREATE TABLE t2
    -> (c1 VARCHAR(32765) NULL, c2 VARCHAR(32766) NULL)
    -> ENGINE = MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

The following statement to create table t3 fails because, although the column length is within the maximum length of 65,535 bytes, two additional bytes are required to record the length, which causes the row size to exceed 65,535 bytes:

```
mysql> CREATE TABLE t3
    -> (c1 VARCHAR(65535) NOT NULL)
    -> ENGINE = MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

Reducing the column length to 65,533 or less permits the statement to succeed.

- Individual storage engines might impose additional restrictions that limit table column count. Examples:
  - InnoDB permits up to 1000 columns.
  - InnoDB restricts row size to something less than half a database page (approximately 8000 bytes), not including VARBINARY, VARCHAR, BLOB, or TEXT columns.
  - Different InnoDB storage formats (COMPRESSED, REDUNDANT) use different amounts of page header and trailer data, which affects the amount of storage available for rows.
- Each table has an .frm file that contains the table definition. The definition affects the content of this file in ways that may affect the number of columns permitted in the table. For more information, see Section C.7.5, "Limits Imposed by .frm File Structure".

C.7.5 Limits Imposed by .frm File Structure

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Each table has an `.frm` file that contains the table definition. The server uses the following expression to check some of the table information stored in the file against an upper limit of 64KB:

```
if (info_length+(ulong) create_fields.elements*FCOMP+288+
    n_length+int_length+com_length > 65535L || int_count > 255)
```

The portion of the information stored in the `.frm` file that is checked against the expression cannot grow beyond the 64KB limit, so if the table definition reaches this size, no more columns can be added.

The relevant factors in the expression are:

- `info_length` is space needed for “screens.” This is related to MySQL's Unireg heritage.
- `create_fields.elements` is the number of columns.
- `FCOMP` is 17.
- `n_length` is the total length of all column names, including one byte per name as a separator.
- `int_length` is related to the list of values for `ENUM` and `SET` columns. In this context, “int” does not mean “integer.” It means “interval,” a term that refers collectively to `ENUM` and `SET` columns.
- `int_count` is the number of unique `ENUM` and `SET` definitions.
- `com_length` is the total length of column comments.

The expression just described has several implications for permitted table definitions:

- Using long column names can reduce the maximum number of columns, as can the inclusion of `ENUM` or `SET` columns, or use of column comments.
- A table can have no more than 255 unique `ENUM` and `SET` definitions. Columns with identical element lists are considered the same against this limt. For example, if a table contains these two columns, they count as one (not two) toward this limit because the definitions are identical:

  ```
  e1 ENUM('a','b','c')
  e2 ENUM('a','b','c')
  ```

- The sum of the length of element names in the unique `ENUM` and `SET` definitions counts toward the 64KB limit, so although the theoretical limit on number of elements in a given `ENUM` column is 65,535, the practical limit is less than 3000.

C.7.6 Windows Platform Limitations

The following limitations apply to use of MySQL on the Windows platform:

- **Number of file descriptors**

  The number of open file descriptors on Windows is limited to a maximum of 2048, which may limit the ability to open a large number of tables simultaneously. This limit is due not to Windows but to C runtime library compatibility functions used to open files on Windows that use the POSIX compatibility layer.

  This limitation will also cause problems if you try to set `open_files_limit` to a value greater than the 2048 file limit.

- **Process memory**

  On Windows 32-bit platforms it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and

This documentation is for an older version. If you're
This documentation is for an older version. If you're

the default setting within Windows is to split the virtual address space between kernel (2GB) and user/applications (2GB).

Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.

- **File system aliases**

  When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` location.

  This facility is often used to move the data and index files to a RAID or other fast solution, while retaining the main `.frm` files in the default data directory configured with the `datadir` option.

- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information about this problem, see http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **Concurrent reads**

  MySQL depends on the `pread()` and `pwrite()` system calls to be able to mix `INSERT` and `SELECT`. We use mutexes to emulate `pread()` and `pwrite()`. We intend to replace the file level interface with a virtual interface in the future so that we can use the `readfile()`/`writefile()` interface to get more speed. The current implementation limits the number of open files that MySQL 5.0 can use to 2,048, which means that you cannot run as many concurrent threads on Windows as on Unix.

  This problem is fixed in MySQL 5.5.

- **Blocking read**

  MySQL uses a blocking read for each connection. That has the following implications if named-pipe connections are enabled:

  - A connection is not disconnected automatically after eight hours, as happens with the Unix version of MySQL.
  - If a connection hangs, it is not possible to break it without killing MySQL.
  - `mysqladmin kill` does not work on a sleeping connection.
  - `mysqladmin shutdown` cannot abort as long as there are sleeping connections.

  These problems are fixed in MySQL 5.1. (Bug #31621)

- **`ALTER TABLE`**

  While you are executing an `ALTER TABLE` statement, the table is locked from being used by other threads. This has to do with the fact that on Windows, you can't delete a file that is in use by another thread. In the future, we may find some way to work around this problem.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

- **`DROP TABLE`**

  `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` handler does the table mapping hidden from the upper layer of MySQL. Because Windows does not permit dropping files that are open, you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows, because MySQL does not support Windows symbolic links. These options also are ignored on systems that have a nonfunctional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by another session.

- **Case-insensitive names**

  File names are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 9.2.2, “Identifier Case Sensitivity”.

- **Directory and file names**

  On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name will not work in the Western locale (code page 1252):

  ```
  datadir="C:/私たちのプロジェクトのデータ"
  ```

  The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA INFILE`.

- **The “`\`” path name separator character**

  Path name components in Windows are separated by the “`\`” character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style file names with “`/`” characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the “`\`” character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

  This is mainly a problem when you try to apply a binary log as follows:

  ```
  shell> mysqlbinlog binary_log_file | mysql --user=root
  ```

This documentation is for an older version. If you're
This documentation is for an older version. If you're

If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

```
shell> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
shell> mysql --user=root --execute "source /tmp/bin.sql"
```

The latter command also can be used to reliably read in any SQL file that may contain binary data.

This documentation is for an older version. If you're
This documentation is for an older version. If you're

General Index

Symbols

A

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

B

This documentation is for an older version. If you're
This documentation is for an older version. If you're

C

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

D

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

E

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ER_CANT_CREATE_FEDERATED_TABLE, 2101
ER_CANT_CREATE_FILE, 2069
ER_CANT_CREATE_GEOMETRY_OBJECT, 2099
ER_CANT_CREATE_TABLE, 2069
ER_CANT_CREATE_THREAD, 2079
ER_CANT_CREATE_USER_WITH_GRANT, 2099
ER_CANT_DELETE_FILE, 2070
ER_CANT_DO_THIS_DURING_AN_TRANSACTION, 2082
ER_CANT_DROP_FIELD_OR_KEY, 2076
ER_CANT_FIND_DL_ENTRY, 2078
ER_CANT_FIND_SYSTEM_REC, 2070
ER_CANT_FIND_UDF, 2078
ER_CANT_GET_STAT, 2070
ER_CANT_GET_WD, 2070
ER_CANT_INITIALIZE_UDF, 2078
ER_CANT_LOCK, 2070
ER_CANT_OPEN_FILE, 2070
ER_CANT_OPEN_LIBRARY, 2078
ER_CANT_READ_DIR, 2071
ER_CANT_REMOVE_ALL_FIELDS, 2076
ER_CANT_REOPEN_TABLE, 2079
ER_CANT_SET_WD, 2071
ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG, 2102
ER_CANT_UPDATE_WITH_READLOCK, 2086
ER_CANT_USE_OPTION_HERE, 2087
ER_CHECKREAD, 2071
ER_CHECK_NOT_IMPLEMENTED, 2082
ER_CHECK_NO_SUCH_TABLE, 2082
ER_COLLATION_CHARSET_MISMATCH, 2088
ER_COLUMNACCESS_DENIED_ERROR, 2080
ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG, 2100
ER_CONFLICTING_DECLARATIONS, 2091
ER_CONNECT_TO_FOREIGN_DATA_SOURCE, 2101
ER_CONNECT_TO_MASTER, 2085
ER_CON_COUNT_ERROR, 2072
ER_CORRUPT_HELP_DB, 2087
ER_CRASHED_ON_REPAIR, 2083
ER_CRASHED_ON_USAGE, 2083
ER_CREATE_DB_WITH_READ_LOCK, 2085
ER_CUT_VALUE_GROUP_CONCAT, 2088
ER_CYCLIC_REFERENCE, 2087
ER_DATA_TOO_LONG, 2098
ER_DATETIME_FUNCTION_OVERFLOW, 2102
ER_DBACCESS_DENIED_ERROR, 2072
ER_DB_CREATE_EXISTS, 2070
ER_DB_DROP_DELETE, 2070
ER_DB_DROP_EXISTS, 2070
ER_DB_DROP_RMDIR, 2070
ER_DELAYED_CANT_CHANGE_LOCK, 2080
ER_DELAYED_INSERT_TABLE_LOCKED, 2081
ER_DERIVED_MUST_HAVE_ALIAS, 2087
ER_DIFF_GROUPS_PROC, 2097
ER_DISK_FULL, 2071
ER_DIVISION_BY_ZERO, 2096
ER_DROP_DB_WITH_READ_LOCK, 2084
ER_DROP_USER, 2089
ER_DUMP_NOT_IMPLEMENTED, 2083
ER_DUPLICATED_VALUE_IN_TYPE, 2090
ER_DUP_ARGUMENT, 2086
ER_DUP_ENTRY, 2074
ER_DUP_FIELDNAME, 2074
ER_DUP_KEY, 2071
ER_DUP_KEYNAME, 2074
ER_DUP_UNIQUE, 2081
ER_EMPTY_QUERY, 2074
ER_ERROR_DURING_CHECKPOINT, 2083
ER_ERROR_DURING_COMMIT, 2082
ER_ERROR_DURING_FLUSH_LOGS, 2082
ER_ERROR_DURING_ROLLBACK, 2082
ER_ERROR_ON_CLOSE, 2071
ER_ERROR_ON_READ, 2071
ER_ERROR_ON_RENAME, 2071
ER_ERROR_ON_WRITE, 2071
ER_ERROR_WHEN_EXECUTING_COMMAND, 2086
ER_EXEC_STMT_WITH_OPEN_CURSOR, 2100
ER_FAILED_ROUTINE_BREAK_BINLOG, 2099
ER_FEATURE_DISABLED, 2090
ER_FIELD_SPECIFIED_TWICE, 2077
ER_FILE_EXISTS_ERROR, 2076
ER_FILE_NOT_FOUND, 2071
ER_FILE_USED, 2071
ER_FILSORT_ABORT, 2071
ER_FLUSH_MASTER_BINLOG_CLOSED, 2083
ER_FORBID_SCHEMA_CHANGE, 2103
ER_FORCING_CLOSE, 2075
ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST, 2101
ER_FOREIGN_DATA_STRING_INVALID, 2101
ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE, 2101
ER_FORM_NOT_FOUND, 2071
ER_FPARSER_BAD_HEADER, 2094
ER_FPARSER_EOF_IN_COMMENT, 2094
ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER, 2094
ER_FPARSER_ERROR_IN_PARAMETER, 2094
ER_FPARSER_TOO_BIG_FILE, 2094
ER_FRM_UNKNOWN_TYPE, 2094
ER_FSEEK_FAIL, 2096
ER_FT_MATCHING_KEY_NOT_FOUND, 2083
ER_FUNCTION_NOT_DEFINED, 2079
ER_GET_ERRMSG, 2091
ER_GET_ERRNO, 2071
ER_GET_TEMPORARY_ERRMSG, 2091

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ER_GLOBAL_VARIABLE, 2086
ER_GOT_SIGNAL, 2075
ER_GRANT_WRONG_HOST_OR_USER, 2080
ER_HANDSHAKE_ERROR, 2072
ER_HASHCHK, 2069
ER_HOSTNAME, 2105
ER_HOST_IS_BLOCKED, 2079
ER_HOST_NOT_PRIVILEGED, 2079
ER_ILLEGAL_GRANT_FOR_TABLE, 2080
ER_ILLEGAL_HA, 2072
ER_ILLEGAL_REFERENCE, 2087
ER_ILLEGAL_VALUE_FOR_TYPE, 2096
ER_INCORRECT_GLOBAL_LOCAL_VAR, 2087
ER_INDEX_REBUILD, 2083
ER_INSERT_INFO, 2076
ER_INVALID_CHARACTER_STRING, 2091
ER_INVALID_DEFAULT, 2074
ER_INVALID_GROUP_FUNC_USE, 2077
ER_INVALID_ON_UPDATE, 2091
ER_INVALID_USE_OF_NULL, 2079
ER_IO_ERR_LOG_INDEX_READ, 2096
ER_IPSOCK_ERROR, 2075
ER_KEY_COLUMN_DOES_NOT_EXITS, 2075
ER_KEY_DOES_NOT_EXITS, 2082
ER_KEY_NOT_FOUND, 2072
ER_KEY_PART_0, 2097
ER_KEY_REF_DO_NOT_MATCH_TABLE_REF, 2087
ER_KILL_DENIED_ERROR, 2076
ER_LOAD_DATA_INVALID_COLUMN, 2106
ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR, 2099
ER_LOAD_INFO, 2076
ER_LOCAL_VARIABLE, 2086
ER_LOCK_DEADLOCK, 2085
ER_LOCK_OR_ACTIVE_TRANSACTION, 2083
ER_LOCK_TABLE_FULL, 2084
ER_LOCK_WAIT_TIMEOUT, 2084
ER_LOGGING_PROHIBIT_CHANGING_OF, 2097
ER_LOG_IN_USE, 2097
ER_LOG_PURGE_NO_FILE, 2106
ER_LOG_PURGE_UNKNOWN_ERR, 2097
ER_MALFORMED_DEFINER, 2102
ER_MASTER, 2083
ER_MASTER_FATAL_ERROR_READING_BINLOG, 2087
ER_MASTER_INFO, 2084
ER_MASTER_NET_READ, 2083
ER_MASTER_NET_WRITE, 2083
ER_MAX_PREPARED_STMT_COUNT_REACHED, 2104
ER_MISSING_SKIP_SLAVE, 2090
ER_MIXING_NOT_ALLOWED, 2086
ER_MIX_OF_GROUP_FUNC_AND_FIELDS, 2079
ER_MULTIPLE_PRI_KEY, 2074
ER_M_BIGGER_THAN_D, 2100
ER_NAME_BECOMES_EMPTY, 2105
ER_NET_ERROR_ON_WRITE, 2081
ER_NET_FCNTL_ERROR, 2080
ER_NET_PACKETS_OUT_OF_ORDER, 2080
ER_NET_PACKET_TOO_LARGE, 2080
ER_NET_READ_ERROR, 2081
ER_NET_READ_ERROR_FROM_PIPE, 2080
ER_NET_READ_INTERRUPTED, 2081
ER_NET_UNCOMPRESS_ERROR, 2081
ER_NET_WRITE_INTERRUPTED, 2081
ER_NEW_ABORTING_CONNECTION, 2083
ER_NISAMCHK, 2069
ER_NO, 2069
ER_NONEXISTING_GRANT, 2079
ER_NONEXISTING_PROC_GRANT, 2098
ER_NONEXISTING_TABLE_GRANT, 2080
ER_NONUNIQ_TABLE, 2074
ER_NONUPDATEABLE_COLUMN, 2094
ER_NON_GROUPING_FIELD_USED, 2104
ER_NON_INSERTABLE_TABLE, 2105
ER_NON_UNIQ_ERROR, 2073
ER_NON_UPDATABLE_TABLE, 2090
ER_NORMAL_SHUTDOWN, 2075
ER_NOT_ALLOWED_COMMAND, 2080
ER_NOT_FORM_FILE, 2072
ER_NOT_KEYFILE, 2072
ER_NOT_SUPPORTED_AUTH_MODE, 2088
ER_NOT_SUPPORTED_YET, 2087
ER_NO_BINARY_LOGGING, 2097
ER_NO_DB_ERROR, 2073
ER_NO_DEFAULT, 2086
ER_NO_DEFAULT_FOR_FIELD, 2096
ER_NO_DEFAULT_FOR_VIEW_FIELD, 2100
ER_NO_FILE_MAPPING, 2097
ER_NO_GROUP_FOR_PROC, 2097
ER_NO_PERMISSION_TO_CREATE_USER, 2085
ER_NO_RAID_COMPILED, 2082
ER_NO_REFERENCED_ROW, 2085
ER_NO_REFERENCED_ROW_2, 2103
ER_NO_SUCH_INDEX, 2075
ER_NO_SUCH_TABLE, 2080
ER_NO_SUCH_THREAD, 2076
ER_NO_SUCH_USER, 2103
ER_NO_TABLES_USED, 2076
ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA, 2104
ER_NO_UNIQUE_LOGFILE, 2076
ER_NO_VIEW_USER, 2102
ER_NULL_COLUMN_IN_INDEX, 2078
ER_OLD_FILE_FORMAT, 2103
ER_OLD_KEYFILE, 2072
ER_OPEN_AS_READONLY, 2072
ER_OPERAND_COLUMNS, 2087

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ER_OPTION_PREVENTS_STATEMENT, 2090
ER_ORDER_WITH_PROC, 2097
ER_OUTOFMEMORY, 2072
ER_OUT_OF_RESOURCES, 2072
ER_OUT_OF_SORTMEMORY, 2072
ER_PARSE_ERROR, 2074
ER_PASSWD_LENGTH, 2096
ER_PASSWORD_ANONYMOUS_USER, 2079
ER_PASSWORD_NOT_ALLOWED, 2079
ER_PASSWORD_NO_MATCH, 2079
ER_PRIMARY_CANT_HAVE_NULL, 2082
ER_PROCACCESS_DENIED_ERROR, 2096
ER_PROC_AUTO_GRANT_FAIL, 2098
ER_PROC_AUTO_REVOKE_FAIL, 2098
ER_PS_MANY_PARAM, 2097
ER_PS_NO_RECURSION, 2102
ER_QUERY_INTERRUPTED, 2092
ER_QUERY_ON_FOREIGN_DATA_SOURCE, 2101
ER_QUERY_ON_MASTER, 2086
ER_READY, 2075
ER_READ_ONLY_TRANSACTION, 2084
ER_RECORD_FILE_FULL, 2078
ER_REGEXP_ERROR, 2079
ER_RELAY_LOG_FAIL, 2096
ER_RELAY_LOG_INIT, 2097
ER_REMOVED_SPACES, 2105
ER_REQUIRES_PRIMARY_KEY, 2082
ER_RESERVED_SYNTAX, 2097
ER_REVOKE_GRANTS, 2089
ER_ROW_IS_REFERENCED, 2085
ER_ROW_IS_REFERENCED_2, 2103
ER_SCALE_BIGGER_THAN_PRECISION, 2100
ER_SELECT_REDUCED, 2088
ER_SERVER_IS_IN_SECURE_AUTH_MODE, 2089
ER_SERVER_SHUTDOWN, 2073
ER_SET_CONSTANTS_ONLY, 2084
ER_SHUTDOWN_COMPLETE, 2075
ER_SLAVE_IGNORED_SSL_PARAMS, 2089
ER_SLAVE_IGNORED_TABLE, 2087
ER_SLAVE_MUST_STOP, 2084
ER_SLAVE_NOT_RUNNING, 2084
ER_SLAVE_THREAD, 2084
ER_SLAVE_WAS_NOT_RUNNING, 2088
ER_SLAVE_WAS_RUNNING, 2088
ER_SPATIAL_CANT_HAVE_NULL, 2088
ER_SPECIFIC_ACCESS_DENIED_ERROR, 2086
ER_SP_ALREADY_EXISTS, 2091
ER_SP_BADRETURN, 2092
ER_SP_BADSELECT, 2092
ER_SP_BADSTATEMENT, 2092
ER_SP_BAD_CURSOR_QUERY, 2093
ER_SP_BAD_CURSOR_SELECT, 2093
ER_SP_BAD_SQLSTATE, 2098
ER_SP_BAD_VAR_SHADOW, 2103
ER_SP_CANT_ALTER, 2093
ER_SP_CANT_SET_AUTOCOMMIT, 2102
ER_SP_CASE_NOT_FOUND, 2094
ER_SP_COND_MISMATCH, 2092
ER_SP_CURSOR_AFTER_HANDLER, 2094
ER_SP_CURSOR_ALREADY_OPEN, 2093
ER_SP_CURSOR_MISMATCH, 2093
ER_SP_CURSOR_NOT_OPEN, 2093
ER_SP_DOES_NOT_EXIST, 2091
ER_SP_DROP_FAILED, 2091
ER_SP_DUP_COND, 2093
ER_SP_DUP_CURS, 2093
ER_SP_DUP_HANDLER, 2099
ER_SP_DUP_PARAM, 2093
ER_SP_DUP_VAR, 2093
ER_SP_FETCH_NO_DATA, 2093
ER_SP_GOTO_IN_HNDLR, 2095
ER_SP_LABEL_MISMATCH, 2092
ER_SP_LABEL_REDEFINE, 2092
ER_SP_LILABEL_MISMATCH, 2092
ER_SP_NORETURN, 2092
ER_SP_NORETURNEND, 2092
ER_SP_NOT_VAR_ARG, 2099
ER_SP_NO_AGGREGATE, 2104
ER_SP_NO_DROP_SP, 2095
ER_SP_NO_RECURSION, 2100
ER_SP_NO_RECURSIVE_CREATE, 2091
ER_SP_NO_RETSET, 2099
ER_SP_NO_RETSET_IN_FUNC, 2099
ER_SP_NO_USE, 2093
ER_SP_PROC_TABLE_CORRUPT, 2104
ER_SP_RECURSION_LIMIT, 2104
ER_SP_STORE_FAILED, 2092
ER_SP_SUBSELECT_NYI, 2093
ER_SP_UNDECLARED_VAR, 2093
ER_SP_UNINIT_VAR, 2092
ER_SP_VARCOND_AFTER_CURSHNDLR, 2094
ER_SP_WRONG_NAME, 2104
ER_SP_WRONG_NO_OF_ARGS, 2092
ER_SP_WRONG_NO_OF_FETCH_ARGS, 2093
ER_STACK_OVERRUN, 2078
ER_STACK_OVERRUN_NEED_MORE, 2101
ER_STARTUP, 2099
ER_STMT_HAS_NO_OPEN_CURSOR, 2100
ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG, 2094
ER_SUBQUERY_NO_1_ROW, 2087
ER_SYNTAX_ERROR, 2080
ER_TABLEACCESS_DENIED_ERROR, 2080
ER_TABLENAME_NOT_ALLOWED_HERE, 2088
ER_TABLE_CANT_HANDLE_AUTO_INCREMENT, 2081
ER_TABLE_CANT_HANDLE_BLOB, 2081
ER_TABLE_CANT_HANDLE_FT, 2085
ER_TABLE_CANT_HANDLE_SPKEYS, 2104

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ER_CANT_CREATE_USER_WITH_GRANT error code, 2099
ER_CANT_DELETE_FILE error code, 2070
ER_CANT_DO_THIS_DURING_AN_TRANSACTION error code, 2082
ER_CANT_DROP_FIELD_OR_KEY error code, 2076
ER_CANT_FIND_DL_ENTRY error code, 2078
ER_CANT_FIND_SYSTEM_REC error code, 2070
ER_CANT_FIND_UDF error code, 2078
ER_CANT_GET_STAT error code, 2070
ER_CANT_GET_WD error code, 2070
ER_CANT_INITIALIZE_UDF error code, 2078
ER_CANT_LOCK error code, 2070
ER_CANT_OPEN_FILE error code, 2070
ER_CANT_OPEN_LIBRARY error code, 2078
ER_CANT_READ_DIR error code, 2071
ER_CANT_REMOVE_ALL_FIELDS error code, 2076
ER_CANT_REOPEN_TABLE error code, 2079
ER_CANT_SET_WD error code, 2071
ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG error code, 2102
ER_CANT_UPDATE_WITH_READLOCK error code, 2086
ER_CANT_USE_OPTION_HERE error code, 2087
ER_CHECKREAD error code, 2071
ER_CHECK_NOT_IMPLEMENTED error code, 2082
ER_CHECK_NO_SUCH_TABLE error code, 2082
ER_COLLATION_CHARSET_MISMATCH error code, 2088
ER_COLUMNACCESS_DENIED_ERROR error code, 2080
ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG error code, 2100
ER_CONFLICTING_DECLARATIONS error code, 2091
ER_CONNECT_TO_FOREIGN_DATA_SOURCE error code, 2101
ER_CONNECT_TO_MASTER error code, 2085
ER_CON_COUNT_ERROR error code, 2072
ER_CORRUPT_HELP_DB error code, 2087
ER_CRASHED_ON_REPAIR error code, 2083
ER_CRASHED_ON_USAGE error code, 2083
ER_CREATE_DB_WITH_READ_LOCK error code, 2085
ER_CUT_VALUE_GROUP_CONCAT error code, 2088
ER_CYCLIC_REFERENCE error code, 2087
ER_DATA_TOO_LONG error code, 2098
ER_DATETIME_FUNCTION_OVERFLOW error code, 2102
ER_DBACCESS_DENIED_ERROR error code, 2072
ER_DB_CREATE_EXISTS error code, 2070
ER_DB_DROP_DELETE error code, 2070
ER_DB_DROP_EXISTS error code, 2070
ER_DB_DROP_RMDIR error code, 2070
ER_DELAYED_CANT_CHANGE_LOCK error code, 2080
ER_DELAYED_INSERT_TABLE_LOCKED error code, 2081
ER_DERIVED_MUST_HAVE_ALIAS error code, 2087
ER_DIFF_GROUPS_PROC error code, 2097
ER_DISK_FULL error code, 2071
ER_DIVISION_BY_ZERO error code, 2096
ER_DROP_DB_WITH_READ_LOCK error code, 2084
ER_DROP_USER error code, 2089
ER_DUMP_NOT_IMPLEMENTED error code, 2083
ER_DUPLICATED_VALUE_IN_TYPE error code, 2090
ER_DUP_ARGUMENT error code, 2086
ER_DUP_ENTRY error code, 2074
ER_DUP_FIELDNAME error code, 2074
ER_DUP_KEY error code, 2071
ER_DUP_KEYNAME error code, 2074
ER_DUP_UNIQUE error code, 2081
ER_EMPTY_QUERY error code, 2074
ER_ERROR_DURING_CHECKPOINT error code, 2083
ER_ERROR_DURING_COMMIT error code, 2082
ER_ERROR_DURING_FLUSH_LOGS error code, 2082
ER_ERROR_DURING_ROLLBACK error code, 2082
ER_ERROR_ON_CLOSE error code, 2071
ER_ERROR_ON_READ error code, 2071
ER_ERROR_ON_RENAME error code, 2071
ER_ERROR_ON_WRITE error code, 2071
ER_ERROR_WHEN_EXECUTING_COMMAND error code, 2086
ER_EXEC_STMT_WITH_OPEN_CURSOR error code, 2100
ER_FAILED_ROUTINE_BREAK_BINLOG error code, 2099
ER_FEATURE_DISABLED error code, 2090
ER_FIELD_SPECIFIED_TWICE error code, 2077
ER_FILE_EXISTS_ERROR error code, 2076
ER_FILE_NOT_FOUND error code, 2071
ER_FILE_USED error code, 2071
ER_FILSORT_ABORT error code, 2071
ER_FLUSH_MASTER_BINLOG_CLOSED error code, 2083
ER_FORBID_SCHEMA_CHANGE error code, 2103
ER_FORCING_CLOSE error code, 2075
ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST error code, 2101
ER_FOREIGN_DATA_STRING_INVALID error code, 2101
ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE error code, 2101
ER_FORM_NOT_FOUND error code, 2071
ER_FPARSER_BAD_HEADER error code, 2094
ER_FPARSER_EOF_IN_COMMENT error code, 2094
ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER error code, 2094
ER_FPARSER_ERROR_IN_PARAMETER error code, 2094

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ER_FPARSER_TOO_BIG_FILE error code, 2094
ER_FRM_UNKNOWN_TYPE error code, 2094
ER_FSEEK_FAIL error code, 2096
ER_FT_MATCHING_KEY_NOT_FOUND error code, 2083
ER_FUNCTION_NOT_DEFINED error code, 2079
ER_GET_ERRMSG error code, 2091
ER_GET_ERRNO error code, 2071
ER_GET_TEMPORARY_ERRMSG error code, 2091
ER_GLOBAL_VARIABLE error code, 2086
ER_GOT_SIGNAL error code, 2075
ER_GRANT_WRONG_HOST_OR_USER error code, 2080
ER_HANDSHAKE_ERROR error code, 2072
ER_HASHCHK error code, 2069
ER_HOSTNAME error code, 2105
ER_HOST_IS_BLOCKED error code, 2079
ER_HOST_NOT_PRIVILEGED error code, 2079
ER_ILLEGAL_GRANT_FOR_TABLE error code, 2080
ER_ILLEGAL_HA error code, 2072
ER_ILLEGAL_REFERENCE error code, 2087
ER_ILLEGAL_VALUE_FOR_TYPE error code, 2096
ER_INCORRECT_GLOBAL_LOCAL_VAR error code, 2087
ER_INDEX_REBUILD error code, 2083
ER_INSERT_INFO error code, 2076
ER_INVALID_CHARACTER_STRING error code, 2091
ER_INVALID_DEFAULT error code, 2074
ER_INVALID_GROUP_FUNC_USE error code, 2077
ER_INVALID_ON_UPDATE error code, 2091
ER_INVALID_USE_OF_NULL error code, 2079
ER_IO_ERR_LOG_INDEX_READ error code, 2096
ER_IPSOCK_ERROR error code, 2075
ER_KEY_COLUMN_DOES_NOT_EXITS error code, 2075
ER_KEY_DOES_NOT_EXITS error code, 2082
ER_KEY_NOT_FOUND error code, 2072
ER_KEY_PART_0 error code, 2097
ER_KEY_REF_DO_NOT_MATCH_TABLE_REF error code, 2087
ER_KILL_DENIED_ERROR error code, 2076
ER_LOAD_DATA_INVALID_COLUMN error code, 2106
ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR error code, 2099
ER_LOAD_INFO error code, 2076
ER_LOCAL_VARIABLE error code, 2086
ER_LOCK_DEADLOCK error code, 2085
ER_LOCK_OR_ACTIVE_TRANSACTION error code, 2083
ER_LOCK_TABLE_FULL error code, 2084
ER_LOCK_WAIT_TIMEOUT error code, 2084
ER_LOGGING_PROHIBIT_CHANGING_OF error code, 2097
ER_LOG_IN_USE error code, 2097
ER_LOG_PURGE_NO_FILE error code, 2106
ER_LOG_PURGE_UNKNOWN_ERR error code, 2097
ER_MALFORMED_DEFINER error code, 2102
ER_MASTER error code, 2083
ER_MASTER_FATAL_ERROR_READING_BINLOG error code, 2087
ER_MASTER_INFO error code, 2084
ER_MASTER_NET_READ error code, 2083
ER_MASTER_NET_WRITE error code, 2083
ER_MAX_PREPARED_STMT_COUNT_REACHED error code, 2104
ER_MISSING_SKIP_SLAVE error code, 2090
ER_MIXING_NOT_ALLOWED error code, 2086
ER_MIX_OF_GROUP_FUNC_AND_FIELDS error code, 2079
ER_MULTIPLE_PRI_KEY error code, 2074
ER_M_BIGGER_THAN_D error code, 2100
ER_NAME_BECOMES_EMPTY error code, 2105
ER_NET_ERROR_ON_WRITE error code, 2081
ER_NET_FCNTL_ERROR error code, 2080
ER_NET_PACKETS_OUT_OF_ORDER error code, 2080
ER_NET_PACKET_TOO_LARGE error code, 2080
ER_NET_READ_ERROR error code, 2081
ER_NET_READ_ERROR_FROM_PIPE error code, 2080
ER_NET_READ_INTERRUPTED error code, 2081
ER_NET_UNCOMPRESS_ERROR error code, 2081
ER_NET_WRITE_INTERRUPTED error code, 2081
ER_NEW_ABORTING_CONNECTION error code, 2083
ER_NISAMCHK error code, 2069
ER_NO error code, 2069
ER_NONEXISTING_GRANT error code, 2079
ER_NONEXISTING_PROC_GRANT error code, 2098
ER_NONEXISTING_TABLE_GRANT error code, 2080
ER_NONUNIQ_TABLE error code, 2074
ER_NONUPDATEABLE_COLUMN error code, 2094
ER_NON_GROUPING_FIELD_USED error code, 2104
ER_NON_INSERTABLE_TABLE error code, 2105
ER_NON_UNIQ_ERROR error code, 2073
ER_NON_UPDATABLE_TABLE error code, 2090
ER_NORMAL_SHUTDOWN error code, 2075
ER_NOT_ALLOWED_COMMAND error code, 2080
ER_NOT_FORM_FILE error code, 2072
ER_NOT_KEYFILE error code, 2072
ER_NOT_SUPPORTED_AUTH_MODE error code, 2088
ER_NOT_SUPPORTED_YET error code, 2087
ER_NO_BINARY_LOGGING error code, 2097
ER_NO_DB_ERROR error code, 2073
ER_NO_DEFAULT error code, 2086
ER_NO_DEFAULT_FOR_FIELD error code, 2096
ER_NO_DEFAULT_FOR_VIEW_FIELD error code, 2100
ER_NO_FILE_MAPPING error code, 2097
ER_NO_GROUP_FOR_PROC error code, 2097

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

F

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

G

H

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

J

K

This documentation is for an older version. If you're
This documentation is for an older version. If you're

L

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

M

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

N

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

O

This documentation is for an older version. If you're
This documentation is for an older version. If you're

P

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Q

R

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

S

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

T

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

U

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

V

This documentation is for an older version. If you're
This documentation is for an older version. If you're

W

This documentation is for an older version. If you're
This documentation is for an older version. If you're

X

Y

Z

This documentation is for an older version. If you're
This documentation is for an older version. If you're

C Function Index

This documentation is for an older version. If you're

This documentation is for an older version. If you're

Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_escape_string()

Section 20.6.6, “C API Function Overview”
Section 6.1.7, “Client Programming Security Guidelines”
Section 20.6.7.16, “mysql_escape_string()”

mysql_fetch_field()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 20.6.7.17, “mysql_fetch_field()”
Section 20.6.7.23, “mysql_field_seek()”
Section 20.6.7.24, “mysql_field_tell()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”

mysql_fetch_field_direct()

Section 20.6.6, “C API Function Overview”
Section 20.6.11.22, “mysql_stmt_result_metadata()”

mysql_fetch_fields()

Section 20.6.6, “C API Function Overview”
Section 20.6.11.22, “mysql_stmt_result_metadata()”

mysql_fetch_lengths()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.20, “mysql_fetch_lengths()”
Section 20.6.7.21, “mysql_fetch_row()”

mysql_fetch_row()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 14.7.1, “Description of the FEDERATED Storage Engine”
Section 20.6.7.13, “mysql_eof()”
Section 20.6.7.14, “mysql_errno()”
Section 20.6.7.20, “mysql_fetch_lengths()”
Section 20.6.7.21, “mysql_fetch_row()”
Section 20.6.7.59, “mysql_row_tell()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”
Section 20.6.14.2, “What Results You Can Get from a Query”

mysql_field_count()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.22, “mysql_field_count()”
Section 20.6.7.47, “mysql_num_fields()”
Section 20.6.7.51, “mysql_query()”
Section 20.6.7.54, “mysql_real_query()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”
Section 20.6.7.69, “mysql_store_result()”

Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_field_seek()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 20.6.7.17, “mysql_fetch_field()”
Section 20.6.7.24, “mysql_field_tell()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”

mysql_field_tell()

Section 20.6.6, “C API Function Overview”
Section 20.6.11.22, “mysql_stmt_result_metadata()”

mysql_free_result()

Section 20.6.6, “C API Function Overview”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section B.5.2.14, “Commands out of sync”
Section 20.6.7.25, “mysql_free_result()”
Section 20.6.7.41, “mysql_list_dbs()”
Section 20.6.7.42, “mysql_list_fields()”
Section 20.6.7.43, “mysql_list_processes()”
Section 20.6.7.44, “mysql_list_tables()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”

mysql_get_character_set_info()

Section 20.6.6, “C API Function Overview”
Section 10.4.2, “Choosing a Collation ID”

mysql_get_client_info()

Section 20.6.6, “C API Function Overview”
Section 20.6.4.4, “C API Server and Client Library Versions”
Section 20.6.7.7, “mysql_connect()”

mysql_get_client_version()

Section 20.6.6, “C API Function Overview”
Section 20.6.4.4, “C API Server and Client Library Versions”

mysql_get_host_info()

Section 20.6.6, “C API Function Overview”

mysql_get_proto_info()

Section 20.6.6, “C API Function Overview”

mysql_get_server_info()

Section 20.6.6, “C API Function Overview”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 20.6.4.4, “C API Server and Client Library Versions”

mysql_get_server_version()

Section 20.6.6, “C API Function Overview”
Section 20.6.4.4, “C API Server and Client Library Versions”

mysql_get_ssl_cipher()

Section 20.6.6, “C API Function Overview”
Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”
Section 20.6.7.33, “mysql_get_ssl_cipher()”

mysql_hex_string()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.34, “mysql_hex_string()”

mysql_info()

Section 13.1.4, “ALTER TABLE Syntax”
Section 20.6.6, “C API Function Overview”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.2.5, “INSERT Syntax”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 20.6.7.35, “mysql_info()”
Section 20.6.7.49, “mysql_options()”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 13.2.10, “UPDATE Syntax”
Section 20.6.14.2, “What Results You Can Get from a Query”

mysql_init()

Section 20.6.6, “C API Function Overview”
Section 20.6.12.1, “my_init()”
Section 20.6.7.5, “mysql_close()”
Section 20.6.7.33, “mysql_get_ssl_cipher()”
Section 20.6.7.36, “mysql_init()”
Section 20.6.7.40, “mysql_library_init()”
Section 20.6.7.49, “mysql_options()”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.7.67, “mysql_ssl_set()”
Section 20.6.12.3, “mysql_thread_init()”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_insert_id()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 13.1.10, “CREATE TABLE Syntax”
Section 20.6.14.3, “How to Get the Unique ID for the Last Inserted Row”
Section 12.13, “Information Functions”
Section 13.2.5, “INSERT Syntax”
Section 20.6.7.37, “mysql_insert_id()”
Section 5.1.4, “Server System Variables”
Section 1.8.2.3, “Transactions and Atomic Operations”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 20.6.14.2, “What Results You Can Get from a Query”

mysql_kill()

Section 20.6.6, “C API Function Overview”
Section 20.6.15, “Controlling Automatic Reconnection Behavior”
Section 20.6.7.70, “mysql_thread_id()”

mysql_library_end()

Section 20.6.13, “C API Embedded Server Function Descriptions”
Section 20.6.6, “C API Function Overview”
Section 20.5, “libmysqld, the Embedded MySQL Server Library”
Section 20.6.7.39, “mysql_library_end()”
Section 20.6.7.40, “mysql_library_init()”
Section 20.6.13.2, “mysql_server_end()”

mysql_library_init()

Section 20.6.13, “C API Embedded Server Function Descriptions”
Section 20.6.6, “C API Function Overview”
Section 20.5, “libmysqld, the Embedded MySQL Server Library”
Section 20.6.12.1, “my_init()”
Section 20.6.7.36, “mysql_init()”
Section 20.6.7.40, “mysql_library_init()”
Section 20.6.13.1, “mysql_server_init()”
Section 20.6.12.3, “mysql_thread_init()”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_list_dbs()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.25, “mysql_free_result()”
Section 20.6.7.41, “mysql_list_dbs()”

mysql_list_fields()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 20.6.7.42, “mysql_list_fields()”

mysql_list_processes()

Section 20.6.6, “C API Function Overview”

mysql_list_tables()

Section 20.6.6, “C API Function Overview”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 20.6.7.44, “mysql_list_tables()”

mysql_more_results()

Section 20.6.6, “C API Function Overview”
Section 20.6.16, “C API Support for Multiple Statement Execution”
Section 20.6.7.45, “mysql_more_results()”
Section 20.6.7.46, “mysql_next_result()”

mysql_next_result()

Section 20.6.6, “C API Function Overview”
Section 20.6.16, “C API Support for Multiple Statement Execution”
Section 13.2.1, “CALL Syntax”
Section 20.6.7.45, “mysql_more_results()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.7.64, “mysql_set_server_option()”
Section 20.6.7.69, “mysql_store_result()”

mysql_num_fields()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.18, “mysql_fetch_field_direct()”
Section 20.6.7.21, “mysql_fetch_row()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”

mysql_num_rows()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.9, “mysql_data_seek()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”
Section 20.6.14.2, “What Results You Can Get from a Query”

mysql_options()

Section 20.6.6, “C API Function Overview”
Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.15, “Controlling Automatic Reconnection Behavior”
Section B.5.2.9, “MySQL server has gone away”
Section 20.6.7.49, “mysql_options()”
Section 20.6.7.50, “mysql_ping()”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 6.3.1, “User Names and Passwords”
Section 5.5.4, “Using Client Programs in a Multiple-Server Environment”
Section 6.3.6, “Using Secure Connections”

mysql_ping()

Section 20.6.6, “C API Function Overview”
Section 20.6.15, “Controlling Automatic Reconnection Behavior”
Section B.5.2.9, “MySQL server has gone away”
Section 20.6.7.50, “mysql_ping()”
Section 20.6.7.70, “mysql_thread_id()”

mysql_query()

Section 20.6.6, “C API Function Overview”
Section 20.6.16, “C API Support for Multiple Statement Execution”
Section 13.2.1, “CALL Syntax”
Section 20.6.14.3, “How to Get the Unique ID for the Last Inserted Row”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.8, “mysql_create_db()”
Section 20.6.7.11, “mysql_drop_db()”
Section 20.6.7.17, “mysql_fetch_field()”
Section 20.6.7.38, “mysql_kill()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.7.51, “mysql_query()”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.7.54, “mysql_real_query()”
Section 20.6.7.56, “mysql_reload()”
Section 20.6.7.63, “mysql_set_local_infile_handler()”
Section 20.6.7.64, “mysql_set_server_option()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”
Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_real_connect()

Section 20.6.6, “C API Function Overview”
Section 20.6.16, “C API Support for Multiple Statement Execution”
Section 13.2.1, “CALL Syntax”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”
Chapter 12, *Functions and Operators*
Section 12.13, “Information Functions”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.3, “mysql_change_user()”
Section 20.6.7.7, “mysql_connect()”
Section 20.6.7.36, “mysql_init()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.7.49, “mysql_options()”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.7.64, “mysql_set_server_option()”
Section 20.6.7.66, “mysql_sqlstate()”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 20.6.7.67, “mysql_ssl_set()”
Section 5.1.4, “Server System Variables”
Section 13.5, “SQL Syntax for Prepared Statements”
Section 18.2.1, “Stored Routine Syntax”
Section 5.5.4, “Using Client Programs in a Multiple-Server Environment”

mysql_real_escape_string()

Section 20.6.6, “C API Function Overview”
Section 6.1.7, “Client Programming Security Guidelines”
Section 20.6.7.16, “mysql_escape_string()”
Section 20.6.7.53, “mysql_real_escape_string()”
Section 20.6.7.61, “mysql_set_character_set()”
Section 11.5.3.3, “Populating Spatial Columns”
Section 9.1.1, “String Literals”

mysql_real_query()

Section 20.6.6, “C API Function Overview”
Section 20.6.16, “C API Support for Multiple Statement Execution”
Section 13.2.1, “CALL Syntax”
Section 14.7.1, “Description of the FEDERATED Storage Engine”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.7.51, “mysql_query()”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.7.54, “mysql_real_query()”
Section 20.6.7.64, “mysql_set_server_option()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”

mysql_refresh()

Section 20.6.6, “C API Function Overview”

mysql_reload()

Section 20.6.6, “C API Function Overview”

mysql_rollback()

Section 20.6.6, “C API Function Overview”

mysql_row_seek()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.58, “mysql_row_seek()”
Section 20.6.7.59, “mysql_row_tell()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”

mysql_row_tell()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.58, “mysql_row_seek()”
Section 20.6.7.59, “mysql_row_tell()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”

mysql_select_db()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.60, “mysql_select_db()”

mysql_server_end()

Section 20.6.13, “C API Embedded Server Function Descriptions”
Section 20.6.6, “C API Function Overview”
Section 20.6.7.39, “mysql_library_end()”
Section 20.6.13.2, “mysql_server_end()”

mysql_server_init()

Section 20.6.13, “C API Embedded Server Function Descriptions”
Section 20.6.6, “C API Function Overview”
Section 20.6.12.1, “my_init()”
Section 20.6.7.40, “mysql_library_init()”
Section 20.6.13.1, “mysql_server_init()”
Section 20.6.12.3, “mysql_thread_init()”

mysql_set_character_set()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.26, “mysql_get_character_set_info()”
Section 20.6.7.53, “mysql_real_escape_string()”

mysql_set_local_infile_default()

Section 20.6.6, “C API Function Overview”

mysql_set_local_infile_handler()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.62, “mysql_set_local_infile_default()”
Section 20.6.7.63, “mysql_set_local_infile_handler()”

mysql_set_server_option()

Section 20.6.6, “C API Function Overview”
Section 20.6.16, “C API Support for Multiple Statement Execution”
Section 20.6.7.64, “mysql_set_server_option()”

mysql_shutdown()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.65, “mysql_shutdown()”
Section 6.2.1, “Privileges Provided by MySQL”

mysql_sqlstate()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.14, “mysql_errno()”
Section 20.6.7.66, “mysql_sqlstate()”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section B.2, “Types of Error Values”

mysql_ssl_set()

Section 20.6.6, “C API Function Overview”
Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”
Section 20.6.7.52, “mysql_real_connect()”
Section 20.6.7.67, “mysql_ssl_set()”
Section 6.3.6, “Using Secure Connections”

mysql_stat()

Section 20.6.6, “C API Function Overview”

mysql_stmt_affected_rows()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.1, “mysql_stmt_affected_rows()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.17, “mysql_stmt_num_rows()”

mysql_stmt_attr_get()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.2, “mysql_stmt_attr_get()”
Section 20.6.11.3, “mysql_stmt_attr_set()”

mysql_stmt_attr_set()

Section 20.6.5, “C API Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.9.2, “C API Prepared Statement Type Conversions”
Section 20.6.11.3, “mysql_stmt_attr_set()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.27, “mysql_stmt_store_result()”
Section C.2, “Restrictions on Server-Side Cursors”

mysql_stmt_bind_param()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.18, “C API Prepared Statement Handling of Date and Time Values”
Section 20.6.11.4, “mysql_stmt_bind_param()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.20, “mysql_stmt_prepare()”
Section 20.6.11.25, “mysql_stmt_send_long_data()”

mysql_stmt_bind_result()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.18, “C API Prepared Statement Handling of Date and Time Values”
Section 20.6.11.5, “mysql_stmt_bind_result()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.12, “mysql_stmt_fetch_column()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_close()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.6, “mysql_stmt_close()”
Section 20.6.11.15, “mysql_stmt_init()”

mysql_stmt_data_seek()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.7, “mysql_stmt_data_seek()”
Section 20.6.11.23, “mysql_stmt_row_seek()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_errno()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.8, “mysql_stmt_errno()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section B.2, “Types of Error Values”

mysql_stmt_error()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.9, “mysql_stmt_error()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.20, “mysql_stmt_prepare()”
Section B.2, “Types of Error Values”

mysql_stmt_execute()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.18, “C API Prepared Statement Handling of Date and Time Values”
Section 20.6.9.2, “C API Prepared Statement Type Conversions”
Section 20.6.11.1, “mysql_stmt_affected_rows()”
Section 20.6.11.3, “mysql_stmt_attr_set()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.11, “mysql_stmt_fetch()”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 20.6.11.25, “mysql_stmt_send_long_data()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_fetch()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.9.2, “C API Prepared Statement Type Conversions”
Section 20.6.11.5, “mysql_stmt_bind_result()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”
Section 20.6.11.24, “mysql_stmt_row_tell()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_fetch_column()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section B.4, “Client Error Codes and Messages”
Section 20.6.11.11, “mysql_stmt_fetch()”

mysql_stmt_field_count()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.13, “mysql_stmt_field_count()”

mysql_stmt_free_result()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.3, “mysql_stmt_attr_set()”
Section 20.6.11.14, “mysql_stmt_free_result()”

mysql_stmt_init()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.11, “C API Prepared Statement Function Descriptions”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.8, “C API Prepared Statements”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.20, “mysql_stmt_prepare()”

mysql_stmt_insert_id()

Section 20.6.10, “C API Prepared Statement Function Overview”

mysql_stmt_num_rows()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.7, “mysql_stmt_data_seek()”
Section 20.6.11.17, “mysql_stmt_num_rows()”

mysql_stmt_param_count()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.10, “mysql_stmt_execute()”

mysql_stmt_param_metadata()

Section 20.6.10, “C API Prepared Statement Function Overview”

mysql_stmt_prepare()

Section 20.6.9, “C API Prepared Statement Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.18, “C API Prepared Statement Handling of Date and Time Values”
Section 20.6.11.4, “mysql_stmt_bind_param()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.13, “mysql_stmt_field_count()”
Section 20.6.11.20, “mysql_stmt_prepare()”
Section 20.6.11.21, “mysql_stmt_reset()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”
Section 13.5, “SQL Syntax for Prepared Statements”

mysql_stmt_reset()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.3, “mysql_stmt_attr_set()”
Section 20.6.11.25, “mysql_stmt_send_long_data()”

mysql_stmt_result_metadata()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.9.2, “C API Prepared Statement Type Conversions”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_row_seek()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.23, “mysql_stmt_row_seek()”
Section 20.6.11.24, “mysql_stmt_row_tell()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_row_tell()

Section 20.6.10, “C API Prepared Statement Function Overview”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 20.6.11.23, “mysql_stmt_row_seek()”
Section 20.6.11.24, “mysql_stmt_row_tell()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_stmt_send_long_data()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section B.4, “Client Error Codes and Messages”
Section 20.6.11.21, “mysql_stmt_reset()”
Section 20.6.11.25, “mysql_stmt_send_long_data()”

mysql_stmt_sqlstate()

Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.26, “mysql_stmt_sqlstate()”
Section B.2, “Types of Error Values”

mysql_stmt_store_result()

Section 20.6.5, “C API Data Structures”
Section 20.6.10, “C API Prepared Statement Function Overview”
Section 20.6.11.3, “mysql_stmt_attr_set()”
Section 20.6.11.7, “mysql_stmt_data_seek()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.17, “mysql_stmt_num_rows()”
Section 20.6.11.23, “mysql_stmt_row_seek()”
Section 20.6.11.24, “mysql_stmt_row_tell()”
Section 20.6.11.27, “mysql_stmt_store_result()”

mysql_store_result()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section B.5.2.14, “Commands out of sync”
Section 14.7.1, “Description of the FEDERATED Storage Engine”
Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.9, “mysql_data_seek()”
Section 20.6.7.13, “mysql_eof()”
Section 20.6.7.17, “mysql_fetch_field()”
Section 20.6.7.21, “mysql_fetch_row()”
Section 20.6.7.22, “mysql_field_count()”
Section 20.6.7.25, “mysql_free_result()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.7.47, “mysql_num_fields()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.7.58, “mysql_row_seek()”
Section 20.6.7.59, “mysql_row_tell()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.22, “mysql_stmt_result_metadata()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”
Section 20.6.14.2, “What Results You Can Get from a Query”
Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_thread_end()

Section 20.6.6, “C API Function Overview”
Section 20.5, “libmysqld, the Embedded MySQL Server Library”
Section 20.6.12.2, “mysql_thread_end()”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_thread_id()

Section 20.6.6, “C API Function Overview”
Section 20.6.15, “Controlling Automatic Reconnection Behavior”
Section 20.6.7.50, “mysql_ping()”
Section 20.6.7.70, “mysql_thread_id()”

mysql_thread_init()

Section 20.6.6, “C API Function Overview”
Section 20.5, “libmysqld, the Embedded MySQL Server Library”
Section 20.6.12.1, “my_init()”
Section 20.6.12.2, “mysql_thread_end()”
Section 20.6.12.3, “mysql_thread_init()”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_thread_safe()

Section 20.6.6, “C API Function Overview”

mysql_use_result()

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section B.5.2.14, “Commands out of sync”
Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”
Section 20.6.7.9, “mysql_data_seek()”
Section 20.6.7.13, “mysql_eof()”
Section 20.6.7.21, “mysql_fetch_row()”
Section 20.6.7.25, “mysql_free_result()”
Section 20.6.7.46, “mysql_next_result()”
Section 20.6.7.47, “mysql_num_fields()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.7.58, “mysql_row_seek()”
Section 20.6.7.59, “mysql_row_tell()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section B.5.2.8, “Out of memory”
Section 20.6.14.2, “What Results You Can Get from a Query”
Section 20.6.4.2, “Writing C API Threaded Client Programs”

mysql_warning_count()

Section 20.6.6, “C API Function Overview”
Section 20.6.7.46, “mysql_next_result()”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section B.2, “Types of Error Values”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Command Index

Symbols

[index top]

4OS2.EXE

A

[index top]

aCC

Access

addgroup

adduser

ALL STATUS

APF

apt-get

autoconf

automake

autoreconf

autorun.sh

B

[index top]

bash

bison

bzr

C

[index top]

c++

c++filt

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

configure.js

configure; make; make install

copy

coreadm

cp

crash-me

cron

csh

cxx

D

[index top]

dd

df

drwtsn32.exe

DSPLIB

dump

dyld

This documentation is for an older version. If you're
This documentation is for an older version. If you're

E

[index top]

egcs

emerge

EXIT SINGLE USER MODE

F

[index top]

FCC

fsadm

G

[index top]

g++

gcc

gcc-c++

gdb

gmake

GnuPG

gnutar

gpg

grep

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're

make_win_src_distribution

md5

md5.exe

md5sum

memcache

memcached

memcached-1.2.5 directory:

memcached-tool

memcat

memcp

memflush

memrm

memslap

mgmd

mkdev aio

mkdev mysql

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mkdir

Section 13.1.6, “CREATE DATABASE Syntax”

more

Section 4.5.1.2, “mysql Commands”
Section 4.5.1.1, “mysql Options”

msql2mysql

Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.2, “`replace` — A String-Replacement Utility”

mv

Section 5.4.5, “Server Log Maintenance”

my_print_defaults

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.7, “MySQL Program Development Utilities”
Section 4.1, “Overview of MySQL Programs”

myisam_ftdump

Section 12.9, “Full-Text Search Functions”
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.1, “Overview of MySQL Programs”

myisamchk

Section 13.7.2.1, “ANALYZE TABLE Syntax”
Section 8.5.2, “Bulk Data Loading for MyISAM Tables”
Section 10.5, “Character Set Configuration”
Section 13.7.2.3, “CHECK TABLE Syntax”
Choosing an Installation Type
Section 14.1.3.3, “Compressed Table Characteristics”
Section 14.1.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 21.3.1, “Debugging a MySQL Server”
Section 13.2.2, “DELETE Syntax”
Section 14.1.3.2, “Dynamic Table Characteristics”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.11.4, “External Locking”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 2.15, “Installing MySQL on NetWare”
Section C.7.3, “Limits on Table Size”
Section 13.7.6.4, “LOAD INDEX INTO CACHE Syntax”
Section 21.3.1.7, “Making a Test Case If You Experience Table Corruption”
Section 8.3.7, “MyISAM Index Statistics Collection”
Section 14.1.1, “MyISAM Startup Options”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 14.1.3, “MyISAM Table Storage Formats”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 8.5.1, “Optimizing MyISAM Queries”
Section 4.6.3.4, “Other myisamchk Options”
Section 4.1, “Overview of MySQL Programs”
Section 14.1.4.2, “Problems from Tables Not Being Closed Properly”
Section 13.7.2.6, “REPAIR TABLE Syntax”
Section 5.1.3, “Server Command Options”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.7.5.18, “SHOW INDEX Syntax”
Section 13.7.5.33, “SHOW TABLE STATUS Syntax”
Section 8.5.3, “Speed of REPAIR TABLE Statements”
Section 14.1.3.1, “Static (Fixed-Length) Table Characteristics”
Section 8.12.1, “System Factors and Startup Parameter Tuning”
Section 1.3.2, “The Main Features of MySQL”
Section 14.1, “The MyISAM Storage Engine”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 21.3.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”

myisamchk *.MYI

Section 7.6.3, “How to Repair MyISAM Tables”

myisamchk tbl_name

Section 7.6.2, “How to Check MyISAM Tables for Errors”

myisamlog

Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 4.1, “Overview of MySQL Programs”

myisampack

Section 8.5.2, “Bulk Data Loading for MyISAM Tables”
Section 14.1.3.3, “Compressed Table Characteristics”
Section 13.1.10, “CREATE TABLE Syntax”
Section 8.11.4, “External Locking”
Section C.7.3, “Limits on Table Size”
Section 14.3.1, “MERGE Table Advantages and Disadvantages”
Section 14.1.3, “MyISAM Table Storage Formats”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 4.1, “Overview of MySQL Programs”
Section 13.1.10.4, “Silent Column Specification Changes”
Section 14.3, “The MERGE Storage Engine”
Section 14.1, “The MyISAM Storage Engine”

mysql

Section 1.8.2.5, “'--' as the Start of a Comment”
Section 6.3.2, “Adding User Accounts”
Section 2.20.5.5, “Alpha-DEC-UNIX Notes (Tru64)”
Section 14.2.6, “Backing Up and Recovering an InnoDB Database”
Section 7.1, “Backup and Recovery Types”
Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”
Changes Made by MySQL Installation Wizard
Choosing an Installation Type
Section 4.2.7, “Command-Line Options that Affect Option-File Handling”
Section 9.6, “Comment Syntax”
Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”
Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 3.1, “Connecting to and Disconnecting from the Server”
Section 4.6.10.6, “Connecting to MySQL Instance Manager”
Section 4.2.2, “Connecting to the MySQL Server”
Section 10.1.4, “Connection Character Sets and Collations”
Section 1.9.1, “Contributors to MySQL”
Section 20.6.15, “Controlling Automatic Reconnection Behavior”
Section 2.19.5, “Copying MySQL Databases to Another Machine”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 3.3.1, “Creating and Selecting a Database”
Section 2.10.4.6, “Customizing the PATH for MySQL Tools”
Section 21.3.2, “Debugging a MySQL Client”
Section 18.1, “Defining Stored Programs”
Section 2.2, “Determining Your Current MySQL Version”
Disabling mysql Auto-Reconnect
Section 2.19.2, “Downgrading MySQL”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 3.2, “Entering Queries”
Section 2.21, “Environment Variables”
Section 7.3, “Example Backup and Recovery Strategy”
Section 20.6.3, “Example C API Client Programs”
Section 3.6, “Examples of Common Queries”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Chapter 12, *Functions and Operators*
Section 13.7.1.3, “GRANT Syntax”
Section 13.8.3, “HELP Syntax”
Section 14.2.8.9, “How to Cope with Deadlocks”
Section B.5.1, “How to Determine What Is Causing a Problem”
Section 1.7, “How to Report Bugs or Problems”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section B.5.2.15, “Ignoring user”
Section 12.13, “Information Functions”
Section 14.2.1.1, “Initializing InnoDB”
Input-Line Editing
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 2.11, “Installing MySQL on OS X”
Section 4.2.1, “Invoking MySQL Programs”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section B.5.7, “Known Issues in MySQL”
Section 8.2.1.15, “LIMIT Query Optimization”
Section 13.4.2.2, “LOAD DATA FROM MASTER Syntax”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 13.4.2.3, “LOAD TABLE tbl_name FROM MASTER Syntax”
Section 4.4.2, “`make_win_bin_dist` — Package MySQL Distribution as Zip Archive”
Section 7.4.5.1, “Making a Copy of a Database”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 8.13.1, “Measuring the Speed of Expressions and Functions”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.3, “MySQL Cluster Configuration”
Section 17.2.4, “MySQL Cluster Example with Tables and Data”
Section 4.5.1.2, “mysql Commands”
Section 4.5.1.3, “mysql Logging”
Section 4.5.1.1, “mysql Options”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 10.6, “MySQL Server Time Zone Support”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 4.5.1.4, “mysql Server-Side Help”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 4.5.1.6, “mysql Tips”
Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 20.6.7.14, “mysql_errno()”
Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”
Section 20.6.7.66, “mysql_sqlstate()”
Section 4.4.8, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 17.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”
Section 4.2.9, “Option Defaults, Options Expecting Values, and the = Sign”
Section B.5.2.8, “Out of memory”
Section 4.1, “Overview of MySQL Programs”
Section B.5.2.10, “Packet Too Large”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 4.2.5, “Program Option Modifiers”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 7.4.2, “Reloading SQL-Format Backups”
Resetting the Root Password: Generic Instructions
Section 13.7.1.5, “REVOKE Syntax”
Section 2.18.4, “Securing the Initial MySQL Accounts”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 13.2.8.1, “SELECT ... INTO Syntax”
Section 5.1.3, “Server Command Options”
Section B.3, “Server Error Codes and Messages”
Section 5.1.4, “Server System Variables”
Section 5.1.8, “Server-Side Help”
Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 13.7.5.31, “SHOW SLAVE STATUS Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section 4.2.3, “Specifying Program Options”
Section 13.5, “SQL Syntax for Prepared Statements”
Section 2.10.4.7, “Starting MySQL as a Windows Service”
Section 2.10.3.1, “Starting the MySQL Server Instance Configuration Wizard”
Section 4.6.10.3, “Starting the MySQL Server with MySQL Instance Manager”
Section 2.18.2, “Starting the Server”
Section 9.1.1, “String Literals”
Section 2.10.8.4, “Testing a Windows Source Build”
Section 2.18.3, “Testing the Server”
Section 11.4.3, “The BLOB and TEXT Types”
Section 18.3.1, “Trigger Syntax and Examples”
Section 14.2.13.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Chapter 3, *Tutorial*
Section 1.2, “Typographical and Syntax Conventions”
Section 7.3.2, “Using Backups for Recovery”
Section 3.5, “Using mysql in Batch Mode”
Section 7.4, “Using mysqldump for Backups”
Section 4.2.6, “Using Option Files”
Section 4.2.4, “Using Options on the Command Line”
Section 4.2.8, “Using Options to Set Program Variables”
Section 21.3.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Using the --safe-updates Option
Section 2.10.6, “Windows Postinstallation Procedures”

mysql ...

Section 21.3.1.1, “Compiling MySQL for Debugging”

mysql mysql < mysql.dump

Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”

mysql stop

Section 4.6.10.3, “Starting the MySQL Server with MySQL Instance Manager”

mysql-server

Section 2.20.4.1, “FreeBSD Notes”

mysql-test-run.pl

Section 2.10.8.4, “Testing a Windows Source Build”
Section 21.1.2, “The MySQL Test Suite”

mysql-test-run.pl test_name

Section 21.1.2, “The MySQL Test Suite”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysql.exe

mysql.server

mysql.server stop

mysql_config

mysql_convert_table_format

mysql_explain_log

mysql_find_rows

mysql_fix_extensions

mysql_fix_privilege_tables

mysql_install_db

mysql_secure_installation

mysql_setpermission

mysql_setpermissions

mysql_stmt_execute()

mysql_stmt_prepare()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysql_tableinfo

Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”
Section 4.1, “Overview of MySQL Programs”

mysql_tzinfo_to_sql

Section 10.6, “MySQL Server Time Zone Support”
Section 4.4.8, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”
Section 4.1, “Overview of MySQL Programs”

mysql_upgrade

Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”
Section 2.19.2, “Downgrading MySQL”
Section 2.18.1, “Initializing the Data Directory”
Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.1, “Overview of MySQL Programs”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.3, “Server Command Options”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 2.19.1, “Upgrading MySQL”
Section 2.10.7, “Upgrading MySQL on Windows”

mysql_waitpid

Section 4.6.17, “`mysql_waitpid` — Kill Process and Wait for Its Termination”
Section 4.1, “Overview of MySQL Programs”

mysql_waitpid()

Section 4.6.17, “`mysql_waitpid` — Kill Process and Wait for Its Termination”

mysql_zap

Section B.5.2.2, “Can't connect to [local] MySQL server”
Section 4.6.18, “`mysql_zap` — Kill Processes That Match a Pattern”
Section 4.1, “Overview of MySQL Programs”

mysqlaccess

Section 1.9.1, “Contributors to MySQL”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.1, “Overview of MySQL Programs”
Section 2.18.2, “Starting the Server”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

mysqladmin

Section 6.3.5, “Assigning Account Passwords”
Section 16.3.1.1, “Backing Up a Slave Using mysqldump”
Section 2.20.4.6, “BSD/OS Version 4.x Notes”
Section B.5.2.2, “Can't connect to [local] MySQL server”
Section 4.2.2, “Connecting to the MySQL Server”
Section 1.9.1, “Contributors to MySQL”
Section 13.1.6, “CREATE DATABASE Syntax”
Section 2.10.4.6, “Customizing the PATH for MySQL Tools”
Section 21.3.1, “Debugging a MySQL Server”
Section 13.1.13, “DROP DATABASE Syntax”
Section 20.6.3, “Example C API Client Programs”
Section 13.7.6.2, “FLUSH Syntax”
Section B.5.1, “How to Determine What Is Causing a Problem”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 2.14, “Installing MySQL on i5/OS”
Section 2.15, “Installing MySQL on NetWare”
Section 2.11, “Installing MySQL on OS X”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.4, “MySQL Server Logs”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.1, “Overview of MySQL Programs”
Section 6.2.1, “Privileges Provided by MySQL”
Section 5.5.3, “Running Multiple MySQL Instances on Unix”
Section 2.18.4, “Securing the Initial MySQL Accounts”
Section 2.10.4.7, “Starting MySQL as a Windows Service”
Section 2.10.4.5, “Starting MySQL from the Windows Command Line”
Section 2.18.3, “Testing the Server”
Section 1.3.2, “The Main Features of MySQL”
Section 5.1.10, “The Server Shutdown Process”
Section 8.12.2, “Tuning Server Parameters”
Section 2.10.7, “Upgrading MySQL on Windows”
Section 4.2.6, “Using Option Files”
Section 4.2.4, “Using Options on the Command Line”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysqladmin create

Section 2.20.5.7, “SGI Irix Notes”

mysqladmin debug

Section 21.3.1, “Debugging a MySQL Server”
Section 13.7.1.3, “GRANT Syntax”
Section 6.2.1, “Privileges Provided by MySQL”

mysqladmin extended-status

Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.7.5.32, “SHOW STATUS Syntax”

mysqladmin flush-hosts

Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”
Section B.5.2.6, “Host 'host_name' is blocked”
Section 5.1.4, “Server System Variables”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

mysqladmin flush-logs

Section 7.3.3, “Backup Strategy Summary”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.4.5, “Server Log Maintenance”
Section 5.4.3, “The Binary Log”
Section 5.4.1, “The Error Log”
Section 16.2.2.1, “The Slave Relay Log”

mysqladmin flush-privileges

Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”
Section 2.19.5, “Copying MySQL Databases to Another Machine”
Section 6.2.2, “Grant Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 5.1.3, “Server Command Options”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 6.2.6, “When Privilege Changes Take Effect”

mysqladmin flush-tables

Section 8.5.2, “Bulk Data Loading for MyISAM Tables”
Section 8.11.4, “External Locking”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.5.1, “How MySQL Uses Memory”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 7.6.1, “Using myisamchk for Crash Recovery”

mysqladmin flush-xxx

Section 6.3.2, “Adding User Accounts”

mysqladmin kill

Section 2.20.5.6, “Alpha-DEC-OSF/1 Notes”
Section B.5.3.4, “How MySQL Handles a Full Disk”
Section 2.20.5.3, “IBM-AIX notes”
Section 13.7.6.3, “KILL Syntax”
Section 12.15, “Miscellaneous Functions”
Section B.5.2.9, “MySQL server has gone away”
Section 6.2.1, “Privileges Provided by MySQL”
Section C.7.6, “Windows Platform Limitations”

mysqladmin password

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

mysqladmin processlist

Section 6.3.2, “Adding User Accounts”
Section 8.14, “Examining Thread Information”
Section 13.7.6.3, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 21.1.1, “MySQL Threads”
Section 20.6.7.43, “mysql_list_processes()”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.7.5.27, “SHOW PROCESSLIST Syntax”

mysqladmin processlist status

Section 21.3.1, “Debugging a MySQL Server”

mysqladmin refresh

Section 6.3.2, “Adding User Accounts”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section B.5.7, “Known Issues in MySQL”
Section 5.4.5, “Server Log Maintenance”

mysqladmin reload

Section 6.3.2, “Adding User Accounts”
Section 6.2.2, “Grant Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 5.1.3, “Server Command Options”
Section 6.3.4, “Setting Account Resource Limits”
Section 6.2.6, “When Privilege Changes Take Effect”

mysqladmin reload version

Section 1.7, “How to Report Bugs or Problems”

mysqladmin shutdown

Section 6.2.5, “Access Control, Stage 2: Request Verification”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 2.20.5.6, “Alpha-DEC-OSF/1 Notes”
Section 21.3.1.2, “Creating Trace Files”
Section 13.7.1.3, “GRANT Syntax”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.20.5.3, “IBM-AIX notes”
Section 14.2.1.1, “Initializing InnoDB”
Section 2.11, “Installing MySQL on OS X”
Section 21.3.1.7, “Making a Test Case If You Experience Table Corruption”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.16, “Replication and Temporary Tables”
Section 17.2.5, “Safe Shutdown and Restart of MySQL Cluster”
Section 2.10.4.7, “Starting MySQL as a Windows Service”
Section 5.1.10, “The Server Shutdown Process”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”
Section C.7.6, “Windows Platform Limitations”

mysqladmin status

Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 20.6.7.68, “mysql_stat()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

mysqladmin variables

Section B.5.2.9, “MySQL server has gone away”
Section 13.7.5.36, “SHOW VARIABLES Syntax”

mysqladmin variables extended-status processlist

Section 1.7, “How to Report Bugs or Problems”

mysqladmin ver

Section 21.3.1.1, “Compiling MySQL for Debugging”

mysqladmin version

Section B.5.2.2, “Can't connect to [local] MySQL server”
Section 1.7, “How to Report Bugs or Problems”
Section B.5.2.9, “MySQL server has gone away”
Section 2.18.3, “Testing the Server”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”

mysqlanalyze

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlbinlog

Section 14.2.6, “Backing Up and Recovering an InnoDB Database”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section B.5.7, “Known Issues in MySQL”
Section 12.15, “Miscellaneous Functions”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”
Section 16.4.1.29, “Replication and Variables”
Section 13.7.4, “SET Syntax”
Section 13.7.5.2, “SHOW BINLOG EVENTS Syntax”
Section 13.4.2.7, “START SLAVE Syntax”
Section 5.4.3, “The Binary Log”
Section 16.2.2.1, “The Slave Relay Log”
Section 7.3.2, “Using Backups for Recovery”

mysqlbinlog binary-log-file | mysql

Section 21.3.1.7, “Making a Test Case If You Experience Table Corruption”

mysqlbinlog|mysql

Section B.5.7, “Known Issues in MySQL”

mysqlbug

Section 4.4.4, “`mysqlbug` — Generate Bug Report”

mysqlcheck

Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.1, “Overview of MySQL Programs”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.4, “Server System Variables”
Section 1.3.2, “The Main Features of MySQL”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 14.1, “The MyISAM Storage Engine”

mysqld

Section 21.2.2, “Adding a New User-Defined Function”
Section 21.2, “Adding New Functions to MySQL”
Section 2.20.5.6, “Alpha-DEC-OSF/1 Notes”
Section 2.20.5.5, “Alpha-DEC-UNIX Notes (Tru64)”
Section 14.2.6, “Backing Up and Recovering an InnoDB Database”
Section 14.5.3, “BDB Startup Options”
Section 16.1.2.4, “Binary Log Options and Variables”
Section 2.20.4.5, “BSD/OS Version 3.x Notes”
Section 2.20.4.6, “BSD/OS Version 4.x Notes”
Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 2.10.8.1, “Building MySQL from the Standard Source Distribution”
Section 6.3.6.2, “Building MySQL with Support for Secure Connections”
Section B.5.2.2, “Can't connect to [local] MySQL server”
Section B.5.2.13, “Can't create/write to file”
Section B.5.2.17, “Can't initialize character set”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 14.2.4, “Changing the Number or Size of InnoDB Redo Log Files”
Section 14.5.4, “Characteristics of BDB Tables”
Section 2.4.2.2, “Choosing a Distribution Format”
Section B.5.2.4, “Client does not support authentication protocol”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 9.6, “Comment Syntax”
Section B.5.2.11, “Communication Errors and Aborted Connections”
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 2.17.5, “Compiling and Linking an Optimized mysqld Server”
Section 21.3.1.1, “Compiling MySQL for Debugging”
Section 14.2.1, “Configuring InnoDB”
Section 14.1.4.1, “Corrupted MyISAM Tables”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”
Section 13.1.12, “CREATE VIEW Syntax”
Section 21.3.1.2, “Creating Trace Files”
Section 14.2.1.2, “Dealing with InnoDB Initialization Problems”
Section 2.17.4, “Dealing with Problems Compiling MySQL”
Section 21.3.1, “Debugging a MySQL Server”
Section 21.3, “Debugging and Porting MySQL”
Section 21.3.1.4, “Debugging mysqld under gdb”
Section 17.3.3.6, “Defining SQL and Other API Nodes in a MySQL Cluster”
Section 2.3.1, “Enterprise Server Distribution Types”
Section 2.21, “Environment Variables”
Section 14.5.6, “Errors That May Occur When Using BDB Tables”
Section 8.11.4, “External Locking”
Section B.5.2.18, “File Not Found and Similar Errors”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 13.7.6.2, “FLUSH Syntax”
Section 14.2.6.2, “Forcing InnoDB Recovery”
Section 2.20.4.1, “FreeBSD Notes”
Section 8.14.2, “General Thread States”
Section 15.2.3, “Handling MySQL Recovery with ZFS”
Section B.5.2.6, “Host 'host_name' is blocked”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.5.1, “How MySQL Uses Memory”
Section 8.2.1.17, “How to Avoid Full Table Scans”
Section B.5.1, “How to Determine What Is Causing a Problem”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.20.5.3, “IBM-AIX notes”
Section 9.2.2, “Identifier Case Sensitivity”
Section B.5.2.15, “Ignoring user”
Section 12.13, “Information Functions”
Section 17.2.2, “Initial Configuration of MySQL Cluster”
Section 17.2.3, “Initial Startup of MySQL Cluster”
Section 14.2.1.1, “Initializing InnoDB”
Section 2.18.1, “Initializing the Data Directory”
Section 14.2.11.1, “InnoDB Disk I/O”
Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”
Section 14.2.13.2, “InnoDB General Troubleshooting”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 2.7, “Installation Layouts”
Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 17.2.1, “Installing MySQL Cluster on Linux”
Section 2.14, “Installing MySQL on i5/OS”
Section 2.12, “Installing MySQL on Linux Using RPM Packages”
Section 2.11, “Installing MySQL on OS X”
Section 2.16, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section 13.7.6.3, “KILL Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 2.20.1.7, “Linux Alpha Notes”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 2.20.1.2, “Linux Binary Distribution Notes”
Section 2.20.1.1, “Linux Operating System Notes”
Section 2.20.1.4, “Linux Postinstallation Notes”
Section 2.20.1.3, “Linux Source Distribution Notes”
Section 2.20.1.5, “Linux x86 Notes”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 21.3.1.7, “Making a Test Case If You Experience Table Corruption”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 14.1.1, “MyISAM Startup Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section A.1, “MySQL 5.0 FAQ: General”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section A.3, “MySQL 5.0 FAQ: Server SQL Mode”
Section 17.5.10.3, “MySQL Cluster and MySQL Security Procedures”
Section 17.3, “MySQL Cluster Configuration”
Section 17.3.3.1, “MySQL Cluster Configuration: Basic Example”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.2, “MySQL Cluster Installation and Upgrades”
Section 17.3.2.5, “MySQL Cluster mysqld Option and Variable Reference”
Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 17.4, “MySQL Cluster Programs”
MySQL Cluster System Variables
Section 4.6.10.1, “MySQL Instance Manager Command Options”
Section 4.6.10.2, “MySQL Instance Manager Configuration Files”
Chapter 5, *MySQL Server Administration*
Section 4.3, “MySQL Server and Server-Startup Programs”
Section B.5.2.9, “MySQL server has gone away”
Section 5.4, “MySQL Server Logs”
Section 10.6, “MySQL Server Time Zone Support”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 1.8, “MySQL Standards Compliance”
Section 21.1.1, “MySQL Threads”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 20.6.7.1, “mysql_affected_rows()”
Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 20.6.7.49, “mysql_options()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
mysqld Command Options for MySQL Cluster
Section 4.3.1, “`mysqld` — The MySQL Server”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 17.4.17, “`ndb_show_tables` — Display List of NDB Tables”
Section 14.5.1, “Operating Systems Supported by BDB”
Section 13.7.2.5, “OPTIMIZE TABLE Syntax”
Section B.5.5, “Optimizer-Related Issues”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 2.20.2.1, “OS X 10.x (Darwin)”
Section 17.3.2, “Overview of MySQL Cluster Configuration Parameters, Options, and Variables”
Section 4.1, “Overview of MySQL Programs”
Section B.5.2.10, “Packet Too Large”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 14.1.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.18.1.1, “Problems Running mysql_install_db”
Section B.5.3.1, “Problems with File Permissions”
Section 4.2.5, “Program Option Modifiers”
Section 8.10.3.3, “Query Cache Configuration”
Section 16.1.2.1, “Replication and Binary Logging Option and Variable Reference”
Section 16.1.2, “Replication and Binary Logging Options and Variables”
Section 16.1.2.2, “Replication Master Options and Variables”
Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems
Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”
Section 14.5.5, “Restrictions on BDB Tables”
Section B.5.4.5, “Rollback Failure for Nontransactional Tables”
Section 5.5, “Running Multiple MySQL Instances on One Machine”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 2.20.5.8, “SCO UNIX and OpenServer 5.0.x Notes”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 6.1.4, “Security-Related mysqld Options and Variables”
Section 13.2.8.1, “SELECT ... INTO Syntax”
Section 2.10.4.3, “Selecting a MySQL Server Type”
Section 10.1.3.1, “Server Character Set and Collation”
Section 5.1.3, “Server Command Options”
Section 5.1.9, “Server Response to Signals”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Syntax”
Section 10.2, “Setting the Error Message Language”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 2.20.5.7, “SGI Irix Notes”
Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 20.6.2, “Simultaneous MySQL Server and Connector/C Installations”
Section 2.20.3, “Solaris Notes”
Section 2.20.3.2, “Solaris x86 Notes”
Section 2.18.5, “Starting and Stopping MySQL Automatically”
Section 5.5.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.5.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.10.4.7, “Starting MySQL as a Windows Service”
Section 2.10.4.5, “Starting MySQL from the Windows Command Line”
Section 4.6.10.3, “Starting the MySQL Server with MySQL Instance Manager”
Section 10.6.1, “Staying Current with Time Zone Changes”
Section 2.20.5.4, “SunOS 4 Notes”
Section 1.9.5, “Supporters of MySQL”
Section 16.3.6, “Switching Masters During Failover”
Section 8.11.2, “Table Locking Issues”
Section B.5.2.19, “Table-Corruption Issues”
Section 2.10.8.4, “Testing a Windows Source Build”
Section 2.10.4.8, “Testing The MySQL Installation”
Section 2.18.3, “Testing the Server”
Section 5.4.3, “The Binary Log”
Section 14.10, “The BLACKHOLE Storage Engine”
Section 5.4.1, “The Error Log”
Section 5.4.2, “The General Query Log”
Section 14.1, “The MyISAM Storage Engine”
Section 8.10.3, “The MySQL Query Cache”
Section 5.1, “The MySQL Server”
Section 21.1.2, “The MySQL Test Suite”
Section 5.4.4, “The Slow Query Log”
Section B.5.3.7, “Time Zone Problems”
Section B.5.2.7, “Too many connections”
Section 2.10.5, “Troubleshooting a MySQL Installation Under Windows”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 8.12.2, “Tuning Server Parameters”
Section 1.2, “Typographical and Syntax Conventions”
Section 21.2.2.5, “UDF Compiling and Installing”
Section 21.2.2.6, “UDF Security Precautions”
Section 2.19.1, “Upgrading MySQL”
Section 2.10.7, “Upgrading MySQL on Windows”
Section 21.3.1.5, “Using a Stack Trace”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 4.2.6, “Using Option Files”
Section 21.3.1.3, “Using pdb to create a Windows crashdump”
Section 21.3.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.12.4.3, “Using Symbolic Links for Databases on Windows”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 17.1.4, “What is New in MySQL Cluster”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”
Section 6.2.6, “When Privilege Changes Take Effect”
Section B.5.3.5, “Where MySQL Stores Temporary Files”

mysqld mysqld.trace

Section 21.3.1.2, “Creating Trace Files”

mysqld-abc.exe

Section 4.4.2, “`make_win_bin_dist` — Package MySQL Distribution as Zip Archive”

mysqld-debug

Section 2.4.2.2, “Choosing a Distribution Format”
Section 21.3.1.2, “Creating Trace Files”
Section 2.3.1, “Enterprise Server Distribution Types”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 2.10.4.3, “Selecting a MySQL Server Type”
Section 5.1.3, “Server Command Options”
Section 5.5.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”

mysqld-max

Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysqld-nt

Section 2.10.4.3, “Selecting a MySQL Server Type”
Section 5.1.3, “Server Command Options”
Section 5.5.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.5.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”

mysqld_multi

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.6.10, “`mysqlmanager` — The MySQL Instance Manager”
Section 4.1, “Overview of MySQL Programs”
Section 5.5.3, “Running Multiple MySQL Instances on Unix”

mysqld_safe

Section 2.20.5.5, “Alpha-DEC-UNIX Notes (Tru64)”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 21.3.1.1, “Compiling MySQL for Debugging”
Section 8.12.5.2, “Enabling Large Page Support”
Section B.5.2.18, “File Not Found and Similar Errors”
Section 2.20.4.1, “FreeBSD Notes”
Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 2.20.5.3, “IBM-AIX notes”
Section 14.2.1.1, “Initializing InnoDB”
Section 14.2.13.2, “InnoDB General Troubleshooting”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 2.14, “Installing MySQL on i5/OS”
Section 2.15, “Installing MySQL on NetWare”
Section 2.11, “Installing MySQL on OS X”
Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”
Section B.5.7, “Known Issues in MySQL”
Section 2.20.1.4, “Linux Postinstallation Notes”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.5.10.3, “MySQL Cluster and MySQL Security Procedures”
Section 10.6, “MySQL Server Time Zone Support”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.2.9, “Option Defaults, Options Expecting Values, and the = Sign”
Section 4.1, “Overview of MySQL Programs”
Section B.5.2.10, “Packet Too Large”
Section B.5.3.1, “Problems with File Permissions”
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.5.3, “Running Multiple MySQL Instances on Unix”
Section 2.20.5.8, “SCO UNIX and OpenServer 5.0.x Notes”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 2.18.5, “Starting and Stopping MySQL Automatically”
Section 4.6.10.3, “Starting the MySQL Server with MySQL Instance Manager”
Section 2.18.2, “Starting the Server”
Section 2.18.3, “Testing the Server”
Section 5.4.1, “The Error Log”
Section B.5.3.7, “Time Zone Problems”
Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 8.12.2, “Tuning Server Parameters”
Section 21.2.2.5, “UDF Compiling and Installing”
Section 4.2.6, “Using Option Files”

mysqldump

Section 16.3.1.1, “Backing Up a Slave Using mysqldump”
Section 14.2.6, “Backing Up and Recovering an InnoDB Database”
Chapter 7, *Backup and Recovery*
Section 7.1, “Backup and Recovery Types”
Section 7.3.3, “Backup Strategy Summary”
Section 13.7.2.2, “BACKUP TABLE Syntax”
Section 8.6.4, “Bulk Data Loading for InnoDB Tables”
Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Choosing an Installation Type
Section 4.2.2, “Connecting to the MySQL Server”
Section 1.9.1, “Contributors to MySQL”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 2.19.5, “Copying MySQL Databases to Another Machine”
Section 13.1.8, “CREATE INDEX Syntax”
Section 13.1.12, “CREATE VIEW Syntax”
Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 2.10.4.6, “Customizing the PATH for MySQL Tools”
Section 7.2, “Database Backup Methods”
Section 12.17.2, “DECIMAL Data Type Characteristics”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 14.2.11.3, “Defragmenting a Table”
Section 2.19.2, “Downgrading MySQL”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 15.1.2, “EC2 Instance Limitations”
Section 14.5.6, “Errors That May Occur When Using BDB Tables”
Section 7.3.1, “Establishing a Backup Policy”
Section 7.3, “Example Backup and Recovery Strategy”
Section 1.7, “How to Report Bugs or Problems”
Section 16.1.1, “How to Set Up Replication”
Section 9.2.2, “Identifier Case Sensitivity”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 13.4.2.2, “LOAD DATA FROM MASTER Syntax”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 13.4.2.3, “LOAD TABLE tbl_name FROM MASTER Syntax”
Section 7.4.5.1, “Making a Copy of a Database”
Section 14.2.7, “Moving an InnoDB Database to Another Machine”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.2.4, “MySQL Cluster Example with Tables and Data”
Section 17.1, “MySQL Cluster Overview”
Section 4.5.1.1, “mysql Options”
Section 5.4, “MySQL Server Logs”
Section 7.4.5, “mysqldump Tips”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 11.1.1, “Numeric Type Overview”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 17.5.3, “Online Backup of MySQL Cluster”
Section 4.1, “Overview of MySQL Programs”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section B.5.4.8, “Problems with Floating-Point Values”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 16.3.4, “Replicating Different Databases to Different Slaves”
Section 14.2.5, “Resizing the InnoDB System Tablespace”
Section 14.5.5, “Restrictions on BDB Tables”
Section C.4, “Restrictions on Views”
Section 5.4.5, “Server Log Maintenance”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 16.1.1.8, “Setting Up Replication with Existing Data”
Section B.5.4.7, “Solving Problems with No Matching Rows”
Section 4.2.3, “Specifying Program Options”
Section 2.10.4.7, “Starting MySQL as a Windows Service”
Section 2.10.3.1, “Starting the MySQL Server Instance Configuration Wizard”
Section 11.4.3, “The BLOB and TEXT Types”
Section 8.10.2, “The InnoDB Buffer Pool”
Section 1.3.2, “The Main Features of MySQL”
Section 10.6.2, “Time Zone Leap Second Support”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 17.2.6, “Upgrading and Downgrading MySQL Cluster”
Section 2.19.1, “Upgrading MySQL”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”
Section 7.4, “Using mysqldump for Backups”
Section 16.3.1, “Using Replication for Backups”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”
Section 11.3.4, “YEAR(2) Limitations and Migrating to YEAR(4)”

mysqldump mysql

Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

mysqldumpslow

Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 5.4.4, “The Slow Query Log”

mysqlfailover

Section 16.3.6, “Switching Masters During Failover”

mysqlhotcopy

Chapter 7, *Backup and Recovery*
Section 7.1, “Backup and Recovery Types”
Section 13.7.2.2, “BACKUP TABLE Syntax”
Section 1.9.1, “Contributors to MySQL”
Section 7.2, “Database Backup Methods”
Section 13.4.2.2, “LOAD DATA FROM MASTER Syntax”
Section 13.4.2.3, “LOAD TABLE tbl_name FROM MASTER Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

mysqlimport

mysqlmanager

mysqloptimize

mysqlrepair

mysqlshow

mysqlshow db_name

mysqlshow db_name tbl_name

mysqlshow mysql user

mysqltest

N

[index top]

ndb_config

ndb_cpcd

ndb_delete_all

ndb_desc

ndb_drop_index

ndb_drop_table

ndb_error_reporter

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ndb_mgm

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”
Section 17.2.3, “Initial Startup of MySQL Cluster”
Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 17.2.1, “Installing MySQL Cluster on Linux”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.5.6.1, “MySQL Cluster Logging Management Commands”
Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 17.4, “MySQL Cluster Programs”
Section 17.5.10.1, “MySQL Cluster Security and Networking Issues”
mysqld Command Options for MySQL Cluster
Section 17.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”
Section 17.4.2, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.5.3, “Online Backup of MySQL Cluster”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.2.5, “Safe Shutdown and Restart of MySQL Cluster”
Section 4.2.4, “Using Options on the Command Line”
Section 17.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”

ndb_mgmd

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 17.3.3.4, “Defining a MySQL Cluster Management Server”
Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”
Section 17.2.3, “Initial Startup of MySQL Cluster”
Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 17.2.1, “Installing MySQL Cluster on Linux”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.3.3.1, “MySQL Cluster Configuration: Basic Example”
Section 17.3.3.2, “MySQL Cluster Connection Strings”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.5.6.1, “MySQL Cluster Logging Management Commands”
Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 17.4, “MySQL Cluster Programs”
mysqld Command Options for MySQL Cluster
Section 17.4.2, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.3.1, “Quick Test Setup of MySQL Cluster”
Section 17.2.5, “Safe Shutdown and Restart of MySQL Cluster”
Section 17.5.1, “Summary of MySQL Cluster Start Phases”

ndb_print_backup_file

Section 17.4.11, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 17.4.12, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 17.4.13, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

ndb_print_schema_file

Section 17.4.11, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 17.4.12, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 17.4.13, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

ndb_print_sys_file

Section 17.4.11, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 17.4.12, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 17.4.13, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

ndb_restore

Section 7.1, “Backup and Recovery Types”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.1, “MySQL Cluster Overview”
Section 17.4, “MySQL Cluster Programs”
Section 17.5.8, “MySQL Cluster Single User Mode”
Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.5.3, “Online Backup of MySQL Cluster”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.2.6, “Upgrading and Downgrading MySQL Cluster”

ndb_schema_backup_file

Section 17.4.12, “`ndb_print_schema_file` — Print NDB Schema File Contents”

ndb_select_all

Section 17.5.10.3, “MySQL Cluster and MySQL Security Procedures”
Section 17.4.15, “`ndb_select_all` — Print Rows from an NDB Table”
Section 17.4.17, “`ndb_show_tables` — Display List of NDB Tables”

ndb_select_count

Section 17.4.16, “`ndb_select_count` — Print Row Counts for NDB Tables”

ndb_show_tables

Section 17.5.10.3, “MySQL Cluster and MySQL Security Procedures”
Section 17.4, “MySQL Cluster Programs”
mysqld Command Options for MySQL Cluster
Section 17.4.17, “`ndb_show_tables` — Display List of NDB Tables”

ndb_size.pl

Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
mysqld Command Options for MySQL Cluster
Section 17.4.18, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”
Section 2.22, “Perl Installation Notes”

ndb_waiter

Section 17.4.19, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”

ndbd

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 17.5.2, “Commands in the MySQL Cluster Management Client”
Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”
Section 17.2.3, “Initial Startup of MySQL Cluster”
Section 17.2.1.1, “Installing a MySQL Cluster Binary Release on Linux”
Section 17.2.1.2, “Installing MySQL Cluster from RPM”
Section 17.2.1, “Installing MySQL Cluster on Linux”
Section 17.1.5.9, “Limitations Relating to Multiple MySQL Cluster Nodes”
Section 17.5, “Management of MySQL Cluster”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.3.3.1, “MySQL Cluster Configuration: Basic Example”
Section 17.1.1, “MySQL Cluster Core Concepts”
Section 17.2, “MySQL Cluster Installation and Upgrades”
Section 17.3.4.2, “MySQL Cluster Interconnects and Performance”
Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 17.4, “MySQL Cluster Programs”
mysqld Command Options for MySQL Cluster
Section 17.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”
Section 17.4.19, “`ndb_waiter` — Wait for MySQL Cluster to Reach a Given Status”
Section 17.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”
Section 17.3.2, “Overview of MySQL Cluster Configuration Parameters, Options, and Variables”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 17.1.5.10, “Previous MySQL Cluster Issues Resolved in MySQL 5.0”
Section 17.3.1, “Quick Test Setup of MySQL Cluster”
Section 17.2.5, “Safe Shutdown and Restart of MySQL Cluster”
Section 17.3.3.11, “SCI Transport Connections in MySQL Cluster”
Section 17.5.1, “Summary of MySQL Cluster Start Phases”
Section 17.5.6.3, “Using CLUSTERLOG STATISTICS in the MySQL Cluster Management Client”
Section 17.3.4, “Using High-Speed Interconnects with MySQL Cluster”

NET

Section 2.10.4.7, “Starting MySQL as a Windows Service”

NET START

Section 5.5.2.2, “Starting Multiple MySQL Instances as Windows Services”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

rm

rpm

rpmbuild

rsync

S

[index top]

safe_mysqld

scp

sed

SELECT

Service Control Manager

Services

setenv

setrlimit

sh

SHOW

SHOW ERRORS

SHOW WARNINGS

ssh

Start>Run>cmd.exe

strings

su root

sudo

sysctl

This documentation is for an older version. If you're
This documentation is for an older version. If you're

T

[index top]

tar

tcpdump

tcsh

tee

Telnet

telnet

Text in this style

top

U

[index top]

ulimit

update-rc.d

UPGMYSQL

useradd

usermod

V

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Function Index

Symbols

[index top]

%
Section 1.8.1, “MySQL Extensions to Standard SQL”

A

[index top]

ABS()
Section 21.2, “Adding New Functions to MySQL”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”
Section 9.2.3, “Function Name Parsing and Resolution”
Section 12.6.2, “Mathematical Functions”

ACOS()
Section 12.6.2, “Mathematical Functions”

add()
Section 15.3.3.1, “Basic `memcached` Operations”

ADDDATE()
Section 12.7, “Date and Time Functions”

addslashes()
Section 6.1.7, “Client Programming Security Guidelines”

ADDTIME()
Section 12.7, “Date and Time Functions”

AES_DECRYPT()
Section 12.12, “Encryption and Compression Functions”

AES_ENCRYPT()
Section 12.12, “Encryption and Compression Functions”

Area()
Section 12.14.7, “Geometry Property Functions”
Section 12.14.7.4, “Polygon and MultiPolygon Property Functions”

AsBinary()
Section 11.5.3.4, “Fetching Spatial Data”
Section 12.14.6, “Geometry Format Conversion Functions”

ASCII()
Section 13.8.3, “HELP Syntax”
Section 12.5, “String Functions”

ASIN()
Section 12.6.2, “Mathematical Functions”

AsText()
Section 11.5.3.4, “Fetching Spatial Data”
Section 12.14.6, “Geometry Format Conversion Functions”

AsWKB()
Section 12.14.6, “Geometry Format Conversion Functions”

AsWKT()
Section 12.14.6, “Geometry Format Conversion Functions”

ATAN()
Section 12.6.2, “Mathematical Functions”

ATAN2()
Section 12.6.2, “Mathematical Functions”

AVG()
Section 11.1.2, “Date and Time Type Overview”
Section 12.16.1, “GROUP BY (Aggregate) Function Descriptions”
Section 11.4.4, “The ENUM Type”
Section 1.3.2, “The Main Features of MySQL”
Section 11.4.5, “The SET Type”

B

[index top]

BENCHMARK()
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.13, “Information Functions”
Section 8.13.1, “Measuring the Speed of Expressions and Functions”
Section 13.2.9.10, “Optimizing Subqueries”
Section 13.2.9.8, “Subqueries in the FROM Clause”

BIN()
Section 9.1.6, “Bit-Field Literals”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

CONCAT_WS()

CONNECTION_ID()

Contains()

CONV()

CONVERT()

CONVERT_TZ()

COS()

COT()

COUNT()

CRC32()

Crosses()

crypt()

CURDATE()

CURRENT_DATE

CURRENT_DATE()

CURRENT_TIME

CURRENT_TIME()

CURRENT_TIMESTAMP

This documentation is for an older version. If you're
This documentation is for an older version. If you're

CURRENT_TIMESTAMP()

CURRENT_USER

CURRENT_USER()

CURTIME()

D

[index top]

DATABASE()

DATE()

DATE_ADD()

DATE_FORMAT()

DATE_SUB()

DATEDIFF()

DAY()

DAYNAME()

DAYOFMONTH()

DAYOFWEEK()

DAYOFYEAR()

DECODE()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

MBREqual()

Section 12.14.9.3, “MySQL-Specific Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)”

MBRIntersects()

Section 12.14.9.3, “MySQL-Specific Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)”

MBROverlaps()

Section 12.14.9.3, “MySQL-Specific Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)”

MBRTouches()

Section 12.14.9.3, “MySQL-Specific Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)”

MBRWithin()

Section 12.14.9.3, “MySQL-Specific Spatial Relation Functions That Use Minimum Bounding Rectangles (MBRs)”
Section 11.5.3.7, “Using Spatial Indexes”

MD5()

Section 12.12, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 9.2, “Schema Object Names”
Section 6.1.1, “Security Guidelines”

MICROSECOND()

Section 12.7, “Date and Time Functions”

MID()

Section 10.1.9.1, “Result Strings”
Section 12.5, “String Functions”

MIN()

Section 20.6.17, “C API Prepared Statement Problems”
Section 12.16.1, “GROUP BY (Aggregate) Function Descriptions”
Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
Section 8.3.1, “How MySQL Uses Indexes”
Section B.5.7, “Known Issues in MySQL”
Loose Index Scan
Section 12.16.3, “MySQL Handling of GROUP BY”
Section 11.1.1, “Numeric Type Overview”
Section 13.2.9.10, “Optimizing Subqueries”
Section B.5.4.3, “Problems with NULL Values”
Section 1.3.2, “The Main Features of MySQL”
Section 11.3.8, “Two-Digit Years in Dates”
Section 18.4.3, “Updatable and Insertable Views”
Section 18.4.2, “View Processing Algorithms”

MINUTE()

Section 12.7, “Date and Time Functions”

MLineFromText()

Section 12.14.3, “Functions That Create Geometry Values from WKT Values”

MLineFromWKB()

Section 12.14.4, “Functions That Create Geometry Values from WKB Values”

MOD()

Section 12.6.1, “Arithmetic Operators”
Section 3.3.4.5, “Date Calculations”
Section 12.6.2, “Mathematical Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 5.1.7, “Server SQL Modes”

MONTH()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”

MONTHNAME()

Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 12.7, “Date and Time Functions”
Section 10.7, “MySQL Server Locale Support”
Section 5.1.4, “Server System Variables”

MPointFromText()

Section 12.14.3, “Functions That Create Geometry Values from WKT Values”

MPointFromWKB()

Section 12.14.4, “Functions That Create Geometry Values from WKB Values”

MPolyFromText()

Section 12.14.3, “Functions That Create Geometry Values from WKT Values”

MPolyFromWKB()

Section 12.14.4, “Functions That Create Geometry Values from WKB Values”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

P

[index top]

Q

[index top]

R

[index top]

This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

V

[index top]

VALUES()

VAR_POP()

VAR_SAMP()

VARIANCE()

VERSION()

W

[index top]

WEEK()

WEEKDAY()

WEEKOFYEAR()

Within()

X

[index top]

X()

Y

[index top]

Y()

YEAR()

YEARWEEK()

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INFORMATION_SCHEMA Index

C

[index top]

I

[index top]

K

[index top]

P

[index top]

R

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

S

[index top]

SCHEMA_PRIVILEGES

SCHEMATA

STATISTICS

T

[index top]

TABLE_CONSTRAINTS

TABLE_PRIVILEGES

TABLES

TRIGGERS

U

[index top]

USER_PRIVILEGES

V

[index top]

VIEWS

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Join Types Index

A | C | E | F | I | R | S | U

A

[index top]

ALL

C

[index top]

const

E

[index top]

eq_ref

F

[index top]

fulltext

I

[index top]

index

index_merge

index_subquery

R

[index top]

range

ref

ref_or_null

S

[index top]

system

U

[index top]

unique_subquery

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Operator Index

Symbols | A | B | C | D | E | I | L | N | O | R | X

Symbols

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

/

:=

=

^

|

||

~

A

[index top]

AND

B

[index top]

BETWEEN

BINARY

BINARY str

C

[index top]

CASE

CASE value WHEN END

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

LIKE '_A%'

LIKE 'pattern'

LIKE ... ESCAPE

N

[index top]

N % M

N MOD M

NOT

NOT LIKE

NOT REGEXP

NOT RLIKE

O

[index top]

OR

R

[index top]

REGEXP

RLIKE

X

[index top]

XOR

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Option Index

Symbols | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Z

Symbols

[index top]

--

-#

-1

-?

A

[index top]

-A

-a

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--abort-slave-event-count

--add-drop-database

--add-drop-table

--add-locks

--addtodest

--all-databases

--all-in-1

--allow-keywords

--allow-suspicious-udfs

--allowold

--analyze

--angel-pid-file

--ansi

--append

--auto-rehash

--auto-repair

--autoclose

B

[index top]

-B

This documentation is for an older version. If you're
This documentation is for an older version. If you're

-b

--back_log

--backup

--backup_id

--backup_path

backup_path

--backupid

--basedir

basedir

--batch

--bdb-home

--bdb-lock-detect

--bdb-logdir

--bdb-no-recover

--bdb-no-sync

--bdb-shared-data

--bdb-tmpdir

--big-tables

--bind-address

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--binlog-do-db

--binlog-ignore-db

--block-search

--bootstrap

--brief

--builddir

C

[index top]

-C

-c

--cflags

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--character-set-client-handshake

--character-set-filesystem

--character-set-server

--character-sets-dir

--charset

--check

--check-only-changed

--check-upgrade

--checkpoint

--chroot

--clear

--clear-only

--col

--collation-server

--column-names

--columns

--comments

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--commit
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”

--comp
Section 4.2.3, “Specifying Program Options”

--compact
Section 4.5.4, “mysqldump — A Database Backup Program”

--compatible
Section 4.5.4, “mysqldump — A Database Backup Program”

--complete-insert
Section 4.5.4, “mysqldump — A Database Backup Program”

--compr
Section 4.2.3, “Specifying Program Options”

--compress
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 4.5.1.1, “mysql Options”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.2.3, “Specifying Program Options”

--config-file
Section 17.2.3, “Initial Startup of MySQL Cluster”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.3.3.1, “MySQL Cluster Configuration: Basic Example”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 17.4.2, “ndb_mgmd — The MySQL Cluster Management Server Daemon”

--connect-string
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

--connections
Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”

--console
Section 14.2.1.1, “Initializing InnoDB”
Section 14.2.13.2, “InnoDB General Troubleshooting”
Resetting the Root Password: Windows Systems
Section 5.1.3, “Server Command Options”
Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 2.10.4.4, “Starting the Server for the First Time”
Section 5.4.1, “The Error Log”

--copy
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”

--core-file
Section 21.3.1.4, “Debugging mysqld under gdb”
Section 2.20.1.4, “Linux Postinstallation Notes”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 5.1.3, “Server Command Options”

--core-file-size
Section 2.20.1.4, “Linux Postinstallation Notes”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 5.1.3, “Server Command Options”

--correct-checksum
Section 4.6.3.3, “myisamchk Repair Options”

--count
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

--create-options
Section 4.5.4, “mysqldump — A Database Backup Program”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 2.10.8.4, “Testing a Windows Source Build”
Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 4.2.6, “Using Option Files”

datadir

Section 2.10.4.2, “Creating an Option File”
Section 15.1.1, “Setting Up MySQL on an EC2 AMI”
Section 4.6.10.3, “Starting the MySQL Server with MySQL Instance Manager”
Section 2.10.5, “Troubleshooting a MySQL Installation Under Windows”
Section C.7.6, “Windows Platform Limitations”

--date

Section 4.6.12, “`mysql_explain_log` — Use EXPLAIN on Statements in Query Log”

--db

Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”

--debug

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 21.3.1.1, “Compiling MySQL for Debugging”
Section 4.4.2, “`make_win_bin_dist` — Package MySQL Distribution as Zip Archive”
Section 4.4.3, “`make_win_src_distribution` — Create Source Distribution for Windows”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Options”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.6.6, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.8, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 5.1.3, “Server Command Options”
Section 2.10.4.5, “Starting MySQL from the Windows Command Line”
Section 21.3.3, “The DBUG Package”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”

--debug-info

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.5.1.1, “mysql Options”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--default-character-set

Section 10.5, “Character Set Configuration”
Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 10.1.4, “Connection Character Sets and Collations”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 4.5.1.1, “mysql Options”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 6.3.1, “User Names and Passwords”

--default-collation

Section 5.1.3, “Server Command Options”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--default-mysqld-path

Section 4.6.10.1, “MySQL Instance Manager Command Options”

--default-storage-engine

Section 14.2.1, “Configuring InnoDB”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Chapter 14, *Storage Engines*

--default-table-type

Section 5.1.3, “Server Command Options”
Chapter 14, *Storage Engines*

--default-time-zone

Section 10.6, “MySQL Server Time Zone Support”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

--default.key_buffer_size

Section 5.1.5.1, “Structured System Variables”

--defaults-extra-file

Section 4.2.7, “Command-Line Options that Affect Option-File Handling”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Options”
Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 5.1.3, “Server Command Options”
Section 4.2.6, “Using Option Files”

--defaults-file

Section 4.2.7, “Command-Line Options that Affect Option-File Handling”
Section 14.2.1, “Configuring InnoDB”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.10.1, “MySQL Instance Manager Command Options”
Section 4.6.10.7, “MySQL Instance Manager Commands”
Section 4.6.10.2, “MySQL Instance Manager Configuration Files”
Section 4.5.1.1, “mysql Options”
Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Resetting the Root Password: Windows Systems
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.5.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.3, “Server Command Options”
Section 5.5.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.5.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.10.4.7, “Starting MySQL as a Windows Service”
Section 2.10.3.1, “Starting the MySQL Server Instance Configuration Wizard”

--defaults-group-suffix

Section 4.2.7, “Command-Line Options that Affect Option-File Handling”
Section 2.21, “Environment Variables”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--delay-key-write

--delay_key_write

--delayed-insert

--delete

--delete-master-logs

--delimiter

--demangle

--des-key-file

--descending

--description

--dirname

--disable

--disable-auto-rehash

--disable-grant-options

--disable-keys

--disable-log-bin

--disable-named-commands

--disable-shared

--disable-ssl

--disconnect-slave-event-count

--dont_ignore_systab_0

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're

--fields-escaped-by

--fields-optionally-enclosed-by

--fields-terminated-by

--fields-xxx

--first-slave

--flush

--flush-logs

--flush-privileges

--flush_time

--flushlog

--force

--force-read

--foreground

--fs

G

[index top]

-G

-g

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--gci

--gci64

--gdb

H

[index top]

-H

-h

--header

--header_file

--HELP

--help

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 4.6.15, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.6.16, “mysql_tableinfo — Generate Database Metadata”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.6.17, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.18, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.1, “perror — Explain Error Codes”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 5.1.3, “Server Command Options”
Section 2.18.3, “Testing the Server”
Section 1.3.2, “The Main Features of MySQL”
Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 8.12.2, “Tuning Server Parameters”
Chapter 3, *Tutorial*
Section 4.2.6, “Using Option Files”
Section 4.2.4, “Using Options on the Command Line”

--hex

Section 17.4.14, “ndb_restore — Restore a MySQL Cluster Backup”

--hex-blob

Section 4.5.4, “mysqldump — A Database Backup Program”

--hexdump

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

--host

Section 4.2.2, “Connecting to the MySQL Server”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Options”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.12, “mysql_explain_log — Use EXPLAIN on Statements in Query Log”
Section 4.6.15, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.6.16, “mysql_tableinfo — Generate Database Metadata”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 4.2.9, “Option Defaults, Options Expecting Values, and the = Sign”
Section 5.1.3, “Server Command Options”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 1.2, “Typographical and Syntax Conventions”
Section 5.5.4, “Using Client Programs in a Multiple-Server Environment”
Section 4.2.6, “Using Option Files”
Section 4.2.4, “Using Options on the Command Line”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--init_connect

--initial

--initial-start

--innodb

--innodb-safe-binlog

--innodb-status-file

innodb-status-file

--innodb-xxx

--innodb_checksums

--innodb_file_per_table

innodb_file_per_table

--innodb_rollback_on_timeout

--insert-ignore

--install

--install-manual

--interactive

J

[index top]

-j

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--join

K

[index top]

-K

-k

--keep_files_on_create

--keepold

--key_buffer_size

--keys

--keys-used

L

[index top]

-L

-l

--language

--large-pages

--ldata

--ledir

--length

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--libmysqld-libs

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--libs

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--libs_r

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--line-numbers

Section 4.5.1.1, “mysql Options”

--lines-terminated-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”

--local

Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

--local-infile

Section 4.5.1.1, “mysql Options”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

--local-load

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

--local-service

Section 5.1.3, “Server Command Options”
Section 2.10.4.7, “Starting MySQL as a Windows Service”

--lock

Section 17.4.15, “ndb_select_all — Print Rows from an NDB Table”

--lock-all-tables

Section 4.5.4, “mysqldump — A Database Backup Program”

--lock-tables

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”

--log

Section 4.6.10.1, “MySQL Instance Manager Command Options”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”
Section 5.4.5, “Server Log Maintenance”
Section 5.4.2, “The General Query Log”

--log-bin

Section 7.3.3, “Backup Strategy Summary”
Section 16.1.2.4, “Binary Log Options and Variables”
Section 18.6, “Binary Logging of Stored Programs”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section B.5.7, “Known Issues in MySQL”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.4.5, “Server Log Maintenance”
Section 5.1.4, “Server System Variables”
Section 16.3.6, “Switching Masters During Failover”
Section 5.4.3, “The Binary Log”
Section 16.4.4, “Troubleshooting Replication”
Section 16.4.3, “Upgrading a Replication Setup”
Section 7.3.2, “Using Backups for Recovery”

--log-bin-index

Section 16.1.2.4, “Binary Log Options and Variables”
Section 5.4.3, “The Binary Log”

--log-bin-trust-function-creators

Section 16.1.2.4, “Binary Log Options and Variables”
Section 18.6, “Binary Logging of Stored Programs”

--log-bin-trust-routine-creators

Section 16.1.2.4, “Binary Log Options and Variables”
Section 18.6, “Binary Logging of Stored Programs”

--log-error

Section 13.7.6.2, “FLUSH Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--log-isam

--log-long-format

--log-queries-not-using-indexes

--log-short-format

--log-slave-updates

--log-slow-admin-statements

--log-slow-queries

--log-tc

--log-tc-size

--log-warnings

--loops

--loose

--loose-opt_name

--low-priority

--low-priority-updates

--lower-case-table-names

M

[index top]

-M

-m

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 15.3.2, “Using `memcached`”

--master-connect-retry

Section 16.4.1.14, “Replication and Master or Slave Shutdowns”
Section 8.14.6, “Replication Slave I/O Thread States”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 13.7.5.31, “SHOW SLAVE STATUS Syntax”

--master-data

Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.4, “MySQL Server Logs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--master-host

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-info-file

Section 16.1.2.3, “Replication Slave Options and Variables”
Section 16.2.2.2, “Slave Status Logs”

--master-password

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-port

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-retry-count

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 16.1.2.3, “Replication Slave Options and Variables”

--master-ssl

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-ssl*

Section 6.3.6.5, “Command Options for Secure Connections”

--master-ssl-ca

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-ssl-capath

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-ssl-cert

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-ssl-cipher

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-ssl-key

Section 16.1.2.3, “Replication Slave Options and Variables”

--master-user

Section 16.1.2.3, “Replication Slave Options and Variables”

--max

Section 4.2.8, “Using Options to Set Program Variables”

--max-binlog-dump-events

Section 16.1.2.4, “Binary Log Options and Variables”

--max-binlog-size

Section 16.1.2.3, “Replication Slave Options and Variables”

--max-record-length

Section 4.6.3.3, “myisamchk Repair Options”
Section 13.7.2.6, “REPAIR TABLE Syntax”

--max-relay-log-size

Section 16.1.2.3, “Replication Slave Options and Variables”

--max-seeks-for-key

Section 8.2.1.17, “How to Avoid Full Table Scans”
Section B.5.5, “Optimizer-Related Issues”

--max_a

Section 4.2.8, “Using Options to Set Program Variables”

--max_join_size

Using the --safe-updates Option

--maximum

Section 4.2.5, “Program Option Modifiers”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--name_file

--named-commands

--ndb

--ndb-cluster

--ndb-connectstring

--ndb-mgmd-host

--ndb-nodeid

--ndb-optimized-node-selection

--ndb-shm

--ndbcluster

net_retry_count

net_write_timeout

--nice

--no-auto-rehash

--no-autocommit

--no-beep

--no-contact

--no-create-db

--no-create-info

--no-data

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--no-debug

Section 4.4.2, “make_win_bin_dist — Package MySQL Distribution as Zip Archive”

--no-defaults

Section 4.2.7, “Command-Line Options that Affect Option-File Handling”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Options”
Section 4.4.6, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 5.1.3, “Server Command Options”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

--no-embedded

Section 4.4.2, “make_win_bin_dist — Package MySQL Distribution as Zip Archive”

--no-log

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

--no-named-commands

Section 4.5.1.1, “mysql Options”

--no-nodeid-checks

Section 17.4.2, “ndb_mgmd — The MySQL Cluster Management Server Daemon”

--no-pager

Section 4.5.1.1, “mysql Options”

--no-set-names

Section 4.5.4, “mysqldump — A Database Backup Program”

--no-symlinks

Section 4.6.3.3, “myisamchk Repair Options”

--no-tee

Section 4.5.1.1, “mysql Options”

--nodaemon

Section 17.4.2, “ndb_mgmd — The MySQL Cluster Management Server Daemon”
Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”

--nodata

Section 17.4.15, “ndb_select_all — Print Rows from an NDB Table”

--nodeid

mysqld Command Options for MySQL Cluster
Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 17.4.14, “ndb_restore — Restore a MySQL Cluster Backup”

--nodes

Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”

--noindices

Section 4.6.9, “mysqlhotcopy — A Database Backup Program”

nonguarded

Section 4.6.10, “mysqlmanager — The MySQL Instance Manager”

--nostart

Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”

--not-started

Section 17.4.19, “ndb_waiter — Wait for MySQL Cluster to Reach a Given Status”

--nowait-nodes

Section 17.4.1, “ndbd — The MySQL Cluster Data Node Daemon”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

P

[index top]

-P

-p

--pager

--parallel-recover

--parallelism

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

port

--port-open-timeout

--position

--prefix

--preview

--print

--print-defaults

--print-full-config

--print_*

--print_data

--print_log

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

-r

--raw

--read-from-remote-server

--read-only

--reconnect

--record_log_pos

--recover

--regexp

--relative

--relay-log

--relay-log-index

--relay-log-info-file

--relay-log-purge

--relay-log-space-limit

--relnotes

--remove

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--repair

--replace

--replicate-*

--replicate-*-db

--replicate-*-do-*

--replicate-*-ignore-*

--replicate-*-table

--replicate-do-db

--replicate-do-table

--replicate-ignore-db

--replicate-ignore-table

--replicate-rewrite-db

--replicate-same-server-id

--replicate-wild-do-table

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--replicate-wild-ignore-table

replication-ignore-table

--replication-rewrite-db

--report-host

--report-password

--report-port

--report-user

--resetmaster

--resetslave

--restore_data

--restore_meta

--result-file

--retries

--rhost

--rollback

--routines

--rowid

--rows

--rpm

--run-as-service

S

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

server-id

Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”
Section 16.1, “Replication Configuration”
Section 16.1.2.2, “Replication Master Options and Variables”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 16.1.1.1, “Setting the Replication Master Configuration”
Section 16.1.1.2, “Setting the Replication Slave Configuration”
Section 16.1.1.8, “Setting Up Replication with Existing Data”

--service-startup-timeout

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

--set-auto-increment

Section 4.6.3.4, “Other myisamchk Options”

--set-character-set

Section 4.6.3.3, “myisamchk Repair Options”

--set-charset

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--set-collation

Section 4.6.3.3, “myisamchk Repair Options”

--set-variable

Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 5.1.3, “Server Command Options”
Section 4.2.8, “Using Options to Set Program Variables”

set-variable

Section 4.2.8, “Using Options to Set Program Variables”

--shared-memory

Section 4.2.2, “Connecting to the MySQL Server”
Section 4.5.1.1, “mysql Options”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 5.1.3, “Server Command Options”
Section 5.5.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.10.4.4, “Starting the Server for the First Time”
Section 1.3.2, “The Main Features of MySQL”

--shared-memory-base-name

Section 4.2.2, “Connecting to the MySQL Server”
Section 4.5.1.1, “mysql Options”
Section 20.6.7.49, “mysql_options()”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.1.3, “Server Command Options”
Section 5.5.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 5.5.4, “Using Client Programs in a Multiple-Server Environment”

--short-form

Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--show-slave-auth-info

Section 16.1.2.3, “Replication Slave Options and Variables”
Section 13.7.5.30, “SHOW SLAVE HOSTS Syntax”

--show-table-type

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--show-temp-status

Section 17.4.17, “ndb_show_tables — Display List of NDB Tables”

--show-warnings

Section 4.5.1.1, “mysql Options”

--sigint-ignore

Section 4.5.1.1, “mysql Options”

--silent

Section 4.4.3, “make_win_src_distribution — Create Source Distribution for Windows”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.8.1, “perror — Explain Error Codes”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”

--single-transaction

Section 14.2.6, “Backing Up and Recovering an InnoDB Database”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 4.5.4, “mysqldump — A Database Backup Program”

--single-user

Section 17.4.19, “ndb_waiter — Wait for MySQL Cluster to Reach a Given Status”

--skip

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.2.5, “Program Option Modifiers”

--skip-add-drop-table

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-add-locks

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-auto-rehash

Section 4.5.1.1, “mysql Options”
Section 14.2.13.3, “Troubleshooting InnoDB Data Dictionary Operations”

--skip-bdb

Section 14.5.3, “BDB Startup Options”
Section 14.5.1, “Operating Systems Supported by BDB”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”

--skip-character-set-client-handshake

Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
The cp932 Character Set

--skip-column-names

Section 4.5.1.1, “mysql Options”

--skip-comments

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-compact

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-concurrent-insert

Section 5.1.3, “Server Command Options”

--skip-disable-keys

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-dump-date

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-engine_name

Section 13.7.5.13, “SHOW ENGINES Syntax”

--skip-extended-insert

Section 4.5.4, “mysqldump — A Database Backup Program”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--skip-external-locking

Section 8.11.4, “External Locking”
Section 8.14.2, “General Thread States”
Section 2.15, “Installing MySQL on NetWare”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 8.12.1, “System Factors and Startup Parameter Tuning”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”

--skip-grant-tables

Section 2.10.8.1, “Building MySQL from the Standard Source Distribution”
Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Resetting the Root Password: Generic Instructions
Section 5.1.3, “Server Command Options”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 21.2.2.5, “UDF Compiling and Installing”
Section 4.2.4, “Using Options on the Command Line”
Section 6.2.6, “When Privilege Changes Take Effect”

--skip-host-cache

Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.3, “Server Command Options”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

--skip-innodb

Section 14.2.1, “Configuring InnoDB”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section A.13, “MySQL 5.0 FAQ: Replication”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 13.7.5.13, “SHOW ENGINES Syntax”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

--skip-innodb_adaptive_hash_index

Section 14.2.2, “InnoDB Startup Options and System Variables”

--skip-innodb_checksums

Section 14.2.2, “InnoDB Startup Options and System Variables”

--skip-innodb_doublewrite

Section 14.2.2, “InnoDB Startup Options and System Variables”

--skip-kill-mysqld

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--skip-line-numbers

Section 4.5.1.1, “mysql Options”

--skip-lock-tables

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--skip-merge

Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 14.3, “The MERGE Storage Engine”

--skip-name-resolve

Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”
Section 2.20.4.1, “FreeBSD Notes”
Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 2.18.4, “Securing the Initial MySQL Accounts”
Section 5.1.3, “Server Command Options”
Section 2.10.4.8, “Testing The MySQL Installation”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”

--skip-named-commands

Section 4.5.1.1, “mysql Options”

--skip-ndbcluster

Section 17.3.2.5, “MySQL Cluster mysqld Option and Variable Reference”
MySQL Cluster System Variables
mysqld Command Options for MySQL Cluster

--skip-networking

Section B.5.2.2, “Can't connect to [local] MySQL server”
Section 8.12.6.2, “DNS Lookup Optimization and the Host Cache”
Section B.5.2.9, “MySQL server has gone away”
Resetting the Root Password: Generic Instructions
Section 5.1.3, “Server Command Options”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 5.1.4, “Server System Variables”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 16.4.3, “Upgrading a Replication Setup”

skip-networking

Section A.13, “MySQL 5.0 FAQ: Replication”
Section 16.1.1.1, “Setting the Replication Master Configuration”
Section 16.4.4, “Troubleshooting Replication”

--skip-new

Section 21.3.1, “Debugging a MySQL Server”
Section 13.7.2.5, “OPTIMIZE TABLE Syntax”
Section 5.1.4, “Server System Variables”

--skip-opt

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-pager

Section 4.5.1.1, “mysql Options”

--skip-quick

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-quote-names

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-reconnect

Section 20.6.15, “Controlling Automatic Reconnection Behavior”
Disabling mysql Auto-Reconnect
Section 4.5.1.1, “mysql Options”

--skip-routines

Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”
Section 7.4.5.3, “Dumping Stored Programs”

--skip-safemalloc

Section 2.17.5, “Compiling and Linking an Optimized mysqld Server”
Section 21.3.1.1, “Compiling MySQL for Debugging”
Section 5.1.3, “Server Command Options”

--skip-set-charset

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-show-database

Section 6.2.1, “Privileges Provided by MySQL”
Section 5.1.3, “Server Command Options”
Section 13.7.5.11, “SHOW DATABASES Syntax”
Section 1.9.5, “Supporters of MySQL”

--skip-slave-start

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 16.3.7, “Setting Up Replication to Use Secure Connections”
Section 16.1.1.8, “Setting Up Replication with Existing Data”
Section 13.4.2.7, “START SLAVE Syntax”
Section 16.4.4, “Troubleshooting Replication”
Section 16.4.3, “Upgrading a Replication Setup”

--skip-ssl

Section 6.3.6.5, “Command Options for Secure Connections”
Section 6.3.6.4, “Configuring MySQL to Use Secure Connections”

--skip-stack-trace

Section 21.3.1.4, “Debugging mysqld under gdb”
Section 5.1.3, “Server Command Options”

--skip-symbolic-links

Section 13.1.10, “CREATE TABLE Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 8.12.4.3, “Using Symbolic Links for Databases on Windows”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”

--skip-sync-bdb-logs

Section 14.5.3, “BDB Startup Options”

--skip-tee

Section 4.5.1.1, “mysql Options”

--skip-thread-priority

Section 2.20.4.5, “BSD/OS Version 3.x Notes”
Section 2.20.2.1, “OS X 10.x (Darwin)”
Section 5.1.3, “Server Command Options”

--skip-triggers

Section 2.19.2.1, “Changes Affecting Downgrades from MySQL 5.0”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 7.4.5.3, “Dumping Stored Programs”
Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-tz-utc

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-use-db

Section 4.6.13, “mysql_find_rows — Extract SQL Statements from Files”

--skip-xxx

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip_grant_tables

Section 4.2.4, “Using Options on the Command Line”

--slave-load-tmpdir

Section 16.3.1.2, “Backing Up Raw Data from a Slave”
Section 7.2, “Database Backup Methods”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section B.5.3.5, “Where MySQL Stores Temporary Files”

--slave-net-timeout

Section 16.1.2.3, “Replication Slave Options and Variables”

--slave-skip-errors

Section 16.1.2.3, “Replication Slave Options and Variables”
Section 16.4.1.21, “Slave Errors During Replication”

--slave_compressed_protocol

Section 16.1.2.3, “Replication Slave Options and Variables”

--sleep

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

--socket

Section B.5.2.2, “Can't connect to [local] MySQL server”
Section 4.2.2, “Connecting to the MySQL Server”
Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.6.10.1, “MySQL Instance Manager Command Options”
Section 4.5.1.1, “mysql Options”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 4.7.2, “mysql_config — Display Options for Compiling Clients”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.12, “mysql_explain_log — Use EXPLAIN on Statements in Query Log”
Section 4.6.15, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.6.16, “mysql_tableinfo — Generate Database Metadata”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.5.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.3, “Server Command Options”
Section 2.10.4.8, “Testing The MySQL Installation”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 5.5.4, “Using Client Programs in a Multiple-Server Environment”

--sort-index

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.4, “Other myisamchk Options”

--sort-records

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.4, “Other myisamchk Options”

--sort-recover

Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--ssl-cipher

--ssl-key

ssl-key

--ssl-verify-server-cert

--ssl-xxx

--standalone

--start-datetime

--start-position

--start_row

--statefile

--static

--stats

--status

--stop-datetime

--stop-position

--suffix

--superuser

--symbolic-links

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 15.3.2, “Using memcached”

-u

Section 4.2.2, “Connecting to the MySQL Server”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Options”
Section 4.6.12, “mysql_explain_log — Use EXPLAIN on Statements in Query Log”
Section 4.6.16, “mysql_tableinfo — Generate Database Metadata”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 17.4.7, “ndb_desc — Describe NDB Tables”
Section 17.4.17, “ndb_show_tables — Display List of NDB Tables”
Section 5.1.3, “Server Command Options”
Section 2.10.4.8, “Testing The MySQL Installation”
Section 2.18.3, “Testing the Server”
Section 6.3.1, “User Names and Passwords”
Section 15.3.2, “Using memcached”
Section 2.10.6, “Windows Postinstallation Procedures”

--unbuffered

Section 4.5.1.1, “mysql Options”

--unpack

Section 14.1.3, “MyISAM Table Storage Formats”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

--unqualified

Section 17.4.7, “ndb_desc — Describe NDB Tables”
Section 17.4.17, “ndb_show_tables — Display List of NDB Tables”

--update-state

Section 7.6.3, “How to Repair MyISAM Tables”
Section 4.6.3.2, “myisamchk Check Options”
Section 14.1, “The MyISAM Storage Engine”

--usage

Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

--use-frm

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--use-manager

Section 4.3.3, “mysql.server — MySQL Server Startup Script”

--use-mysqld_safe

Section 4.3.3, “mysql.server — MySQL Server Startup Script”

--useHexFormat

Section 17.4.15, “ndb_select_all — Print Rows from an NDB Table”

--user

Section 4.2.2, “Connecting to the MySQL Server”
Section 7.3, “Example Backup and Recovery Strategy”
Section B.5.2.18, “File Not Found and Similar Errors”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.18.1, “Initializing the Data Directory”
Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”
Section 4.2.1, “Invoking MySQL Programs”
Section 2.20.1.2, “Linux Binary Distribution Notes”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.5.10.3, “MySQL Cluster and MySQL Security Procedures”
Section 4.6.10.1, “MySQL Instance Manager Command Options”
Section 4.5.1.1, “mysql Options”
Section 4.3.3, “mysql.server — MySQL Server Startup Script”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.12, “mysql_explain_log — Use EXPLAIN on Statements in Query Log”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

V

[index top]

-V

-v

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 4.8.1, “perror — Explain Error Codes”
Section 4.8.2, “replace — A String-Replacement Utility”
Section 5.1.3, “Server Command Options”
Section 15.3.2, “Using memcached”
Section 4.2.4, “Using Options on the Command Line”

--var_name

Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 5.1.3, “Server Command Options”

--verbose

Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Options”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.4.6, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.9, “mysql_upgrade — Check Tables for MySQL Upgrade”
Section 4.6.17, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 17.4.14, “ndb_restore — Restore a MySQL Cluster Backup”
Section 4.6.3.4, “Other myisamchk Options”
Section 4.8.1, “perror — Explain Error Codes”
Section 5.1.3, “Server Command Options”
Section 2.18.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 8.12.2, “Tuning Server Parameters”
Section 4.2.6, “Using Option Files”
Section 4.2.4, “Using Options on the Command Line”

--version

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.10.1, “MySQL Instance Manager Command Options”
Section 4.5.1.1, “mysql Options”
Section 4.7.2, “mysql_config — Display Options for Compiling Clients”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.17, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.6, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 17.4.4, “ndb_config — Extract MySQL Cluster Configuration Information”
Section 17.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--vertical

W

[index top]

-W

-w

--wait

--wait-timeout

--where

--windows

--with-archive-storage-engine

--with-berkeley-db

--with-big-tables

--with-blackhole-storage-engine

--with-charset

--with-collation

--with-comment

This documentation is for an older version. If you're
This documentation is for an older version. If you're

--with-csv-storage-engine

Section 14.9, “The CSV Storage Engine”

--with-debug

Section 2.4.2.2, “Choosing a Distribution Format”
Section 2.17.5, “Compiling and Linking an Optimized mysqld Server”
Section 21.3.1.1, “Compiling MySQL for Debugging”
Section 21.3.2, “Debugging a MySQL Client”
Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”
Section 4.5.1.1, “mysql Options”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 5.1.3, “Server Command Options”
Section B.5.3.3, “What to Do If MySQL Keeps Crashing”

--with-embedded-privilege-control

Section 20.5, “libmysqld, the Embedded MySQL Server Library”

--with-embedded-server

Section 2.17.3, “MySQL Source-Configuration Options”

--with-example-storage-engine

Section 14.6, “The EXAMPLE Storage Engine”

--with-extra-charsets

Section B.5.2.17, “Can't initialize character set”
Section 2.17.3, “MySQL Source-Configuration Options”

--with-federated-storage-engine

Section 14.7, “The FEDERATED Storage Engine”

--with-libevent

Section 15.3.1, “Installing `memcached`”

--with-libwrap

Section 2.4.2.2, “Choosing a Distribution Format”

--with-low-memory

Section 2.20.4.4, “BSD/OS Version 2.x Notes”
Section 2.17.4, “Dealing with Problems Compiling MySQL”

--with-max-indexes

Section 14.1, “The MyISAM Storage Engine”

--with-mysqld-ldflags

Section 21.2.2, “Adding a New User-Defined Function”
Section 2.20.1.7, “Linux Alpha Notes”

--with-named-thread-libs

Section 2.20.5.6, “Alpha-DEC-OSF/1 Notes”

--with-named-z-libs

Section 2.4.2.2, “Choosing a Distribution Format”
Section 2.20.3.1, “Solaris 2.7/2.8 Notes”
Section 2.20.3, “Solaris Notes”

--with-ndb-sci

Section 17.3.4.1, “Configuring MySQL Cluster to use SCI Sockets”
Section 17.3.3.11, “SCI Transport Connections in MySQL Cluster”

--with-ndb-shm

Section 17.3.3.10, “MySQL Cluster Shared-Memory Connections”

--with-ndbcluster

Section 17.2.1.3, “Building MySQL Cluster from Source on Linux”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”

--with-pstack

Section 5.1.3, “Server Command Options”

--with-tcp-port

Section 2.17.2, “Installing MySQL Using a Development Source Tree”
Section 2.17.3, “MySQL Source-Configuration Options”

--with-unix-socket-path

Section B.5.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 2.17.2, “Installing MySQL Using a Development Source Tree”

with-unix-socket-path

Section 2.17.3, “MySQL Source-Configuration Options”

--with-vio

Section 6.3.6.2, “Building MySQL with Support for Secure Connections”

--with-zlib-dir

Section 2.17.3, “MySQL Source-Configuration Options”

--without-query-cache

Section 8.10.3, “The MySQL Query Cache”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're

Privileges Index

A | C | D | E | F | G | I | L | P | R | S | T | U

A

[index top]

ALL

ALL PRIVILEGES

ALTER

ALTER ROUTINE

C

[index top]

CREATE

CREATE ROUTINE

CREATE TEMPORARY TABLES

CREATE USER

CREATE VIEW

D

[index top]

DELETE

DROP

This documentation is for an older version. If you're
This documentation is for an older version. If you're

E

[index top]

EXECUTE

F

[index top]

FILE

G

[index top]

GRANT OPTION

I

[index top]

INDEX

INSERT

L

[index top]

LOCK TABLES

This documentation is for an older version. If you're
This documentation is for an older version. If you're

P

[index top]

PROCESS

R

[index top]

REFERENCES

RELOAD

REPLICATION CLIENT

REPLICATION SLAVE

S

[index top]

SELECT

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SHOW DATABASES

SHOW VIEW

SHUTDOWN

SUPER

T

[index top]

TRIGGER

U

[index top]

UPDATE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 13.2.5, “INSERT Syntax”
Section 17.5.10.2, “MySQL Cluster and MySQL Privileges”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.7.1.4, “RENAME USER Syntax”
Section 13.7.1.5, “REVOKE Syntax”
Section 19.5, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”
Section 19.14, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
Section 14.3, “The MERGE Storage Engine”
Section 18.3.1, “Trigger Syntax and Examples”
Section 13.2.10, “UPDATE Syntax”

USAGE

Section 13.7.1.3, “GRANT Syntax”
Section 6.2.1, “Privileges Provided by MySQL”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SQL Modes Index

A

[index top]

ALLOW_INVALID_DATES

ANSI

ANSI_QUOTES

D

[index top]

DB2

E

[index top]

ERROR_FOR_DIVISION_BY_ZERO

H

[index top]

HIGH_NOT_PRECEDENCE

I

[index top]

IGNORE_SPACE

M

[index top]

MAXDB

MSSQL

MYSQL323

MYSQL40

N

[index top]

NO_AUTO_CREATE_USER

NO_AUTO_VALUE_ON_ZERO

NO_BACKSLASH_ESCAPES

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 12.17.3, “Expression Handling”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section A.3, “MySQL 5.0 FAQ: Server SQL Mode”
Section 5.1.7, “Server SQL Modes”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Statement/Syntax Index

A | B | C | D | E | F | G | H | I | K | L | O | P | R | S | T | U | W | X

A

[index top]

ALTER DATABASE

ALTER FUNCTION

ALTER PROCEDURE

ALTER SCHEMA

ALTER TABLE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

ALTER TABLE ... ENGINE = MEMORY

ALTER TABLE ... RENAME

ALTER VIEW

ANALYZE TABLE

B

[index top]

BACKUP TABLE

BEGIN

This documentation is for an older version. If you're
This documentation is for an older version. If you're

BEGIN ... END

BEGIN WORK

C

[index top]

CACHE INDEX

CALL

CASE

CHANGE MASTER TO

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 16.3.6, “Switching Masters During Failover”

CHECK TABLE

Section 14.2.6, “Backing Up and Recovering an InnoDB Database”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 13.7.2.3, “CHECK TABLE Syntax”
Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Section 14.1.4.1, “Corrupted MyISAM Tables”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.12, “CREATE VIEW Syntax”
Section 8.11.4, “External Locking”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 14.2.13.2, “InnoDB General Troubleshooting”
Section 2.15, “Installing MySQL on NetWare”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section A.6, “MySQL 5.0 FAQ: Views”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section B.5.2.9, “MySQL server has gone away”
Section 20.6.7.69, “mysql_store_result()”
Section 4.4.9, “`mysql_upgrade` — Check Tables for MySQL Upgrade”
Section 20.6.7.71, “mysql_use_result()”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 14.1.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section C.2, “Restrictions on Server-Side Cursors”
Section C.1, “Restrictions on Stored Programs”
Section C.4, “Restrictions on Views”
Section 5.1.3, “Server Command Options”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 14.8, “The ARCHIVE Storage Engine”
Section 14.3, “The MERGE Storage Engine”
Section 5.4.4, “The Slow Query Log”

CHECK TABLE ... EXTENDED

Section 13.7.2.3, “CHECK TABLE Syntax”

CHECK TABLE ... FOR UPGRADE

Section 2.19.3, “Checking Whether Tables or Indexes Must Be Rebuilt”
Section 13.7.2.6, “REPAIR TABLE Syntax”

CHECKSUM TABLE

Section 13.7.2.4, “CHECKSUM TABLE Syntax”
Section 13.1.10, “CREATE TABLE Syntax”
Section 16.4.1.3, “Replication and CHECKSUM TABLE”

COMMIT

Section 18.6, “Binary Logging of Stored Programs”
Section 8.6.4, “Bulk Data Loading for InnoDB Tables”
Section 14.5.4, “Characteristics of BDB Tables”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 14.2.3.1, “How to Use Transactions in InnoDB with Different APIs”
Section 14.2.8.7, “Implicit Transaction Commit and Rollback”
Section 14.2.12, “InnoDB Error Handling”
Section 14.2.8, “InnoDB Transaction Model and Locking”
Section B.5.7, “Known Issues in MySQL”
Section 14.2.14, “Limits on InnoDB Tables”
Section 13.3, “MySQL Transactional and Locking Statements”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 16.4.1.26, “Replication and Transactions”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT, and Syntax”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 14, *Storage Engines*
Section 14.5, “The BDB (BerkeleyDB) Storage Engine”
Section 5.4.3, “The Binary Log”
Section 1.8.2.3, “Transactions and Atomic Operations”
Section 18.3.1, “Trigger Syntax and Examples”

CREATE DATABASE

Section 7.1, “Backup and Recovery Types”
Section 16.1.2.4, “Binary Log Options and Variables”
Section 20.6.6, “C API Function Overview”
Section 10.1.5, “Configuring the Character Set and Collation for Applications”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 13.1.6, “CREATE DATABASE Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 10.1.3.2, “Database Character Set and Collation”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 9.2.2, “Identifier Case Sensitivity”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 20.6.7.8, “mysql_create_db()”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 10.1.3.1, “Server Character Set and Collation”
Section B.3, “Server Error Codes and Messages”
Section 13.7.5.6, “SHOW CREATE DATABASE Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CREATE DATABASE dbx

Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”

CREATE FUNCTION

Section 21.2, “Adding New Functions to MySQL”
Section 13.1.2, “ALTER FUNCTION Syntax”
Section 18.6, “Binary Logging of Stored Programs”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 1.9.1, “Contributors to MySQL”
Section 13.1.7, “CREATE FUNCTION Syntax”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.7.3.2, “DROP FUNCTION Syntax”
Section 9.2.3, “Function Name Parsing and Resolution”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section C.1, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 18.2.1, “Stored Routine Syntax”
Section 21.2.2.1, “UDF Calling Sequences for Simple Functions”
Section 21.2.2.5, “UDF Compiling and Installing”
Section 21.2.2.6, “UDF Security Precautions”
Section 2.19.1, “Upgrading MySQL”

CREATE INDEX

Section 8.3.4, “Column Indexes”
Section 13.1.8, “CREATE INDEX Syntax”
Section 13.1.10, “CREATE TABLE Syntax”
Section 11.5.3.6, “Creating Spatial Indexes”
Section 12.9, “Full-Text Search Functions”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section 8.7, “Optimizing for MEMORY Tables”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 13.7.5.18, “SHOW INDEX Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.4.4, “The Slow Query Log”
Section 17.1.5.6, “Unsupported or Missing Features in MySQL Cluster”

CREATE OR REPLACE VIEW

Section 13.1.5, “ALTER VIEW Syntax”
Section 13.1.12, “CREATE VIEW Syntax”
Section C.4, “Restrictions on Views”

CREATE PROCEDURE

Section 13.1.3, “ALTER PROCEDURE Syntax”
Section 18.6, “Binary Logging of Stored Programs”
Section 13.2.1, “CALL Syntax”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 18.2.1, “Stored Routine Syntax”

CREATE SCHEMA

Section 13.1.6, “CREATE DATABASE Syntax”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”

CREATE TABLE

Section 13.1.4, “ALTER TABLE Syntax”
Section 14.2.3.3, “AUTO_INCREMENT Handling in InnoDB”
Section 7.1, “Backup and Recovery Types”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 10.1.3.4, “Column Character Set and Collation”
Section 8.3.4, “Column Indexes”
Section 13.1.8, “CREATE INDEX Syntax”
Section 13.1.10.2, “CREATE TABLE ... SELECT Syntax”
Section 13.1.10, “CREATE TABLE Syntax”
Section 3.3.2, “Creating a Table”
Section 14.2.3, “Creating and Using InnoDB Tables”
Section 11.5.3.2, “Creating Spatial Columns”
Section 11.5.3.6, “Creating Spatial Indexes”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 7.2, “Database Backup Methods”
Section 10.1.3.2, “Database Character Set and Collation”
Section 17.3.3.5, “Defining MySQL Cluster Data Nodes”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 1.8.3.2, “FOREIGN KEY Constraints”
Section 12.9, “Full-Text Search Functions”
Section 3.4, “Getting Information About Databases and Tables”
Section 13.8.3, “HELP Syntax”
Section 9.2.2, “Identifier Case Sensitivity”
Section 17.2.2, “Initial Configuration of MySQL Cluster”
Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”
Section 14.2.3.5, “InnoDB and MySQL Replication”
Section 14.2.13.2, “InnoDB General Troubleshooting”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section 17.1.5.9, “Limitations Relating to Multiple MySQL Cluster Nodes”
Section 14.2.14, “Limits on InnoDB Tables”
Section C.7.3, “Limits on Table Size”
Section 17.1.5.3, “Limits Relating to Transaction Handling in MySQL Cluster”
Section 3.3.3, “Loading Data into a Table”
Section 14.1.3, “MyISAM Table Storage Formats”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
Section 17.3.3.1, “MySQL Cluster Configuration: Basic Example”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.1.1, “mysql Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.4.1, “Optimizing Data Size”
Section 8.6.6, “Optimizing InnoDB DDL Operations”
Section 6.2.1, “Privileges Provided by MySQL”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 16.4.1.1, “Replication and AUTO_INCREMENT”
Section 16.4.1.2, “Replication and Character Sets”
Section 16.4.1.6, “Replication and DIRECTORY Table Options”
Section 16.4.1.9, “Replication and System Functions”
Section 8.14.8, “Replication Slave Connection Thread States”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 13.7.5.9, “SHOW CREATE TABLE Syntax”
Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 13.7.5.12, “SHOW ENGINE Syntax”
Section 13.7.5.18, “SHOW INDEX Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 13.7.5.33, “SHOW TABLE STATUS Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section 13.1.10.4, “Silent Column Specification Changes”
Section B.1, “Sources of Error Information”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 14, *Storage Engines*
Section 11.1.3, “String Type Overview”
Section 10.1.3.3, “Table Character Set and Collation”
Section 11.4.4, “The ENUM Type”
Section 14.4, “The MEMORY (HEAP) Storage Engine”
Section 14.1, “The MyISAM Storage Engine”
Section 13.2.9.1, “The Subquery as Scalar Operand”
Section 14.2.13.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 17.1.5.6, “Unsupported or Missing Features in MySQL Cluster”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”
Section 3.3.4.9, “Using More Than one Table”
Section 7.4, “Using mysqldump for Backups”
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section C.7.6, “Windows Platform Limitations”

CREATE TABLE ... LIKE

Section 13.1.10.1, “CREATE TABLE ... LIKE Syntax”
Section 16.4.1.1, “Replication and AUTO_INCREMENT”
Section 14.3, “The MERGE Storage Engine”

CREATE TABLE ... SELECT

Section 18.6, “Binary Logging of Stored Programs”
Section 12.10, “Cast Functions and Operators”
Section 14.2.8.4, “Consistent Nonlocking Reads”
Section 13.1.10.2, “CREATE TABLE ... SELECT Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 16.4.1.4, “Replication of CREATE TABLE ... SELECT Statements”
Section 1.8.2.1, “SELECT INTO TABLE”

CREATE TABLE ... SELECT ...

Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

CREATE TEMPORARY TABLE

Section 13.7.1.3, “GRANT Syntax”
Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

CREATE TRIGGER

CREATE USER

CREATE VIEW

D

[index top]

DEALLOCATE PREPARE

DECLARE

DECLARE ... CONDITION

DECLARE ... HANDLER

DELETE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 13.1.11, “CREATE TRIGGER Syntax”
Section 13.1.12, “CREATE VIEW Syntax”
Section 13.2.2, “DELETE Syntax”
Section B.5.4.6, “Deleting Rows from Related Tables”
Section 14.2.6.2, “Forcing InnoDB Recovery”
Chapter 12, *Functions and Operators*
Section 8.14.2, “General Thread States”
Section 13.7.1.3, “GRANT Syntax”
Section 6.2.2, “Grant Tables”
Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.13, “Information Functions”
Chapter 19, *INFORMATION_SCHEMA Tables*
Section 14.2.3.5, “InnoDB and MySQL Replication”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
Section 13.2.8.2, “JOIN Syntax”
Section 9.3, “Keywords and Reserved Words”
Section 13.7.6.3, “KILL Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 14.7.3, “Limitations of the FEDERATED Storage Engine”
Section 17.1.5.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 14.3.2, “MERGE Table Problems”
Section 17.5.10.3, “MySQL Cluster and MySQL Security Procedures”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.1.1, “mysql Options”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.13, “mysql_stmt_field_count()”
Section 20.6.11.17, “mysql_stmt_num_rows()”
Section 17.4.6, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 8.2.2, “Optimizing DML Statements”
Section 8.2.1, “Optimizing SELECT Statements”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.10, “Replication and LIMIT”
Section 16.4.1.15, “Replication and MEMORY Tables”
Section 16.4.1.19, “Replication and the Query Optimizer”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section C.4, “Restrictions on Views”
Section 13.7.1.5, “REVOKE Syntax”
Section 13.2.9.11, “Rewriting Subqueries as Joins”
Section 3.3.4.1, “Selecting All Data”
Section 5.1.3, “Server Command Options”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Syntax”
Section 13.2.9.9, “Subquery Errors”
Section 13.2.9, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.4.3, “The Binary Log”
Section 14.10, “The BLACKHOLE Storage Engine”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.3.2, “The Main Features of MySQL”
Section 14.4, “The MEMORY (HEAP) Storage Engine”
Section 14.3, “The MERGE Storage Engine”
Section 6.2, “The MySQL Access Privilege System”
Section 18.3.1, “Trigger Syntax and Examples”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 13.1.21, “TRUNCATE TABLE Syntax”
Section 18.4.3, “Updatable and Insertable Views”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”
Using the --safe-updates Option
Section 20.6.14.2, “What Results You Can Get from a Query”
Section 6.2.6, “When Privilege Changes Take Effect”
Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

DELETE FROM ... WHERE ...

Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

DELETE FROM a.t

Section 16.1.2.3, “Replication Slave Options and Variables”

DESCRIBE

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 14.5.4, “Characteristics of BDB Tables”
Section 13.1.10, “CREATE TABLE Syntax”
Section 3.3.2, “Creating a Table”
Section 13.8.1, “DESCRIBE Syntax”
Section 13.8.2, “EXPLAIN Syntax”
Section 19.18, “Extensions to SHOW Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 8.2.1.15, “LIMIT Query Optimization”
Section 20.6.11.27, “mysql_stmt_store_result()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 13.1.10.4, “Silent Column Specification Changes”
Section 3.6.6, “Using Foreign Keys”
Section 10.1.11, “UTF-8 for Metadata”

DO

Section 18.6, “Binary Logging of Stored Programs”
Section 13.2.3, “DO Syntax”
Section 12.15, “Miscellaneous Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 13.2.9, “Subquery Syntax”

DROP DATABASE

Section 16.1.2.4, “Binary Log Options and Variables”
Section 18.6, “Binary Logging of Stored Programs”
Section 20.6.6, “C API Function Overview”
Section 13.1.13, “DROP DATABASE Syntax”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 8.10.3.1, “How the Query Cache Operates”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 20.6.7.11, “mysql_drop_db()”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section C.7.6, “Windows Platform Limitations”

DROP DATABASE IF EXISTS

Section 16.4.1.5, “Replication of DROP ... IF EXISTS Statements”

DROP FUNCTION

Section 21.2, “Adding New Functions to MySQL”
Section 13.1.2, “ALTER FUNCTION Syntax”
Section 18.6, “Binary Logging of Stored Programs”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 1.9.1, “Contributors to MySQL”
Section 13.7.3.1, “CREATE FUNCTION Syntax for User-defined Functions”
Section 13.1.14, “DROP FUNCTION Syntax”
Section 13.7.3.2, “DROP FUNCTION Syntax”
Section 13.1.16, “DROP PROCEDURE and DROP FUNCTION Syntax”
Section 9.2.3, “Function Name Parsing and Resolution”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 18.2.1, “Stored Routine Syntax”
Section 21.2.2.5, “UDF Compiling and Installing”
Section 21.2.2.6, “UDF Security Precautions”
Section 2.19.1, “Upgrading MySQL”

DROP INDEX

Section 13.1.4, “ALTER TABLE Syntax”
Section 11.5.3.6, “Creating Spatial Indexes”
Section 13.1.15, “DROP INDEX Syntax”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 5.1.3, “Server Command Options”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.4.4, “The Slow Query Log”

DROP PROCEDURE

Section 13.1.3, “ALTER PROCEDURE Syntax”
Section 18.6, “Binary Logging of Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 18.2.1, “Stored Routine Syntax”

DROP SCHEMA

Section 13.1.13, “DROP DATABASE Syntax”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”

DROP TABLE

Section 13.1.4, “ALTER TABLE Syntax”
Section 14.2.8.4, “Consistent Nonlocking Reads”
Section 13.1.11, “CREATE TRIGGER Syntax”
Section 13.1.17, “DROP TABLE Syntax”
Section 8.10.3.1, “How the Query Cache Operates”
Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section B.5.7, “Known Issues in MySQL”
Section 14.7.3, “Limitations of the FEDERATED Storage Engine”
Section 17.1.5.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 14.3.2, “MERGE Table Problems”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.1.1, “mysql Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.4.8, “`ndb_drop_index` — Drop Index from an NDB Table”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

DROP TABLE IF EXISTS

DROP TRIGGER

DROP USER

DROP VIEW

DROP VIEW IF EXISTS

E

[index top]

EXECUTE

EXPLAIN

This documentation is for an older version. If you're
This documentation is for an older version. If you're

EXPLAIN EXTENDED

EXPLAIN SELECT

EXPLAIN SELECT ... ORDER BY

EXPLAIN tbl_name

F

[index top]

FETCH

FETCH ... INTO var_list

FLUSH

FLUSH DES_KEY_FILE

FLUSH HOSTS

FLUSH LOGS

FLUSH MASTER

FLUSH PRIVILEGES

This documentation is for an older version. If you're
This documentation is for an older version. If you're

FLUSH QUERY CACHE

FLUSH SLAVE

FLUSH STATUS

FLUSH TABLE

FLUSH TABLES

FLUSH TABLES WITH READ LOCK

FLUSH USER_RESOURCES

G

[index top]

GRANT

This documentation is for an older version. If you're
This documentation is for an older version. If you're

GRANT ALL

GRANT USAGE

H

[index top]

HANDLER

HANDLER ... CLOSE

HANDLER ... OPEN

HELP

I

[index top]

IF

INSERT

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 19, *INFORMATION_SCHEMA Tables*
Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.5.1, “INSERT ... SELECT Syntax”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.2.5, “INSERT Syntax”
Section 8.11.1, “Internal Locking Methods”
Section 14.7.3, “Limitations of the FEDERATED Storage Engine”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 3.3.3, “Loading Data into a Table”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 14.3.2, “MERGE Table Problems”
Section 12.15, “Miscellaneous Functions”
Section A.1, “MySQL 5.0 FAQ: General”
Section A.5, “MySQL 5.0 FAQ: Triggers”
Section A.6, “MySQL 5.0 FAQ: Views”
Section 17.2.4, “MySQL Cluster Example with Tables and Data”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.1.1, “mysql Options”
Section B.5.2.9, “MySQL server has gone away”
Section 20.6.7.1, “mysql_affected_rows()”
Section 20.6.7.37, “mysql_insert_id()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.13, “mysql_stmt_field_count()”
Section 20.6.11.16, “mysql_stmt_insert_id()”
Section 20.6.11.17, “mysql_stmt_num_rows()”
Section 20.6.11.20, “mysql_stmt_prepare()”
Section 20.6.7.69, “mysql_store_result()”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 13.7.2.5, “OPTIMIZE TABLE Syntax”
Section 8.2.2, “Optimizing DML Statements”
Section 8.5.1, “Optimizing MyISAM Queries”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 17.5.5, “Performing a Rolling Restart of a MySQL Cluster”
Section 11.5.3.3, “Populating Spatial Columns”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.2.7, “REPLACE Syntax”
Section 16.4.1.1, “Replication and AUTO_INCREMENT”
Section 16.4.1.22, “Replication and Server SQL Mode”
Section 16.4.1.9, “Replication and System Functions”
Section 16.4.1.29, “Replication and Variables”
Section 16.1.2.2, “Replication Master Options and Variables”
Section 16.2.3.3, “Replication Rule Application”
Section C.1, “Restrictions on Stored Programs”
Section 5.1.3, “Server Command Options”
Section B.3, “Server Error Codes and Messages”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.7.5.25, “SHOW PROCEDURE CODE Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section 16.4.1.21, “Slave Errors During Replication”
Section 8.2.2.1, “Speed of INSERT Statements”
Section 13.2.9, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 10.1.7.6, “The _bin and binary Collations”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.4.3, “The Binary Log”
Section 14.10, “The BLACKHOLE Storage Engine”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.3.2, “The Main Features of MySQL”
Section 14.3, “The MERGE Storage Engine”
Section 14.1, “The MyISAM Storage Engine”
Section 6.2, “The MySQL Access Privilege System”
Section 8.10.3, “The MySQL Query Cache”
Section 5.1.10, “The Server Shutdown Process”
Section 18.3.1, “Trigger Syntax and Examples”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 18.4.3, “Updatable and Insertable Views”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”
Section 18.3, “Using Triggers”
Section 20.6.14.2, “What Results You Can Get from a Query”
Section 6.2.6, “When Privilege Changes Take Effect”
Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section C.7.6, “Windows Platform Limitations”

INSERT ... ON DUPLICATE KEY UPDATE

Section 12.13, “Information Functions”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.2.5, “INSERT Syntax”
Section 14.7.3, “Limitations of the FEDERATED Storage Engine”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 14.3.2, “MERGE Table Problems”
Section 12.15, “Miscellaneous Functions”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

INSERT ... SELECT

INSERT ... SET

INSERT ... VALUES

INSERT DELAYED

INSERT IGNORE

INSERT INTO ... SELECT

INSERT INTO ... SELECT ...

INSERT INTO ... SELECT FROM memory_table

ITERATE

K

[index top]

KILL

KILL CONNECTION

This documentation is for an older version. If you're
This documentation is for an older version. If you're

KILL QUERY

L

[index top]

LEAVE

LOAD DATA

LOAD DATA FROM MASTER

LOAD DATA INFILE

LOAD DATA INFILE ...

LOAD DATA LOCAL

This documentation is for an older version. If you're
This documentation is for an older version. If you're

LOAD DATA LOCAL INFILE

LOAD INDEX INTO CACHE

LOAD INDEX INTO CACHE ... IGNORE LEAVES

LOAD TABLE FROM MASTER

LOCK TABLE

LOCK TABLES

LOOP

O

[index top]

OPTIMIZE TABLE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.1, “myisamchk General Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 13.7.2.5, “OPTIMIZE TABLE Syntax”
Section 8.5.1, “Optimizing MyISAM Queries”
Section 8.2.4, “Other Optimization Tips”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.4.1.8, “Replication and FLUSH”
Section C.1, “Restrictions on Stored Programs”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 8.2.2.2, “Speed of UPDATE Statements”
Section 14.1.3.1, “Static (Fixed-Length) Table Characteristics”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.1.10, “The Server Shutdown Process”
Section 5.4.4, “The Slow Query Log”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”

P

[index top]

PREPARE

Section 13.2.1, “CALL Syntax”
Section 13.5.3, “DEALLOCATE PREPARE Syntax”
Section 13.5.2, “EXECUTE Syntax”
Section 9.2.2, “Identifier Case Sensitivity”
Section 13.5.1, “PREPARE Syntax”
Section C.1, “Restrictions on Stored Programs”
Section C.4, “Restrictions on Views”
Section 5.1.6, “Server Status Variables”
Section 13.5, “SQL Syntax for Prepared Statements”

PURGE BINARY LOGS

Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.1.3, “GRANT Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 5.4.5, “Server Log Maintenance”
Section 5.1.4, “Server System Variables”
Section 5.4.3, “The Binary Log”

R

[index top]

RELEASE SAVEPOINT

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT, and Syntax”

RENAME TABLE

Section 13.1.4, “ALTER TABLE Syntax”
Section 13.2.2, “DELETE Syntax”
Section 8.14.2, “General Thread States”
Section 9.2.2, “Identifier Case Sensitivity”
Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 13.1.20, “RENAME TABLE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”

RENAME USER

Section 13.7.1.3, “GRANT Syntax”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.7.1.4, “RENAME USER Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”

REPAIR TABLE

Section 13.1.4, “ALTER TABLE Syntax”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 14.1.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 8.11.4, “External Locking”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.2, “General Thread States”
Section 15.2.3, “Handling MySQL Recovery with ZFS”
Section B.5.3.4, “How MySQL Handles a Full Disk”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 2.15, “Installing MySQL on NetWare”
Section 13.7.6.3, “KILL Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 14.3.2, “MERGE Table Problems”
Section 14.1.1, “MyISAM Startup Options”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 6.2.1, “Privileges Provided by MySQL”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 14.1.4.2, “Problems from Tables Not Being Closed Properly”
Section B.5.6.1, “Problems with ALTER TABLE”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 13.7.2.6, “REPAIR TABLE Syntax”
Section 16.4.1.8, “Replication and FLUSH”
Section 16.4.1.13, “Replication and REPAIR TABLE”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 8.5.3, “Speed of REPAIR TABLE Statements”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.1.10, “The Server Shutdown Process”
Section 5.4.4, “The Slow Query Log”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”

REPEAT

Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 18.1, “Defining Stored Programs”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.5.6, “REPEAT Syntax”
Section 13.6.2, “Statement Label Syntax”

REPLACE

Section 13.1.10.2, “CREATE TABLE ... SELECT Syntax”
Section 13.1.11, “CREATE TRIGGER Syntax”
Section 11.6, “Data Type Default Values”
Section 12.13, “Information Functions”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.2.5, “INSERT Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 14.3.2, “MERGE Table Problems”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section B.5.2.9, “MySQL server has gone away”
Section 20.6.7.1, “mysql_affected_rows()”
Section 8.1, “Optimization Overview”
Section 17.1.5.10, “Previous MySQL Cluster Issues Resolved in MySQL 5.0”
Section 13.2.7, “REPLACE Syntax”
Section 5.1.3, “Server Command Options”
Section 13.2.9, “Subquery Syntax”
Section 14.8, “The ARCHIVE Storage Engine”
Section 1.3.2, “The Main Features of MySQL”

REPLACE ... SELECT

Section B.5.7, “Known Issues in MySQL”

RESET

Section 13.7.6.2, “FLUSH Syntax”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 13.7.6.5, “RESET Syntax”

RESET MASTER

Section 13.7.6.2, “FLUSH Syntax”
Section 20.6.7.55, “mysql_refresh()”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 16.3.6, “Switching Masters During Failover”
Section 5.4.3, “The Binary Log”

RESET SLAVE

Section 13.7.6.2, “FLUSH Syntax”
Section 20.6.7.55, “mysql_refresh()”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 13.4.2.5, “RESET SLAVE Syntax”

RESTORE TABLE

Section 13.7.2.2, “BACKUP TABLE Syntax”
Section 13.7.2.7, “RESTORE TABLE Syntax”
Section 8.12.4.2, “Using Symbolic Links for MyISAM Tables on Unix”

RETURN

Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.5, “LOOP Syntax”
Section C.1, “Restrictions on Stored Programs”
Section 13.6.5.7, “RETURN Syntax”

REVOKE

Section 6.2.5, “Access Control, Stage 2: Request Verification”
Section 13.7.1.2, “DROP USER Syntax”
Section 13.7.6.2, “FLUSH Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 6.2.2, “Grant Tables”
Section 8.12.5.1, “How MySQL Uses Memory”
Section A.13, “MySQL 5.0 FAQ: Replication”
Section 17.5.10.2, “MySQL Cluster and MySQL Privileges”
Section 1.8.2, “MySQL Differences from Standard SQL”
Section 6.2.1, “Privileges Provided by MySQL”
Section 2.18.1.1, “Problems Running mysql_install_db”
Section 16.4.1.18, “Replication and User Privileges”
Section 16.4.1.17, “Replication of the mysql System Database”
Section 13.7.1.5, “REVOKE Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

REVOKE ALL PRIVILEGES

ROLLBACK

ROLLBACK TO SAVEPOINT

ROLLBACK to SAVEPOINT

S

[index top]

SAVEPOINT

SELECT

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Chapter 12, *Functions and Operators*
Section 8.14.2, “General Thread States”
Section 13.7.1.3, “GRANT Syntax”
Section 12.16.1, “GROUP BY (Aggregate) Function Descriptions”
Section 13.2.4, “HANDLER Syntax”
Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
Section 8.10.3.1, “How the Query Cache Operates”
Section 14.2.8.9, “How to Cope with Deadlocks”
Section 1.7, “How to Report Bugs or Problems”
Section 14.2.3.1, “How to Use Transactions in InnoDB with Different APIs”
Section 9.2.1, “Identifier Qualifiers”
Section 8.9.2, “Index Hints”
Section 12.13, “Information Functions”
Chapter 19, *INFORMATION_SCHEMA Tables*
Section 2.18.1, “Initializing the Data Directory”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 13.2.5.1, “INSERT ... SELECT Syntax”
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 13.2.5, “INSERT Syntax”
Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”
Section 8.11.1, “Internal Locking Methods”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 13.2.8.2, “JOIN Syntax”
Section 9.3, “Keywords and Reserved Words”
Section 13.7.6.3, “KILL Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 14.7.3, “Limitations of the FEDERATED Storage Engine”
Section 17.1.5.3, “Limits Relating to Transaction Handling in MySQL Cluster”
Section 13.4.2.2, “LOAD DATA FROM MASTER Syntax”
Section 13.4.2.3, “LOAD TABLE tbl_name FROM MASTER Syntax”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 14.3.2, “MERGE Table Problems”
Section 8.3.5, “Multiple-Column Indexes”
Section 7.6.4, “MyISAM Table Optimization”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.13, “MySQL 5.0 FAQ: Replication”
Section A.4, “MySQL 5.0 FAQ: Stored Procedures and Functions”
Section 17.2.4, “MySQL Cluster Example with Tables and Data”
Section 1.8.2, “MySQL Differences from Standard SQL”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.1.1, “mysql Options”
Section 20.6.7.1, “mysql_affected_rows()”
Section 4.6.12, “`mysql_explain_log` — Use EXPLAIN on Statements in Query Log”
Section 20.6.7.17, “mysql_fetch_field()”
Section 20.6.7.22, “mysql_field_count()”
Section 20.6.7.37, “mysql_insert_id()”
Section 20.6.7.47, “mysql_num_fields()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.11, “mysql_stmt_fetch()”
Section 20.6.11.17, “mysql_stmt_num_rows()”
Section 20.6.11.27, “mysql_stmt_store_result()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 17.4.15, “`ndb_select_all` — Print Rows from an NDB Table”
Section 8.3, “Optimization and Indexes”
Section B.5.5, “Optimizer-Related Issues”
Section 8.5.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.2.1.14, “Optimizing Subqueries with EXISTS Strategy”
Section 4.6.3.4, “Other myisamchk Options”
Section 6.2.1, “Privileges Provided by MySQL”
Section B.5.4.2, “Problems Using DATE Columns”
Section 8.10.3.2, “Query Cache SELECT Options”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 16.2, “Replication Implementation”
Section 16.1.2.2, “Replication Master Options and Variables”
Section C.1, “Restrictions on Stored Programs”
Section C.4, “Restrictions on Views”
Section 3.3.4, “Retrieving Information from a Table”
Section 3.6.7, “Searching on Two Keys”
Section 2.18.4, “Securing the Initial MySQL Accounts”
Section 14.2.8.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”
Section 13.2.8.1, “SELECT ... INTO Syntax”
Section 13.2.8, “SELECT Syntax”
Section 3.3.4.1, “Selecting All Data”
Section 3.3.4.2, “Selecting Particular Rows”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.7.4, “SET Syntax”
Section 13.3.6, “SET TRANSACTION Syntax”
Section 2.20.5.7, “SGI Irix Notes”
Section 13.7.5.2, “SHOW BINLOG EVENTS Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 13.7.5.8, “SHOW CREATE PROCEDURE Syntax”
Section 13.7.5.10, “SHOW CREATE VIEW Syntax”
Section 13.7.5.14, “SHOW ERRORS Syntax”
Section 13.7.5.25, “SHOW PROCEDURE CODE Syntax”
Section 13.7.5.27, “SHOW PROCESSLIST Syntax”
Section 13.7.5, “SHOW Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section B.5.4.7, “Solving Problems with No Matching Rows”
Section 8.2.1.1, “Speed of SELECT Statements”
Section 8.2.2.2, “Speed of UPDATE Statements”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 18.2.1, “Stored Routine Syntax”
Section 9.1.1, “String Literals”
Section 13.2.9.8, “Subqueries in the FROM Clause”
Section 13.2.9.6, “Subqueries with EXISTS or NOT EXISTS”
Section 13.2.9.9, “Subquery Errors”
Section 13.2.9, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.4.3, “The Binary Log”
Section 11.4.4, “The ENUM Type”
Section 19.4, “The INFORMATION_SCHEMA COLUMNS Table”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.3.2, “The Main Features of MySQL”
Section 14.3, “The MERGE Storage Engine”
Section 6.2, “The MySQL Access Privilege System”
Section 8.10.3, “The MySQL Query Cache”
The Range Access Method for Single-Part Indexes
Section 13.2.9.1, “The Subquery as Scalar Operand”
Section 18.3.1, “Trigger Syntax and Examples”
Section 1.2, “Typographical and Syntax Conventions”
Section 13.2.8.3, “UNION Syntax”
Section 13.2.10, “UPDATE Syntax”
Section 9.4, “User-Defined Variables”
Section 8.4.2.4, “Using PROCEDURE ANALYSE”
Section 21.3.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 11.5.3.7, “Using Spatial Indexes”
Section 5.1.5, “Using System Variables”
Using the --safe-updates Option
Section 10.1.11, “UTF-8 for Metadata”
Section 18.4.1, “View Syntax”
Section B.5.3.5, “Where MySQL Stores Temporary Files”
Section C.7.6, “Windows Platform Limitations”

SELECT *

Section 11.4.3, “The BLOB and TEXT Types”

SELECT * INTO OUTFILE 'file_name' FROM tbl_name

Section 7.2, “Database Backup Methods”

SELECT ... FOR UPDATE

Section 14.2.8.9, “How to Cope with Deadlocks”
Section 14.2.8.1, “InnoDB Lock Modes”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 14.2.8.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”

SELECT ... FROM

Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... FROM ... FOR UPDATE

Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... FROM ... LOCK IN SHARE MODE

Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... INTO

Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 16.4.1.9, “Replication and System Functions”
Section 13.2.8.1, “SELECT ... INTO Syntax”
Section 1.8.2.1, “SELECT INTO TABLE”
Section 13.2.8, “SELECT Syntax”

SELECT ... INTO DUMPFILE

Section 2.18.1, “Initializing the Data Directory”
Section 2.17.1, “Installing MySQL Using a Standard Source Distribution”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.4, “Server System Variables”

SELECT ... INTO OUTFILE

Section 7.1, “Backup and Recovery Types”
Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 14.2.6.2, “Forcing InnoDB Recovery”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 9.1.7, “NULL Values”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 6.2.1, “Privileges Provided by MySQL”
Section 13.2.8.1, “SELECT ... INTO Syntax”
Section 1.8.2.1, “SELECT INTO TABLE”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 1.2, “Typographical and Syntax Conventions”
Section C.7.6, “Windows Platform Limitations”

SELECT ... INTO OUTFILE 'file_name'

Section 13.2.8.1, “SELECT ... INTO Syntax”

SELECT ... INTO var_list

Section C.1, “Restrictions on Stored Programs”
Section 13.6.4, “Variables in Stored Programs”

SELECT ... LOCK IN SHARE MODE

Section 14.2.8.1, “InnoDB Lock Modes”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 14.2.8.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”
Section 13.3.6, “SET TRANSACTION Syntax”

SELECT DISTINCT

Section 8.14.2, “General Thread States”
Section C.3, “Restrictions on Subqueries”

SET

Section 12.3.4, “Assignment Operators”
Section 16.1.2.4, “Binary Log Options and Variables”
Section 18.6, “Binary Logging of Stored Programs”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 18.1, “Defining Stored Programs”
Section 5.1.5.2, “Dynamic System Variables”
Section 12.1, “Function and Operator Reference”
Chapter 12, *Functions and Operators*
Section 12.13, “Information Functions”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.6.13, “`mysql_find_rows` — Extract SQL Statements from Files”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 12.3, “Operators”
Section 8.10.3.3, “Query Cache Configuration”
Section 16.1.2.2, “Replication Master Options and Variables”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.7.4, “SET Syntax”
Section 13.7.5.36, “SHOW VARIABLES Syntax”
Section 13.2.9, “Subquery Syntax”
Section 18.3.1, “Trigger Syntax and Examples”
Section 9.4, “User-Defined Variables”
Section 4.2.8, “Using Options to Set Program Variables”
Section 5.1.5, “Using System Variables”
Using the --safe-updates Option
Section 13.6.4, “Variables in Stored Programs”

SET autocommit

Section 8.6.4, “Bulk Data Loading for InnoDB Tables”
Section 13.3, “MySQL Transactional and Locking Statements”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

SET GLOBAL

Section 5.1.5.2, “Dynamic System Variables”
Section 13.7.1.3, “GRANT Syntax”
Section 8.10.1.2, “Multiple Key Caches”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.7.4, “SET Syntax”
Section 5.1.5, “Using System Variables”

SET PASSWORD

Section 6.3.5, “Assigning Account Passwords”
Section B.5.2.4, “Client does not support authentication protocol”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 16.4.1.18, “Replication and User Privileges”
Section 16.4.1.29, “Replication and Variables”
Section 16.4.1.17, “Replication of the mysql System Database”
Resetting the Root Password: Generic Instructions
Section 2.18.4, “Securing the Initial MySQL Accounts”
Section 5.1.4, “Server System Variables”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 13.7.4, “SET Syntax”
Section 6.2.3, “Specifying Account Names”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 6.2.6, “When Privilege Changes Take Effect”

SET PASSWORD ... = PASSWORD()

Section 6.3.5, “Assigning Account Passwords”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SET SESSION

Section 5.1.5.2, “Dynamic System Variables”
Section 13.7.4, “SET Syntax”
Section 5.1.5, “Using System Variables”

SET sql_mode='modes'

Section A.3, “MySQL 5.0 FAQ: Server SQL Mode”

SET TIMESTAMP = value

Section 8.14, “Examining Thread Information”

SET TRANSACTION

Section 14.2.8, “InnoDB Transaction Model and Locking”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

SET TRANSACTION ISOLATION LEVEL

Section 13.7.4, “SET Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

SHOW

Section 20.6.5, “C API Data Structures”
Section 20.6.6, “C API Function Overview”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 3.3, “Creating and Using a Database”
Section 14.2.3, “Creating and Using InnoDB Tables”
Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 19.18, “Extensions to SHOW Statements”
Chapter 19, *INFORMATION_SCHEMA Tables*
Section A.13, “MySQL 5.0 FAQ: Replication”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 20.6.11.27, “mysql_stmt_store_result()”
Section 20.6.7.69, “mysql_store_result()”
Section 20.6.7.71, “mysql_use_result()”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section C.1, “Restrictions on Stored Programs”
Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 13.7.5.18, “SHOW INDEX Syntax”
Section 13.7.5.23, “SHOW OPEN TABLES Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 13.7.5, “SHOW Syntax”
Section 13.7.5.34, “SHOW TABLES Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 18.2.3, “Stored Routine Metadata”
Section 5.4.3, “The Binary Log”
Section 19.1, “The INFORMATION_SCHEMA CHARACTER_SETS Table”
Section 19.3, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”
Section 19.2, “The INFORMATION_SCHEMA COLLATIONS Table”
Section 19.5, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”
Section 19.4, “The INFORMATION_SCHEMA COLUMNS Table”
Section 19.6, “The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table”
Section 19.7, “The INFORMATION_SCHEMA PROFILING Table”
Section 19.10, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”
Section 19.9, “The INFORMATION_SCHEMA SCHEMATA Table”
Section 19.11, “The INFORMATION_SCHEMA STATISTICS Table”
Section 19.13, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”
Section 19.14, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
Section 19.12, “The INFORMATION_SCHEMA TABLES Table”
Section 19.15, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 19.16, “The INFORMATION_SCHEMA USER_PRIVILEGES Table”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.3.2, “The Main Features of MySQL”
Section 10.1.11, “UTF-8 for Metadata”

SHOW BINARY LOGS

Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 13.7.5.1, “SHOW BINARY LOGS Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”

SHOW BINLOG EVENTS

Section C.2, “Restrictions on Server-Side Cursors”
Section 13.7.5.2, “SHOW BINLOG EVENTS Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 13.4.2.7, “START SLAVE Syntax”

SHOW CHARACTER SET

Section 13.1.1, “ALTER DATABASE Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 10.1.2, “Character Sets and Collations in MySQL”
Section 10.1.13, “Character Sets and Collations That MySQL Supports”
Section 19.18, “Extensions to SHOW Statements”
Section 13.7.5.3, “SHOW CHARACTER SET Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”

SHOW COLLATION

Section 13.1.1, “ALTER DATABASE Syntax”
Section 20.6.5, “C API Data Structures”
Section 10.5, “Character Set Configuration”
Section 10.1.2, “Character Sets and Collations in MySQL”
Section 10.1.3.5, “Character String Literal Character Set and Collation”
Section 10.4.2, “Choosing a Collation ID”
Section 10.1.3.4, “Column Character Set and Collation”
Section 10.1.3.2, “Database Character Set and Collation”
Section 2.17.3, “MySQL Source-Configuration Options”
Section 13.7.5.4, “SHOW COLLATION Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 10.1.3.3, “Table Character Set and Collation”
Section 19.3, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”
Section 19.2, “The INFORMATION_SCHEMA COLLATIONS Table”

SHOW COLUMNS

Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 13.8.2, “EXPLAIN Syntax”
Section 19.18, “Extensions to SHOW Statements”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 8.2.1.15, “LIMIT Query Optimization”
Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”
Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”

SHOW COLUMNS FROM tbl_name LIKE 'enum_col'

Section 11.4.4, “The ENUM Type”

SHOW COUNT()

Section 13.7.5.14, “SHOW ERRORS Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”

SHOW CREATE DATABASE

Section 5.1.4, “Server System Variables”
Section 13.7.5.6, “SHOW CREATE DATABASE Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”

SHOW CREATE FUNCTION

Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 1.7, “How to Report Bugs or Problems”
Section A.4, “MySQL 5.0 FAQ: Stored Procedures and Functions”
Section 13.7.5.8, “SHOW CREATE PROCEDURE Syntax”
Section 18.2.3, “Stored Routine Metadata”

SHOW CREATE PROCEDURE

Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 1.7, “How to Report Bugs or Problems”
Section A.4, “MySQL 5.0 FAQ: Stored Procedures and Functions”
Section 13.7.5.7, “SHOW CREATE FUNCTION Syntax”
Section 18.2.3, “Stored Routine Metadata”

SHOW CREATE SCHEMA

Section 13.7.5.6, “SHOW CREATE DATABASE Syntax”

SHOW CREATE TABLE

Section 14.5.4, “Characteristics of BDB Tables”
Section 13.1.8, “CREATE INDEX Syntax”
Section 13.1.10, “CREATE TABLE Syntax”
Section 11.6, “Data Type Default Values”
Section 13.8.2, “EXPLAIN Syntax”
Section 3.4, “Getting Information About Databases and Tables”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 14.7.2, “How to Use FEDERATED Tables”
Section 17.1.5.1, “Noncompliance with SQL Syntax in MySQL Cluster”
Section 2.19.4, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 13.7.5.9, “SHOW CREATE TABLE Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 13.1.10.4, “Silent Column Specification Changes”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”
Section 3.6.6, “Using Foreign Keys”

SHOW CREATE VIEW

Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 13.7.1.3, “GRANT Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 6.2.1, “Privileges Provided by MySQL”
Section C.4, “Restrictions on Views”
Section 13.7.5.10, “SHOW CREATE VIEW Syntax”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 18.4.5, “View Metadata”

SHOW DATABASES

Section 13.1.6, “CREATE DATABASE Syntax”
Section 3.3, “Creating and Using a Database”
Section 19.18, “Extensions to SHOW Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 13.7.1.3, “GRANT Syntax”
Section 6.2.2, “Grant Tables”
Section 9.2.2, “Identifier Case Sensitivity”
Chapter 19, *INFORMATION_SCHEMA Tables*
Section 2.20.1.6, “Linux SPARC Notes”
Section 13.4.2.2, “LOAD DATA FROM MASTER Syntax”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 17.5.10.2, “MySQL Cluster and MySQL Privileges”
Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”
Section 5.1.3, “Server Command Options”
Section 5.1.4, “Server System Variables”
Section 13.7.5.11, “SHOW DATABASES Syntax”

SHOW ENGINE

Section 13.7.5.12, “SHOW ENGINE Syntax”

SHOW ENGINE BDB LOGS

Section 13.7.5.12, “SHOW ENGINE Syntax”
Section 13.7.5.20, “SHOW LOGS Syntax”

SHOW ENGINE INNODB MUTEX

Section 5.1.4, “Server System Variables”
Section 13.7.5.22, “SHOW MUTEX STATUS Syntax”

SHOW ENGINE INNODB STATUS

Section 14.2.8.9, “How to Cope with Deadlocks”
Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”
Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”
InnoDB Standard Monitor and Lock Monitor Output
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 14.2.13.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”
Section 13.7.5.12, “SHOW ENGINE Syntax”
Section 13.7.5.19, “SHOW INNODB STATUS Syntax”
Section B.1, “Sources of Error Information”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”

SHOW ENGINE NDB STATUS

Section 17.5, “Management of MySQL Cluster”
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”
Section 13.7.5.12, “SHOW ENGINE Syntax”

SHOW ENGINE NDBCLUSTER STATUS

mysqld Command Options for MySQL Cluster
Section 17.5.9, “Quick Reference: MySQL Cluster SQL Statements”
Section 13.7.5.12, “SHOW ENGINE Syntax”

SHOW ENGINES

Section A.10, “MySQL 5.0 FAQ: MySQL Cluster”
MySQL Cluster System Variables
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 2.10.4.3, “Selecting a MySQL Server Type”
Section 5.1.4, “Server System Variables”
Section 13.7.5.13, “SHOW ENGINES Syntax”
Chapter 14, *Storage Engines*
Section 14.2, “The InnoDB Storage Engine”

SHOW ERRORS

Section 14.2.3.4, “InnoDB and FOREIGN KEY Constraints”
Section 5.1.4, “Server System Variables”
Section 13.7.5.14, “SHOW ERRORS Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section B.1, “Sources of Error Information”

SHOW FULL COLUMNS

Section 13.1.10, “CREATE TABLE Syntax”
Section 10.1.9.3, “SHOW Statements and INFORMATION_SCHEMA”
Section 19.5, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”

SHOW FULL PROCESSLIST

Section 8.14, “Examining Thread Information”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

SHOW FULL TABLES

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 13.7.5.34, “SHOW TABLES Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SHOW FUNCTION CODE

Section 13.7.5.15, “SHOW FUNCTION CODE Syntax”
Section 13.7.5.25, “SHOW PROCEDURE CODE Syntax”

SHOW FUNCTION STATUS

Section 13.7.5.26, “SHOW PROCEDURE STATUS Syntax”
Section 18.2.3, “Stored Routine Metadata”

SHOW GLOBAL STATUS

Section 5.1.4, “Server System Variables”

SHOW GRANTS

Section 6.3.2, “Adding User Accounts”
Section 13.7.1.2, “DROP USER Syntax”
Section 13.7.1.3, “GRANT Syntax”
Section 6.2.2, “Grant Tables”
Section 13.7.1.5, “REVOKE Syntax”
Section 6.1.1, “Security Guidelines”
Section 13.7.5.17, “SHOW GRANTS Syntax”
Section 13.7.5.24, “SHOW PRIVILEGES Syntax”
Section 6.2, “The MySQL Access Privilege System”

SHOW INDEX

Section 13.7.2.1, “ANALYZE TABLE Syntax”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.9.2, “Index Hints”
Section 14.2.14, “Limits on InnoDB Tables”
Section 8.3.7, “MyISAM Index Statistics Collection”
Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”
Section 4.6.3.4, “Other myisamchk Options”
Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 13.7.5.18, “SHOW INDEX Syntax”
Section 19.11, “The INFORMATION_SCHEMA STATISTICS Table”
Section 19.13, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”

SHOW INNODB STATUS

Section 13.7.5.12, “SHOW ENGINE Syntax”
Section 13.7.5.19, “SHOW INNODB STATUS Syntax”

SHOW MASTER LOGS

Section 13.7.5.1, “SHOW BINARY LOGS Syntax”

SHOW MASTER STATUS

Section 16.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 16.4.4, “Troubleshooting Replication”

SHOW MUTEX STATUS

Section 13.7.5.22, “SHOW MUTEX STATUS Syntax”

SHOW OPEN TABLES

Section 13.7.5.23, “SHOW OPEN TABLES Syntax”

SHOW PRIVILEGES

Section 13.7.5.24, “SHOW PRIVILEGES Syntax”

SHOW PROCEDURE CODE

Section 13.7.5.15, “SHOW FUNCTION CODE Syntax”
Section 13.7.5.25, “SHOW PROCEDURE CODE Syntax”

SHOW PROCEDURE STATUS

Section 13.7.5.16, “SHOW FUNCTION STATUS Syntax”
Section 18.2.3, “Stored Routine Metadata”

SHOW PROCESSLIST

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 16.1.3.1, “Checking Replication Status”
Section 8.14.2, “General Thread States”
Section 13.7.1.3, “GRANT Syntax”
Section 12.13, “Information Functions”
Section 14.2.12, “InnoDB Error Handling”
Section 13.7.6.3, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section A.13, “MySQL 5.0 FAQ: Replication”
Section 17.5.4, “MySQL Server Usage for MySQL Cluster”
Section 20.6.7.43, “mysql_list_processes()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 6.2.1, “Privileges Provided by MySQL”
Section 16.2.1, “Replication Implementation Details”
Section 13.7.5.27, “SHOW PROCESSLIST Syntax”
Section 13.7.5.28, “SHOW PROFILE Syntax”
Section 13.7.5.31, “SHOW SLAVE STATUS Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 16.3.6, “Switching Masters During Failover”
Section B.5.2.7, “Too many connections”
Section 16.4.4, “Troubleshooting Replication”

SHOW PROFILE

Section 8.14, “Examining Thread Information”
Section 8.14.2, “General Thread States”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 2.17.3, “MySQL Source-Configuration Options”
Section 5.1.4, “Server System Variables”
Section 13.7.5.28, “SHOW PROFILE Syntax”
Section 13.7.5.29, “SHOW PROFILES Syntax”
Section 19.7, “The INFORMATION_SCHEMA PROFILING Table”

SHOW PROFILES

Section 2.17.3, “MySQL Source-Configuration Options”
Section 5.1.4, “Server System Variables”
Section 13.7.5.28, “SHOW PROFILE Syntax”
Section 13.7.5.29, “SHOW PROFILES Syntax”
Section 19.7, “The INFORMATION_SCHEMA PROFILING Table”

SHOW SCHEMAS

Section 13.7.5.11, “SHOW DATABASES Syntax”

SHOW SLAVE HOSTS

Section 16.1.3.1, “Checking Replication Status”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 13.4.1, “SQL Statements for Controlling Master Servers”

SHOW SLAVE STATUS

Section 16.1.3.1, “Checking Replication Status”
Section 16.4.5, “How to Report Replication Bugs or Problems”
Section A.13, “MySQL 5.0 FAQ: Replication”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.1, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 16.2.1, “Replication Implementation Details”
Section 8.14.6, “Replication Slave I/O Thread States”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 16.3.7, “Setting Up Replication to Use Secure Connections”
Section 13.7.5.31, “SHOW SLAVE STATUS Syntax”
Section 16.4.1.21, “Slave Errors During Replication”
Section 16.2.2.2, “Slave Status Logs”
Section B.1, “Sources of Error Information”
Section 13.4.2, “SQL Statements for Controlling Slave Servers”
Section 13.4.2.7, “START SLAVE Syntax”
Section 16.4.4, “Troubleshooting Replication”

SHOW STATUS

Section 2.19.1.1, “Changes Affecting Upgrades to 5.0”
Section 17.3.3.6, “Defining SQL and Other API Nodes in a MySQL Cluster”
Section 13.2.5.2, “INSERT DELAYED Syntax”
mysqld Command Options for MySQL Cluster
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 16.4.1.16, “Replication and Temporary Tables”
Section 16.2.1, “Replication Implementation Details”
Section 16.4.1.23, “Replication Retries and Timeouts”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.7.5.32, “SHOW STATUS Syntax”

SHOW TABLE STATUS

Section 14.2.3.3, “AUTO_INCREMENT Handling in InnoDB”
Section 13.1.10, “CREATE TABLE Syntax”
Section 14.2.3, “Creating and Using InnoDB Tables”
Section 13.8.2, “EXPLAIN Syntax”
Section 14.2.11.2, “File Space Management”
Section 14.7.2, “How to Use FEDERATED Tables”
Section 14.2.14, “Limits on InnoDB Tables”
Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”
Section 14.2.10.5, “Physical Row Structure”
Section 13.7.5.5, “SHOW COLUMNS Syntax”
Section 13.7.5.33, “SHOW TABLE STATUS Syntax”
Section 14.8, “The ARCHIVE Storage Engine”

SHOW TABLES

Section 3.3.2, “Creating a Table”
Section 19.18, “Extensions to SHOW Statements”
Section 9.2.2, “Identifier Case Sensitivity”
Chapter 19, *INFORMATION_SCHEMA Tables*
Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”
Section 13.7.5.33, “SHOW TABLE STATUS Syntax”
Section 13.7.5.34, “SHOW TABLES Syntax”
Section B.5.2.16, “Table 'tbl_name' doesn't exist”
Section B.5.6.2, “TEMPORARY Table Problems”

SHOW TRIGGERS

Section A.5, “MySQL 5.0 FAQ: Triggers”
Section 13.7.5.35, “SHOW TRIGGERS Syntax”
Section 19.15, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 18.3.2, “Trigger Metadata”

SHOW VARIABLES

Section 2.2, “Determining Your Current MySQL Version”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.5, “Running Multiple MySQL Instances on One Machine”
Section 5.1.4, “Server System Variables”
Section 13.7.4, “SET Syntax”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

SHOW WARNINGS

START SLAVE

START TRANSACTION

START TRANSACTION WITH CONSISTENT SNAPSHOT

STOP SLAVE

T

[index top]

TRUNCATE TABLE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

U

[index top]

UNION

UNION ALL

UNION DISTINCT

UNLOCK TABLES

UPDATE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 13.7.6.3, “KILL Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 14.7.3, “Limitations of the FEDERATED Storage Engine”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”
Section 12.15, “Miscellaneous Functions”
Section A.4, “MySQL 5.0 FAQ: Stored Procedures and Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.1.1, “mysql Options”
Section 20.6.7.1, “mysql_affected_rows()”
Section 4.6.12, “`mysql_explain_log` — Use EXPLAIN on Statements in Query Log”
Section 20.6.7.35, “mysql_info()”
Section 20.6.7.37, “mysql_insert_id()”
Section 20.6.7.48, “mysql_num_rows()”
Section 20.6.7.49, “mysql_options()”
Section 20.6.11.10, “mysql_stmt_execute()”
Section 20.6.11.16, “mysql_stmt_insert_id()”
Section 20.6.11.17, “mysql_stmt_num_rows()”
Section 12.3, “Operators”
Section 8.2.2, “Optimizing DML Statements”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 6.2.1, “Privileges Provided by MySQL”
Section B.5.4.2, “Problems Using DATE Columns”
Section 16.4.1.10, “Replication and LIMIT”
Section 16.4.1.19, “Replication and the Query Optimizer”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section C.4, “Restrictions on Views”
Section 13.2.9.11, “Rewriting Subqueries as Joins”
Section 2.18.4, “Securing the Initial MySQL Accounts”
Section 14.2.8.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”
Section 3.3.4.1, “Selecting All Data”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.7.1.6, “SET PASSWORD Syntax”
Section 13.3.6, “SET TRANSACTION Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”
Section 16.4.1.21, “Slave Errors During Replication”
Section 13.2.9.9, “Subquery Errors”
Section 13.2.9, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 13.3.5.3, “Table-Locking Restrictions and Conditions”
Section 10.1.7.6, “The _bin and binary Collations”
Section 14.8, “The ARCHIVE Storage Engine”
Section 5.4.3, “The Binary Log”
Section 14.10, “The BLACKHOLE Storage Engine”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.3.2, “The Main Features of MySQL”
Section 14.3, “The MERGE Storage Engine”
Section 14.1, “The MyISAM Storage Engine”
Section 6.2, “The MySQL Access Privilege System”
Section 5.1.10, “The Server Shutdown Process”
Section 1.8.2.3, “Transactions and Atomic Operations”
Section 18.3.1, “Trigger Syntax and Examples”
Section 6.2.7, “Troubleshooting Problems Connecting to MySQL”
Section 18.4.3, “Updatable and Insertable Views”
Section 1.8.2.2, “UPDATE”
Section 13.2.10, “UPDATE Syntax”
Section 13.1.10.3, “Using FOREIGN KEY Constraints”
Using the --safe-updates Option
Section 20.6.14.2, “What Results You Can Get from a Query”
Section 6.2.6, “When Privilege Changes Take Effect”
Section 20.6.14.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

UPDATE ... ()

Section 14.2.8.4, “Consistent Nonlocking Reads”

UPDATE ... WHERE ...

Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

UPDATE IGNORE

Section 5.1.7, “Server SQL Modes”

USE

Section 16.1.2.4, “Binary Log Options and Variables”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 3.3.1, “Creating and Selecting a Database”
Section 3.3, “Creating and Using a Database”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Chapter 19, *INFORMATION_SCHEMA Tables*
Section 4.5.1.1, “mysql Options”
Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.4.2, “Reloading SQL-Format Backups”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

USE db2

USE db_name

USE test

W

[index top]

WHILE

X

[index top]

XA COMMIT

XA END

XA PREPARE

XA RECOVER

XA ROLLBACK

XA START

XA START xid

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Status Variable Index

A | B | C | D | F | H | I | K | L | M | N | O | P | Q | R | S | T | U

A

[index top]

Aborted_clients
Section B.5.2.11, “Communication Errors and Aborted Connections”
Section 5.1.6, “Server Status Variables”

Aborted_connects
Section B.5.2.11, “Communication Errors and Aborted Connections”
Section 5.1.6, “Server Status Variables”

B

[index top]

Binlog_cache_disk_use
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 5.4.3, “The Binary Log”

Binlog_cache_use
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 5.4.3, “The Binary Log”

Bytes_received
Section 5.1.6, “Server Status Variables”

Bytes_sent
Section 5.1.6, “Server Status Variables”

C

[index top]

Com_flush
Section 5.1.6, “Server Status Variables”

Compression
Section 5.1.6, “Server Status Variables”

Connections
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Created_tmp_disk_tables
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

Created_tmp_files
Section 5.1.6, “Server Status Variables”

Created_tmp_tables
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

D

[index top]

Delayed_errors
Section 5.1.6, “Server Status Variables”

Delayed_insert_threads
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 5.1.6, “Server Status Variables”

Delayed_writes
Section 13.2.5.2, “INSERT DELAYED Syntax”
Section 5.1.6, “Server Status Variables”

F

[index top]

Flush_commands
Section 5.1.6, “Server Status Variables”

H

[index top]

Handler_commit
Section 5.1.6, “Server Status Variables”

Handler_delete
Section 5.1.6, “Server Status Variables”

Handler_discover
MySQL Cluster Status Variables

Handler_prepare
Section 5.1.6, “Server Status Variables”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

I

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

K

[index top]

L

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

M

[index top]

Max_used_connections

N

[index top]

Ndb_cluster_node_id

Ndb_config_from_host

Ndb_config_from_port

Ndb_number_of_data_nodes

Not_flushed_delayed_rows

O

[index top]

Open_files

Open_streams

Open_tables

Opened_tables

P

[index top]

Prepared_stmt_count

Q

[index top]

Qcache_free_blocks

Qcache_free_memory

Qcache_hits

Qcache_inserts

Qcache_lowmem_prunes

Qcache_not_cached

Qcache_queries_in_cache

Qcache_total_blocks

Queries

Questions

R

[index top]

Rpl_status

This documentation is for an older version. If you're
This documentation is for an older version. If you're

S

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're

System Variable Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W

A

[index top]

auto_increment_increment

auto_increment_offset

autocommit

automatic_sp_privileges

B

[index top]

back_log

basedir

bdb_cache_size

bdb_home

bdb_log_buffer_size

bdb_logdir

bdb_max_lock

bdb_shared_data

bdb_tmpdir

big_tables

binlog_cache_size

binlog_format

bulk_insert_buffer_size

This documentation is for an older version. If you're
This documentation is for an older version. If you're

C

[index top]

character_set_client

character_set_connection

character_set_database

character_set_filesystem

character_set_results

character_set_server

character_set_system

character_sets_dir

collation_connection

collation_database

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”
Section 14.2.2, “InnoDB Startup Options and System Variables”
InnoDB Tablespace Monitor Output
Section 16.3.4, “Replicating Different Databases to Different Slaves”
Section 14.2.13.3, “Troubleshooting InnoDB Data Dictionary Operations”

innodb_flush_log_at_trx_commit

Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 8.6.2, “Optimizing InnoDB Transaction Management”

innodb_flush_method

Section 14.2.1.4, “InnoDB File-Per-Table Tablespaces”
Section 8.6.7, “Optimizing InnoDB Disk I/O”

innodb_force_recovery

Section 14.2.6.2, “Forcing InnoDB Recovery”
Section 1.7, “How to Report Bugs or Problems”
Section 8.6.2, “Optimizing InnoDB Transaction Management”
Section 14.2.6.1, “The InnoDB Recovery Process”

innodb_lock_wait_timeout

Section 14.2.8.8, “Deadlock Detection and Rollback”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 16.4.1.23, “Replication Retries and Timeouts”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section B.3, “Server Error Codes and Messages”

innodb_locks_unsafe_for_binlog

Section 14.2.8.4, “Consistent Nonlocking Reads”
Section 14.2.8.2, “InnoDB Record, Gap, and Next-Key Locks”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 14.2.8.6, “Locks Set by Different SQL Statements in InnoDB”

innodb_log_buffer_size

Section 8.6.7, “Optimizing InnoDB Disk I/O”
Section 8.6.3, “Optimizing InnoDB Redo Logging”

innodb_log_file_size

Section 14.2.1, “Configuring InnoDB”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 8.6.7, “Optimizing InnoDB Disk I/O”
Section 8.6.3, “Optimizing InnoDB Redo Logging”

innodb_log_files_in_group

Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 8.6.3, “Optimizing InnoDB Redo Logging”

innodb_max_dirty_pages_pct

Section 8.6.7, “Optimizing InnoDB Disk I/O”

innodb_max_purge_lag

Section 14.2.9, “InnoDB Multi-Versioning”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 8.6.7, “Optimizing InnoDB Disk I/O”

innodb_open_files

Section 8.6.7, “Optimizing InnoDB Disk I/O”

innodb_support_xa

Section 14.2.8.9, “How to Cope with Deadlocks”
Section 14.2.2, “InnoDB Startup Options and System Variables”
Section 8.6.2, “Optimizing InnoDB Transaction Management”

innodb_table_locks

Section 14.2.2, “InnoDB Startup Options and System Variables”

innodb_thread_concurrency

InnoDB Standard Monitor and Lock Monitor Output
Section 14.2.2, “InnoDB Startup Options and System Variables”

innodb_use_legacy_cardinality_algorithm

Section 14.2.2, “InnoDB Startup Options and System Variables”

insert_id

Section 5.1.4, “Server System Variables”

interactive_timeout

Section B.5.2.11, “Communication Errors and Aborted Connections”
Section 2.20.4.1, “FreeBSD Notes”
Section 20.6.7.52, “mysql_real_connect()”
Section 2.20.2.1, “OS X 10.x (Darwin)”
Section 5.1.4, “Server System Variables”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

J

[index top]

join_buffer_size

K

[index top]

keep_files_on_create

key_buffer_size

key_cache_age_threshold

key_cache_block_size

key_cache_division_limit

L

[index top]

language

large_files_support

large_page_size

large_pages

last_insert_id

lc_time_names

license

local

local_infile

locked_in_memory

log

log_bin

log_bin_trust_function_creators

log_error

This documentation is for an older version. If you're
This documentation is for an older version. If you're

log_queries_not_using_indexes

log_slave_updates

log_slow_queries

log_warnings

long_query_time

low_priority_updates

lower_case_file_system

lower_case_table_names

M

[index top]

max_allowed_packet

max_binlog_cache_size

max_binlog_size

max_connect_errors

max_connections

max_delayed_threads

max_error_count

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 5.1.4, “Server System Variables”
Section 13.7.5.14, “SHOW ERRORS Syntax”
Section 13.7.5.37, “SHOW WARNINGS Syntax”

max_heap_table_size

Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section C.7.3, “Limits on Table Size”
Section 16.4.1.15, “Replication and MEMORY Tables”
Section 16.4.1.29, “Replication and Variables”
Section C.2, “Restrictions on Server-Side Cursors”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 14.4, “The MEMORY (HEAP) Storage Engine”

max_insert_delayed_threads

Section 5.1.4, “Server System Variables”

max_join_size

Section 8.8.2, “EXPLAIN Output Format”
Section 5.1.4, “Server System Variables”
Section 13.7.4, “SET Syntax”
Section 5.1.5, “Using System Variables”

max_length_for_sort_data

Section 8.2.1.11, “ORDER BY Optimization”
Section 5.1.4, “Server System Variables”

max_prepared_stmt_count

Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”
Section 13.5, “SQL Syntax for Prepared Statements”

max_relay_log_size

Section 16.1.2.4, “Binary Log Options and Variables”
Section 16.1.2.3, “Replication Slave Options and Variables”
Section 5.1.4, “Server System Variables”
Section 16.2.2.1, “The Slave Relay Log”

max_seeks_for_key

Section 14.2.14, “Limits on InnoDB Tables”
Section 5.1.4, “Server System Variables”

max_sort_length

Section 13.1.10, “CREATE TABLE Syntax”
Section B.5.7, “Known Issues in MySQL”
Section 5.1.4, “Server System Variables”
Section 11.4.3, “The BLOB and TEXT Types”

max_sp_recursion_depth

Section 5.1.4, “Server System Variables”
Section 18.2.1, “Stored Routine Syntax”

max_tmp_tables

Section 5.1.4, “Server System Variables”

max_user_connections

Section 13.7.1.3, “GRANT Syntax”
Section 6.2.2, “Grant Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.4, “Server System Variables”
Section 6.3.4, “Setting Account Resource Limits”

max_write_lock_count

Section 5.1.4, “Server System Variables”
Section 8.11.2, “Table Locking Issues”

multi_range_count

Section 5.1.4, “Server System Variables”

myisam_data_pointer_size

Section 13.1.10, “CREATE TABLE Syntax”
Section C.7.3, “Limits on Table Size”
Section 5.1.4, “Server System Variables”

myisam_max_extra_sort_file_size

Section 5.1.4, “Server System Variables”

myisam_max_sort_file_size

Section 14.1.1, “MyISAM Startup Options”
Section 5.1.4, “Server System Variables”
Section 8.5.3, “Speed of REPAIR TABLE Statements”

myisam_mmap_size

Section 5.1.4, “Server System Variables”

myisam_recover_options

Section 5.1.4, “Server System Variables”

myisam_repair_threads

Section 5.1.4, “Server System Variables”

myisam_sort_buffer_size

Section 13.1.4, “ALTER TABLE Syntax”
Section 14.1.1, “MyISAM Startup Options”
Section 5.1.4, “Server System Variables”
Section 8.5.3, “Speed of REPAIR TABLE Statements”

myisam_stats_method

Section 8.3.7, “MyISAM Index Statistics Collection”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

N

[index top]

named_pipe

ndb_autoincrement_prefetch_sz

ndb_cache_check_time

ndb_force_send

ndb_index_stat_cache_entries

ndb_index_stat_enable

ndb_index_stat_update_freq

ndb_optimized_node_selection

ndb_report_thresh_binlog_epoch_slip

ndb_report_thresh_binlog_mem_usage

ndb_use_exact_count

ndb_use_transactions

net_buffer_length

net_read_timeout

net_retry_count

net_write_timeout

new

O

[index top]

old_passwords

one_shot

open_files_limit

optimizer_prune_level

optimizer_search_depth

P

[index top]

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Section 8.2.1.11, “ORDER BY Optimization”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

sql_auto_is_null

Section 12.3.2, “Comparison Functions and Operators”
Section 13.1.10, “CREATE TABLE Syntax”
Section 16.4.1.29, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Section 5.4.3, “The Binary Log”

sql_big_selects

Section 5.1.4, “Server System Variables”

sql_buffer_result

Section 5.1.4, “Server System Variables”

sql_log_bin

Section 16.1.2.4, “Binary Log Options and Variables”
Section 17.1.5.8, “Issues Exclusive to MySQL Cluster”
Section 16.1.2.1, “Replication and Binary Logging Option and Variable Reference”
Section 5.1.4, “Server System Variables”
Section 13.4.1.3, “SET sql_log_bin Syntax”
Section 13.7.4, “SET Syntax”
Section 16.4.3, “Upgrading a Replication Setup”

sql_log_off

Section 16.1.2.4, “Binary Log Options and Variables”
Section 16.1.2.1, “Replication and Binary Logging Option and Variable Reference”
Section 5.4.5, “Server Log Maintenance”
Section 5.1.4, “Server System Variables”

sql_log_update

Section 5.1.4, “Server System Variables”

sql_mode

Section 13.1.9, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.11, “CREATE TRIGGER Syntax”
Section 12.17.3, “Expression Handling”
Section 1.7, “How to Report Bugs or Problems”
Section 13.2.6, “LOAD DATA INFILE Syntax”
Section A.11, “MySQL 5.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 1.8, “MySQL Standards Compliance”
Section B.5.4.2, “Problems Using DATE Columns”
Section 16.4.1.29, “Replication and Variables”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 13.7.5.10, “SHOW CREATE VIEW Syntax”
Section 5.4.3, “The Binary Log”
Section 19.17, “The INFORMATION_SCHEMA VIEWS Table”
Section 4.2.6, “Using Option Files”
Section 5.1.5, “Using System Variables”

sql_notes

Section 5.1.4, “Server System Variables”
Section 13.7.5.37, “SHOW WARNINGS Syntax”

sql_quote_show_create

Section 5.1.4, “Server System Variables”
Section 13.7.5.6, “SHOW CREATE DATABASE Syntax”
Section 13.7.5.9, “SHOW CREATE TABLE Syntax”

sql_safe_updates

Section 5.1.4, “Server System Variables”

sql_select_limit

Section 5.1.4, “Server System Variables”

sql_slave_skip_counter

Section 16.1.2.3, “Replication Slave Options and Variables”
Section 13.7.5.31, “SHOW SLAVE STATUS Syntax”

sql_warnings

Section 5.1.4, “Server System Variables”

ssl_ca

Section 5.1.4, “Server System Variables”

ssl_capath

Section 5.1.4, “Server System Variables”

ssl_cert

Section 5.1.4, “Server System Variables”

ssl_cipher

Section 5.1.4, “Server System Variables”

ssl_key

Section 5.1.4, “Server System Variables”

storage_engine

Section 16.4.1.29, “Replication and Variables”
Section 5.1.4, “Server System Variables”
Chapter 14, *Storage Engines*
Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're

This documentation is for an older version. If you're

unique_checks

updatable_views_with_limit

V

[index top]

version

version_bdb

version_comment

version_compile_machine

version_compile_os

W

[index top]

wait_timeout

warning_count

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're

Transaction Isolation Level Index

R | S

R

[index top]

READ COMMITTED

READ UNCOMMITTED

READ-COMMITTED

READ-UNCOMMITTED

REPEATABLE READ

REPEATABLE-READ

S

[index top]

SERIALIZABLE

This documentation is for an older version. If you're
This documentation is for an older version. If you're

This documentation is for an older version. If you're
This documentation is for an older version. If you're