MySQL 5.0 Reference Manual

MySQL 5.0 Reference Manual

Ésta es una traducción del manual de referencia de MySQL, que puede encontrarse en dev.mysql.com. El manual de referencia original de MySQL está escrito en inglés, y esta traducción no necesariamente está tan actualizada como la versión original. Para cualquier sugerencia sobre la traducción y para señalar errores de cualquier tipo, no dude en dirigirse a mysql-es@vespito.com.

Resumen

Documento generado el: 2014-05-24 (revisión: 572)

Tabla de contenidos

Prefacio

Éste es el manual de referencia para el sistema de base de datos MySQL, en su versión 5.0, hasta la versión 5.0.9-beta. No debería utilizarse con ediciones más antiguas del software MySQL, por las muchas diferencias funcionales y de otro tipo entre MySQL 5.0 y versiones anteriores. Si se está utilizando una versión anterior del software MySQL, es preferible hacer referencia al *Manual de referencia de MySQL 4.1*, que cubre las versiones 3.22, 3.23, 4.0 y 4.1 de MySQL. En este texto se señalan las diferencias entre las diversas versiones de MySQL 5.0, indicando la entrega (5.0.*x*).

La traducción al español de este manual se debe a Vespito, empresa de Barcelona especializada en la gestión de bases de datos MySQL y partner de MySQL AB desde 2001. Ha colaborado en la traducción Claudio Alberto Nipotti, de San Lorenzo (Santa Fe), Argentina.

Capítulo 1. Información general

Tabla de contenidos

El software MySQL® proporciona un servidor de base de datos SQL (Structured Query Language) muy rápido, multi-threaded, multi usuario y robusto. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido. MySQL es una marca registrada de MySQL AB.

El software MySQL tiene una doble licencia. Los usuarios pueden elegir entre usar el software MySQL como un producto Open Source bajo los términos de la licencia GNU General Public License (http://www.fsf.org/licenses/) o pueden adquirir una licencia comercial estándar de MySQL AB. Consulte http://www.mysql.com/company/legal/licensing/ para más información acerca de nuestras políticas de licencia.

La siguiente lista describe algunas secciones de particular interés en este manual:

- Para una discusión acerca de las capacidades del servidor de base de datos MySQL consulte Sección 1.4.2, “Las principales características de MySQL”.
- Para instrucciones acerca de la instalación, consulte Capítulo 2, *Instalar MySQL*.
- Para consejos sobre portar el software MySQL a nuevas arquitecturas o sistemas operativos, consulte Apéndice D, *Portar a otros sistemas*.
- Para información acerca de actualizar desde la versión 4.1, consulte Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0”.
- Para un tutorial introductorio al servidor de base de datos MySQL, consulte Capítulo 3, *Curso (tutorial) de MySQL*.

- Para ejemplos de SQL e información de rendimiento, consulte el directorio de pruebas de rendimiento (`sql-bench` en la distribución).
- Para la historia de nuevas características y fallos arreglados, consulte Apéndice C, *Historial de cambios de MySQL*.
- Para una lista de fallos conocidos y características no implementadas, consulte Sección A.8, “Problemas conocidos en MySQL”.
- Para la lista de todos los que han contribuido a este proyecto, consulte Apéndice B, *Credits*.

Importante:

Informes de errores (a menudo llamados "bugs"), así como preguntas y comentarios, deben enviarse a http://bugs.mysql.com. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

Si encuentra un fallo de seguridad importante en el servidor MySQL, por favor comuníquelo inmediatamente mediante un correo electrónico a `<security@mysql.com>`.

1.1. Sobre este manual

Este es el manual de referencia para el servidor de base de datos MySQL, versión 5.0, hasta la versión 5.0.9-beta. No está destinado para usarse con versiones más antiguas del software MySQL debido a las numerosas diferencias funcionales y de otro tipo entre MySQL 5.0 y versiones previas. Si usa una versión anterior del software MySQL, por favor consulte *Manual de referencia de MySQL 4.1*, que cubre las series 3.22, 3.23, 4.0, y 4.1 del software MySQL. Las diferencias entre versiones menores de MySQL 5.0 están destacadas en este texto con referencias a los números de versiones (5.0.*x*).

Este manual es una referencia, por lo que no proporciona instrucciones sobre conceptos generales de SQL o de bases de datos relacionales. Tampoco enseña sobre cómo usar su sistema operativo o su intérprete de línea de comandos.

El software de base de datos MySQL está bajo desarrollo constante, y el Manual de Referencia se actualiza constantemente. La versión más reciente de este manual está disponible en línea permitiendo búsquedas en http://dev.mysql.com/doc/. Hay otros formatos disponibles, incluyendo HTML, PDF, y Windows CHM.

El formato básico para toda la documentación MySQL consiste en un conjunto de ficheros DocBook XML. HTML y otros formatos se producen automáticamente a partir de los mismos, usando entre otras herramientas DocBook XSL stylesheets.

Si tiene sugerencias acerca de correcciones o contenido de este manual, por favor envíelos al equipo de documentación en http://www.mysql.com/company/contact/.

Este manual lo escribieron inicialmente David Axmark y Michael "Monty" Widenius. Lo mantiene el equipo de documentación de MySQL formado por Paul DuBois, Stefan Hinz, Mike Hillyer, y Jon Stephens. Para consultar todos los colaboradores, consulte Apéndice B, *Credits*.

1.2. Convenciones utilizadas en este manual

Este manual usa ciertas convenciones tipográficas:

- `El texto de este estilo` se usa para sentencias SQL; nombres de bases de datos, tablas y columnas; código C y Perl; y variables de entorno. Ejemplo: "Para recargar las tablas de permisos use el comando `FLUSH PRIVILEGES`".
- *`El texto de este estilo`* se usa para entrada de variables las cuales debe substituir por un valor de su propia elección.
- Nombres de ficheros y directorios se escriben así: "El fichero global `my.cnf` se encuentra en el directorio `/etc`".
- Las secuencias de caracteres se escriben así: "Para especificar un comodín, use el carácter '`%`'."
- *El texto de este estilo* se usa para enfatizar.
- **El texto de este estilo** se usa en las cabeceras y para dar un enfásis especialmente fuerte.

Cuando se muestran comandos que deben ser ejecutados en un programa particular, el programa se indica con un prompt mostrado antes del comando. Por ejemplo, `shell>` indica un comando que se ejecuta desde el login shell, y `mysql>` indica un comando que se ejecuta desde el programa cliente `mysql` :

```
shell> escriba un comando de shell aquí
mysql> escriba un comando mysql aquí
```

El "shell" es su intérprete de comandos. En Unix, esto es normalmente un programa como `sh` `csh`. En Windows, el programa equivalente es `command.com` o `cmd.exe`, normalmente ejecutado en una ventana de consola.

Cuando introduzca un comando u órden no escriba el prompt mostrado en el ejemplo.

Nombres de bases de datos, tablas y columnas a menudo deben reemplazarse en los comandos. Para indicar que dicha substitución es necesaria, el manual usa *`db_name`*, *`tbl_name`*, y *`col_name`*. Por ejemplo, puede ver un comando como este:

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

Significa que si quisiera introducir un comando similar, debería escribir su propio nombre de base de datos, tabla y columna, tal vez así:

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

En SQL no tiene importancia si las palabras clave se escriben en mayúsculas o minúsculas. Este manual usa mayúsculas.

En descripciones de sintaxis, corchetes ('`[`' y '`]`') se usan para indicar palabras o cláusulas opcionales. Por ejemplo, en el siguiente comando, `IF EXISTS` es opcional:

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

Cuando un elemento de sintaxis consiste en un número de alternativas, las alternativas se separan mediante barras verticales ('`|`'). Cuando un miembro de una serie de elecciones *puede* ser elegido, las alternativas se muestran entre corchetes ('`[`' y '`]`'):

```
```

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

Cuando un miembro de una serie de elecciones *debe* ser elegido, las alternativas se muestran entre llaves ('{' y '}'):

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

Puntos suspensivos (`...`) indica la omisión de una parte del comando, típicamente para proporcionar una versión corta de una sintaxis más compleja. Por ejemplo, `INSERT ... SELECT` es la versión corta de un comando `INSERT` seguido de un comando `SELECT`.

Puntos suspensivos pueden tambier indicar que el elemento de sintaxis precedente de un comando puede repertirse. En el siguiente ejemplo, pueden darse varios valores *`reset_option`* cada uno de ellos tras el primero precedidos por comas:

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

Los comandos para inicializar variables del shell se muestran usando la sintaxis del shell Bourne. Por ejemplo, la secuencia para incializar una variable de entorno y ejecutar un comando es la siguiente con la sintaxis del shell Bourne:

```
shell> VARNAME=value some_command
```

Si utiliza `csh` o `tcsh`, debe proporcionar comandos ligeramente distintos. Debería ejecutar la secuencia anterior así:

```
shell> setenv VARNAME value
shell> some_command
```

1.3. Panorámica de MySQL AB

MySQL AB de los fundadores de MySQL y principales desarrolladores. MySQL AB se estableció originalmente en Suecía por David Axmark, Allan Larsson, y Michael "Monty" Widenius.

Nos dedicamos a desarrollar el software para la base de datos MySQL y promocionarlo a nuevos usuarios. MySQL AB posee el copyright del código fuente MySQL, el logo MySQL y la marca registrada, y su manual. Consulte Sección 1.4, “Panorámica del sistema de gestión de base de datos MySQL”.

Los valores clave MySQL muestran nuestra dedicación a MySQL y Open Source.

Los valores clave dirigen cómo MySQL AB trabaja el software de base de datos MySQL:

- Ser la mejor y más usada base de datos en el mundo.
- Estar disponible y ser comprable por cualquiera.
- Fácil de usar.
- Mejorarlo contínuamente mientras es rápido y seguro.
- Ser divertido de usar y mejorar.
- Libre de errores.

Estos son los valores clave de la compañía MySQL AB y sus empleados:

- Suscribimos la filosofía Open Source y apoyamos la comunidad Open Source.
- Ser buenos ciudadanos.
- Preferimos socios que compartan nuestros valores y forma de pensar.
- Responder los correos electrónicos y proporcionar soporte.
- Somos una compañía virtual, conectada con otras.
- Estamos en contra de las patentes de software.

El sitio web MySQL (http://www.mysql.com/) proporciona la última información sobre MySQL y MySQL AB

La parte "AB" del nombre de la compañía es el acrónimo del sueco "aktiebolag", o "stock company", o "sociedad anónima". Se traduce como "MySQL, Inc" o "MySQL, SA". De hecho, MySQL, Inc. y MySQL GmbH son ejemplos de empresas subsidiarias de MySQL AB. Están establecidas en los Estados Unidos y Alemania respectivamente.

1.4. Panorámica del sistema de gestión de base de datos MySQL

MySQL, el sistema de gestión de bases de datos SQL Open Source más popular, lo desarrolla, distribuye y soporta MySQL AB. MySQL AB es una compañía comercial, fundada por los desarrolladores de MySQL. Es una compañía Open Source de segunda generación que une los valores y metodología Open Source con un exitoso modelo de negocio.

El sitio web MySQL (http://www.mysql.com/) proporciona la última información sobre MySQL y MySQL AB.

- MySQL es un sistema de gestión de bases de datos

  Una base de datos es una colección estruturada de datos. Puede ser cualquier cosa, desde una simple lista de compra a una galería de pintura o las más vastas cantidades de información en una red corporativa. Para añadir, acceder, y procesar los datos almacenados en una base de datos, necesita un sistema de gestión de base de datos como MySQL Server. Al ser los computadores muy buenos en tratar grandes cantidades de datos, los sistemas de gestión de bases de datos juegan un papel central en computación, como aplicaciones autónomas o como parte de otras aplicaciones.

- MySQL es un sistema de gestión de bases de datos relacionales

  Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén. Esto añade velocidad y flexibilidad. La parte SQL de "MySQL"se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos y está definido por el estándar ANSI/ISO SQL. El estándar SQL ha evolucionado desde 1986 y existen varias versiones. En este manual, "SQL-92" se refiere al estándar del 1992, "SQL:1999" se refiere a la versión del 1999, y "SQL:2003" se refiere a la versión actual del estándar. Usamos la frase "el estándar SQL" para referirnos a la versión actual de SQL.

- MySQL software es Open Source.

  Open Source significa que es posible para cualquiera usar y modificar el software. Cualquiera puede bajar el software MySQL desde internet y usarlo sin pagar nada. Si lo desea, puede estudiar el código fuente y cambiarlo para adapatarlo a sus necesidades. El software MySQL usa la licencia GPL (GNU General Public License), http://www.fsf.org/licenses/, para definir lo que puede y no puede hacer con el software en diferentes situaciones. Si no se encuentra cómodo con la GPL o necesita añadir código

MySQL en una aplicación comercial, puede comprarnos una licencia comercial. Consulte la Introducción a las Licencias MySQL para más información (http://www.mysql.com/company/legal/licensing/).

- El servidor de base de datos MySQL es muy rápido, fiable y fácil de usar.

  Si esto es lo que está buscando, debería probarlo. El servidor MySQL también tiene una serie de características prácticas desarrolladas en cooperación con los usuarios. Puede encontrar comparaciones de rendimiento de MySLQL Server con otros sistemas de gestión de bases de datos en nuestra página de comparativas de rendimiento. Consulte Sección 7.1.4, “El paquete de pruebas de rendimiento (benchmarks) de MySQL”.

  MySQL Server se desarrolló originalmente para tratar grandes bases de datos mucho más rápido que soluciones existentes y ha sido usado con éxito en entornos de producción de alto rendimiento durante varios años. MySQL Server ofrece hoy en día una gran cantidad de funciones. Su conectividad, velocidad, y seguridad hacen de MySQL Server altamente apropiado para acceder bases de datos en Internet

- MySQL Server trabaja en entornos cliente/servidor o incrustados

  El software de bases de datos MySQL es un sistema cliente/servidor que consiste en un servidor SQL multi-threaded que trabaja con diferentes bakends, programas y bibliotecas cliente, herramientas administrativas y un amplio abanico de interfaces de programación para aplicaciones (APIs).

  También proporcionamos el MySQL Server como biblioteca incrustada multi-threaded que puede lincar en su aplicación para obtener un producto más pequeño, rápido y fácil de administrar.

- Una gran cantidad de software de contribuciones está disponible para MySQL

  Es muy posible que su aplicación o lenguaje favorito soporte el servidor de base de datos MySQL.

La forma oficial de pronunciar "MySQL" es "My Ess Que Ell" (no "my sicuel"), pero no importa si lo pronuncia como "my sicuel" o de alguna otra forma.

1.4.1. Historia de MySQL

Empezamos con la intención de usar `mSQL` para conectar a nuestras tablas utilizando nuestras propias rutinas rápidas de bajo nivel (ISAM). Sin embargo y tras algunas pruebas, llegamos a la conclusión que `mSQL` no era lo suficientemente rápido o flexible para nuestras necesidades. Esto provocó la creación de una nueva interfaz SQL para nuestra base de datos pero casi con la misma interfaz API que `mSQL`. Esta API fue diseñada para permitir código de terceras partes que fue escrito para poder usarse con `mSQL` para ser fácilmente portado para el uso con MySQL.

La derivación del nombre MySQL no está clara. Nuestro directorio base y un gran número de nuestras bibliotecas y herramientas han tenido el prefijo "my" por más de 10 años. Sin embargo, la hija del co-fundador Monty Widenius también se llama My. Cuál de los dos dió su nombre a MySQL todavía es un misterio, incluso para nosotros.

El nombre del delfín de MySQL (nuestro logo) es "Sakila", que fué elegido por los fundadores de MySQL AB de una gran lista de nombres sugerida por los usuarios en el concurso "Name the Dolphin" (ponle nombre al delfín). El nombre ganador fue enviado por Ambrose Twebaze, un desarrollador de software Open Source de Swaziland, África. Según Ambrose, el nombre femenino de Sakila tiene sus raíces en SiSwate, el idioma local de Swaziland. Sakila también es el nombre de una ciudad en Arusha, Tanzania, cerca del país de origen de Ambrose, Uganda.

1.4.2. Las principales características de MySQL

La siguiente lista describe algunas de las características más importantes del software de base de datos MySQL. Consulte Sección 1.5, “Mapa de desarrollo de MySQL” para más información acerca de las características actuales y próximas.

- Interioridades y portabilidad
  - Escrito en C y en C++
  - Probado con un amplio rango de compiladores diferentes
  - Funciona en diferentes plataformas. Consulte Sección 2.1.1, “Sistemas operativos que MySQL soporta”.
  - Usa GNU Automake, Autoconf, y Libtool para portabilidad.
  - APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl. Consulte Capítulo 24, *APIs de MySQL*.
  - Uso completo de multi-threaded mediante threads del kernel. Pueden usarse fácilmente multiple CPUs si están disponibles.
  - Proporciona sistemas de almacenamiento transaccionales y no transaccionales.
  - Usa tablas en disco B-tree (`MyISAM`) muy rápidas con compresión de índice.
  - Relativamente sencillo de añadir otro sistema de almacenamiento. Esto es útil si desea añadir una interfaz SQL para una base de datos propia.
  - Un sistema de reserva de memoria muy rápido basado en threads.
  - Joins muy rápidos usando un multi-join de un paso optimizado.
  - Tablas hash en memoria, que son usadas como tablas temporales.
  - Las funciones SQL están implementadas usando una librería altamente optimizada y deben ser tan rápidas como sea posible. Normalmente no hay reserva de memoria tras toda la inicialización para consultas.
  - El código MySQL se prueba con Purify (un detector de memoria perdida comercial) así como con Valgrind, una herramienta GPL (http://developer.kde.org/~sewardj/).
  - El servidor está disponible como un programa separado para usar en un entorno de red cliente/servidor. También está disponible como biblioteca y puede ser incrustado (linkado) en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible..
- Tipos de columnas
  - Diversos tipos de columnas: enteros con/sin signo de 1, 2, 3, 4, y 8 bytes de longitud, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET`, `ENUM`, y tipos espaciales OpenGIS. Consulte Capítulo 11, *Tipos de columna*.
  - Registros de longitud fija y longitud variable.
- Sentencias y funciones
  - Soporte completo para operadores y funciones en las cláusulas de consultas `SELECT` y `WHERE`. Por ejemplo:

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

- Soporte completo para las cláusulas SQL `GROUP BY` y `ORDER BY`. Soporte de funciones de agrupación (`COUNT()`, `COUNT(DISTINCT ...)`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()`, y `GROUP_CONCAT()`).
- Soporte para `LEFT OUTER JOIN` y `RIGHT OUTER JOIN` cumpliendo estándares de sintaxis SQL y ODBC.
- Soporte para alias en tablas y columnas como lo requiere el estándar SQL.
- `DELETE`, `INSERT`, `REPLACE`, y `UPDATE` devuelven el número de filas que han cambiado (han sido afectadas). Es posible devolver el número de filas que serían afectadas usando un flag al conectar con el servidor.
- El comando específico de MySQL `SHOW` puede usarse para obtener información acerca de la base de datos, el motor de base de datos, tablas e índices. El comando `EXPLAIN` puede usarse para determinar cómo el optimizador resuelve una consulta.
- Los nombres de funciones no colisionan con los nombres de tabla o columna. Por ejemplo, `ABS` es un nombre válido de columna. La única restricción es que para una llamada a una función, no se permiten espacios entre el nombre de función y el '`(`' a continuación. Consulte Sección 9.6, “Tratamiento de palabras reservadas en MySQL”.
- Puede mezclar tablas de distintas bases de datos en la misma consulta (como en MySQL 3.22).

- Seguridad
  - Un sistema de privilegios y contraseñas que es muy flexible y seguro, y que permite verficación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está cifrado cuando se conecta con un servidor.
- Escalabilidad y límites
  - Soporte a grandes bases de datos. Usamos MySQL Server con bases de datos que contienen 50 millones de registros. También conocemos a usuarios que usan MySQL Server con 60.000 tablas y cerca de 5.000.000.000.000 de registros.
  - Se permiten hasta 64 índices por tabla (32 antes de MySQL 4.1.2). Cada índice puede consistir desde 1 hasta 16 columnas o partes de columnas. El máximo ancho de límite son 1000 bytes (500 antes de MySQL 4.1.2).Un índice puede usar prefijos de una columna para los tipos de columna `CHAR`, `VARCHAR`, `BLOB`, o `TEXT`.
- Conectividad
  - Los clientes pueden conectar con el servidor MySQL usando sockets TCP/IP en cualquier plataforma. En sistemas Windows de la familia NT (NT,2000,XP, o 2003), los clientes pueden usar named pipes para la conexión. En sistemas Unix, los clientes pueden conectar usando ficheros socket Unix.
  - En MySQL 5.0, los servidores Windows soportan conexiones con memoria compartida si se inicializan con la opción `--shared-memory`. Los clientes pueden conectar a través de memoria compartida usando la opción `--protocol=memory`.

  - La interfaz para el conector ODBC (MyODBC) proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (Open Database Connectivity). Por ejemplo, puede usar MS Access para conectar al servidor MySQL. Los clientes pueden ejecutarse en Windows o Unix. El código fuente de MyODBC está disponible. Todas las funciones para ODBC 2.5 están soportadas, así como muchas otras. Consulte Sección 25.1, “MySQL Connector/ODBC”.

  - La interfaz para el conector J MySQL proporciona soporte para clientes Java que usen conexiones JDBC. Estos clientes pueden ejecutarse en Windows o Unix. El código fuente para el conector J está disponible. Consulte Sección 25.4, “MySQL Connector/J”.

- Localización

  - El servidor puede proporcionar mensajes de error a los clientes en muchos idomas. Consulte Sección 5.9.2, “Escoger el idioma de los mensajes de error”.

  - Soporte completo para distintos conjuntos de caracteres, incluyendo `latin1` (ISO-8859-1), `german`, `big5`, `ujis`, y más. Por ejemplo, los caracteres escandinavos 'â', 'ä' y 'ö' están permitidos en nombres de tablas y columnas. El soporte para Unicode está disponible

  - Todos los datos se guardan en el conjunto de caracteres elegido. Todas las comparaciones para columnas normales de cadenas de caracteres son case-insensitive.

  - La ordenación se realiza acorde al conjunto de caracteres elegido (usando colación Sueca por defecto). Es posible cambiarla cuando arranca el servidor MySQL. Para ver un ejemplo de ordenación muy avanzada, consulte el código Checo de ordenación. MySQL Server soporta diferentes conjuntos de caracteres que deben ser especificados en tiempo de compilación y de ejecución.

- Clientes y herramientas

  - MySQL server tiene soporte para comandos SQL para chequear, optimizar, y reparar tablas. Estos comandos están disponibles a través de la línea de comandos y el cliente `mysqlcheck`. MySQL también incluye `myisamchk`, una utilidad de línea de comandos muy rápida para efectuar estas operaciones en tablas `MyISAM`. Consulte Capítulo 5, *Administración de bases de datos*.

  - Todos los programas MySQL pueden invocarse con las opciones `--help` o `-?` para obtener asistencia en línea.

1.4.3. Estabilidad de MySQL

Esta sección trata las preguntas "*¿Qué estabilidad tiene MySQL Server?*" y, "*¿Puedo fiarme de MySQL Server para este proyecto?*" Intentaremos clarificar estas cuestiones y responder algunas preguntas importantes que preocupan a muchos usuarios potenciales. La información en esta sección se basa en datos recopilados de las listas de correo, que son muy activas para identificar problemas así como para reportar tipos de usos.

El código original se remonta a los principos de los años 80. En TcX, la predecesora de MySQL AB, el código MySQL ha funcionado en proyectos desde mediados de 1996 sin ningún problema. Cuando el software de base de datos MySQL fue distribuído entre un público más amplio, nuestros nuevos usuarios rápidamente encontraron trozos de código no probados. Cada nueva versión desde entonces ha tenido pocos problemas de portabilidad incluso considerando que cada nueva versión ha tenido muchas nuevas funcionalidades.

Cada versión de MySQL Server ha sido usable. Los problemas han ocurrido únicamente cuando los usuarios han probado código de las "zonas grises". Naturalmente, los nuevos usuarios no conocen cuáles son estas zonas; esta sección, por lo tanto, trata de documentar dichas áreas conocidas a día de hoy.

Las descripciones mayormente se corresponden con la versión 3.23, 4.0 y 4.1 de MySQL Server. Todos los bugs reportados y conocidos se arreglan en la última versión, con las excepciones listadas en las secciones de bugs y que están relacionados con problemas de diseño. Consulte Sección A.8, “Problemas conocidos en MySQL”.

El diseño de MySQL Server es multi capa, con módulos independientes. Algunos de los últimos módulos se listan a continuación con una indicación de lo bien testeados que están:

- Replicatión (Estable)

  Hay grandes grupos de servidores usando replicación en producción, con buenos resultados. Se trabaja para mejorar características de replicación en MySQL 5.x.

- `InnoDB` tablas (Estable)

  El motor de almacenamiento transaccional `InnoDB` es estable y usado en grandes sistemas de producción con alta carga de trabajo.

- `BDB` tablas (Estable)

  El código `Berkeley DB` es muy estable, pero todavía lo estamos mejorando con el interfaz del motor de almacenamiento transaccional `BDB` en MySQL Server.

- Búsquedas Full-text (Estable)

  Búsquedas Full-text es ámpliamente usada.

- `MyODBC` 3.51 (Estable)

  `MyODBC` 3.51 usa ODBC SDK 3.51 y es usado en sistemas de producción ámpliamente. Algunas cuestiones surgidas parecen ser cuestión de las aplicaciones que lo usan e independientes del controlador ODBC o la base de datos subyacente.

1.4.4. Dimensiones máximas de las tablas MySQL

En MySQL 5.0, usando el motor de almacenamiento `MyISAM`, el máximo tamaño de las tablas es de 65536 terabytes (256 ^ 7 - 1 bytes). Por lo tanto, el tamaño efectivo máximo para las bases de datos en MySQL usualmente los determinan los límites de tamaño de ficheros del sistema operativo, y no por límites internos de MySQL.

El motor de almacenamiento `InnoDB` mantiene las tablas en un espacio que puede ser creado a partir de varios ficheros. Esto permite que una tabla supere el tamaño máximo individual de un fichero. Este espacio puede incluir particiones de disco, lo que permite tablas extremadamente grandes. El tamaño máximo del espacio de tablas es 64TB.

La siguiente tabla lista algunos ejemplos de límites de tamaño de ficheros de sistemas operativos. Esto es sólo una burda guía y no pretende ser definitiva. Para la información más actual, asegúrese de consultar la documentación específica de su sistema operativo.

Sistema operativo	**Tamaño máximo de fichero**
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4	(usando sistema de ficheros ext3) 4TB
Solaris 9/10	16TB
Sistema de ficheros NetWare w/NSS	8TB

win32 w/ FAT/FAT32	2GB/4GB
win32 w/ NTFS	2TB (posiblemente mayor)
MacOS X w/ HFS+	2TB

En Linux 2.2, puede utilizar tablas `MyISAM` mayores de 2GB usando el parche para LFS (Large File Support) en el sistema de ficheros ext2. En Linux 2.4 y posteriores, existen parches para ReiserFS soportando grandes archivos (hasta 2TB). La mayoría de distribuciones Linux se basan en el kernel 2.4 o 2.6 e incluyen todos los parches LFS necesarios. Con JFS y XFS, se permiten ficheros mayores de un petabyte para Linux. Sin embargo, el tamaño máximo de ficheros todavía depende de diversos factores, uno de ellos siendo el sistema de ficheros usado para almacenar tablas MySQL.

Para un resumen más detallado acerca de LFS en Linux, recomendamos la página de Andreas Jaeger *Large File Support in Linux* en http://www.suse.de/~aj/linux_lfs.html.

Usuarios de Windows, por favor tengan en cuenta que: FAT and VFAT (FAT32) **no** se consideran apropiados para sistemas de producción con MySQL. Use NTFS para ello.

Por defecto, MySQL crea tablas `MyISAM` con una estructura interna que permite un tamaño máximo de unas 4GB. Puede chequear el tamaño máximo de tabla para una tabla con el comando `SHOW TABLE STATUS` o con `myisamchk -dv tbl_name`. Consulte Sección 13.5.4, “Sintaxis de `SHOW`”.

Si necesita una tabla `MyISAM` con un tamaño mayor a 4GB (y su sistema operativo soporta ficheros grandes), el comando `CREATE TABLE` permite las opciones `AVG_ROW_LENGTH` y `MAX_ROWS`. Consulte Sección 13.1.5, “Sintaxis de `CREATE TABLE`”. También puede cambiar esas opciones con `ALTER TABLE` una vez que la tabla se ha creado, para aumentar el tamaño máximo de la tabla. Consulte Sección 13.1.2, “Sintaxis de `ALTER TABLE`”.

Otros métodos para cambiar los límites de tamaño de ficheros para tablas `MyISAM` son:

- Si una tabla es de sólo lectura, puede usar `myisampack` para comprimirla. `myisampack` normalmente comprime una tabla al menos un 50%, lo que permite, a efectos prácticos, tablas mucho mayores. `myisampack` puede mezclar múltiples tablas en una misma tabla. Consulte Sección 8.2, “`myisampack`, el generador de tablas comprimidas de sólo lectura de MySQL”.
- MySQL incluye la biblioteca `MERGE` que permite tratar una colección de tablas `MyISAM` con una estructura idéntica en una tabla `MERGE`. Consulte Sección 14.2, “El motor de almacenamiento `MERGE`”.

1.4.5. Conformidad con el efecto 2000

MySQL Server por sí mismo no tiene problemas de conformidad con el año 2000 (Y2K):

- MySQL Server utiliza funciones de tiempo Unix que tratan las fechas hasta el año `2037` para valores `TIMESTAMP`. Para valores `DATE` y `DATETIME`, se aceptan fechas hasta el año `9999`.
- Todas las funciones de fecha MySQL se implementan en un mismo fichero fuente, `sql/time.cc`, y están programados cuidadosamente para no tener problemas con el año 2000.
- En MySQL 5.0 y posterior, el tipo de columna `YEAR` puede almacenar los años `0` y `1901` hasta `2155` en un byte y mostrarlo usando de dos a cuatro dígitos. Todos los años de dos dígitos se consideran en el rango `1970` hasta `2069`, lo que significa que si almacena `01` en una columna de tipo `YEAR`, MySQL Server lo trata como `2001`.

La siguiente demostración ilustra que MySQL Server no tiene problemas con valores `DATE` o `DATETIME` hasta el año 9999, ni tampoco tiene problemas con valores de tipo `TIMESTAMP` hasta el año 2030:

```
mysql> DROP TABLE IF EXISTS y2k;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE y2k (date DATE,
    ->                   date_time DATETIME,
    ->                   time_stamp TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO y2k VALUES
    -> ('1998-12-31','1998-12-31 23:59:59',19981231235959),
    -> ('1999-01-01','1999-01-01 00:00:00',19990101000000),
    -> ('1999-09-09','1999-09-09 23:59:59',19990909235959),
    -> ('2000-01-01','2000-01-01 00:00:00',20000101000000),
    -> ('2000-02-28','2000-02-28 00:00:00',20000228000000),
    -> ('2000-02-29','2000-02-29 00:00:00',20000229000000),
    -> ('2000-03-01','2000-03-01 00:00:00',20000301000000),
    -> ('2000-12-31','2000-12-31 23:59:59',20001231235959),
    -> ('2001-01-01','2001-01-01 00:00:00',20010101000000),
    -> ('2004-12-31','2004-12-31 23:59:59',20041231235959),
    -> ('2005-01-01','2005-01-01 00:00:00',20050101000000),
    -> ('2030-01-01','2030-01-01 00:00:00',20300101000000),
    -> ('2040-01-01','2040-01-01 00:00:00',20400101000000),
    -> ('9999-12-31','9999-12-31 23:59:59',99991231235959);
Query OK, 14 rows affected (0.01 sec)
Records: 14  Duplicates: 0  Warnings: 2

mysql> SELECT * FROM y2k;
+------------+---------------------+----------------+
| date       | date_time           | time_stamp     |
+------------+---------------------+----------------+
| 1998-12-31 | 1998-12-31 23:59:59 | 19981231235959 |
| 1999-01-01 | 1999-01-01 00:00:00 | 19990101000000 |
| 1999-09-09 | 1999-09-09 23:59:59 | 19990909235959 |
| 2000-01-01 | 2000-01-01 00:00:00 | 20000101000000 |
| 2000-02-28 | 2000-02-28 00:00:00 | 20000228000000 |
| 2000-02-29 | 2000-02-29 00:00:00 | 20000229000000 |
| 2000-03-01 | 2000-03-01 00:00:00 | 20000301000000 |
| 2000-12-31 | 2000-12-31 23:59:59 | 20001231235959 |
| 2001-01-01 | 2001-01-01 00:00:00 | 20010101000000 |
| 2004-12-31 | 2004-12-31 23:59:59 | 20041231235959 |
| 2005-01-01 | 2005-01-01 00:00:00 | 20050101000000 |
| 2030-01-01 | 2030-01-01 00:00:00 | 20300101000000 |
| 2040-01-01 | 2040-01-01 00:00:00 | 00000000000000 |
| 9999-12-31 | 9999-12-31 23:59:59 | 00000000000000 |
+------------+---------------------+----------------+
14 rows in set (0.00 sec)
```

Los dos últimos valores de columna `TIMESTAMP` son cero porque los valores de año (`2040`, `9999`) excede el máximo de `TIMESTAMP`. El tipo de datos `TIMESTAMP` que se usa para almacenar el tiempo actual, soporta valores del rango `19700101000000` hasta `20300101000000` en máquinas de 32-bit (valores con signo). En máquinas de 64-bit, `TIMESTAMP` trata valores hasta `2106` (valores sin signo).

Aunque MySQL Server por sí mismo no tiene problemas con el año 2000, puede tenerlos si interactúa con aplicaciones que sí los tengan. Por ejemplo, muchas aplicaciones antiguas almacenan o manipulan años usando valores de dos dígitos (que son ambíguos) en lugar de cuatro dígitos. Este problema puede darse por aplicaciones que usan valores tales como `00` o `99` como indicadores de valores "perdidos". Por desgracia, estos problemas pueden ser complicados de arreglar, ya que aplicaciones diferentes pueden haber sido programadas por distintos programadores, cada uno de los cuales puede usar una serie de distintas convenciones y funciones de fechas.

Así, aunque MySQL Server no tiene problemas con el año 2000, es la responsabilidad de la aplicación de proporcionar entradas inambíguas. Consulte Sección 11.3.4, “Efecto 2000 (Y2K) y tipos de datos” para las reglas de MySQL Server para tratar fechas ambíguas de dos digitos.

1.5. Mapa de desarrollo de MySQL

Esta sección proporciona un vistazo del plan de desarrollo de MySQL, incluyendo las principales características implementadas o planeadas para MySQL 4.0, 4.1, 5.0, y 5.1. La siguiente sección proporciona información para cada serie.

La actual serie en producción es MySQL 5.0, cuya versión estable es la 5.0.9, publicada en agosto del 2005. La serie de producción anterior es la MySQL 4.1, cuya versión estable es 4.1.7, publicada en octubre del 2004. Estatus de producción significa que el futuro del desarrollo 5.0 y 4.1. está limitado sólo a arreglar problemas. Para versiones anteriores a MySQL 4.0 y la serie 3.23, sólo se arreglan bugs críticos.

Desarrollo activo de MySQL actualmente tiene lugar en la serie MySQL 5.1, lo que significa que nuevas características se añaden a la misma.

Antes de actualizar de una serie a la siguiente, por favor consulte los comentarios en Sección 2.10, “Aumentar la versión de MySQL”.

Planes para las características más demandadas se resumen en la siguiente tabla.

Característica	Serie MySQL
Claves foráneas para tablas `MyISAM`	5.1 (ya implemantado para tablas `InnoDB`)
Disparadores	5.0 y 5.1
Full outer join	5.1
Restricciones de integridad	5.1

1.5.1. El servidor MySQL incrustado (embedded)

La biblioteca del servidor incrustado `libmysqld` permite MySQL Server pueda trabajar con una gran cantidad de dominios de aplicaciones. Usando esta biblioteca, los desarrolladores pueden añadir MySQL Server en varias aplicaciones y dispositivos electrónicos, donde el usuario final no tiene conocimiento que hay una base de datos subyacente. MySQL Server incrustado es ideal para uso tras aplicaciones en Internet, kioskos públicos, combinación de hardware/software en llaveros, servidores de alto rendimiento de Internet, bases de datos autocontenidas distribuidas en CD-ROM, y así.

Muchos usuarios de `libmysqld` se benefician de la licencia dual de MySQL. Para los que no quieran estar ligados a la licencia GPL, el software está disponible con licencia comercial. Consulte http://www.mysql.com/company/legal/licensing/ para más información de la política de licencias de MySQL AB. La biblioteca incrustada MySSQL usa la misma interfaz que la biblioteca cliente normal, por lo que es conveniente y fácil de usar. Consulte Sección 24.2.16, “libmysqld, la biblioteca del servidor MySQL incrustado (embedded)”.

En Windows hay dos bibliotecas diferentes:

`libmysqld.lib`	Biblioteca dinámica para aplicaciones threaded.
`mysqldemb.lib`	Biblioteca estático para aplicaciones no threaded.

1.5.2. Qué hay de nuevo en MySQL 5.0

Las siguientes características se implementan en MySQL 5.0.

- Tipo de datos **BIT**: Consulte Sección 11.2, “Tipos numéricos”.
- **Cursores**: Soporte elemental. Consulte Sección 19.2.11, “Cursores”.

- **Diccionario de datos (Information Schema)**: ConsulteCapítulo 22, *La base de datos de información `INFORMATION_SCHEMA`*.
- **Administrador de instancias**: Puede usarse para iniciar y parar el MySQL Server, incluso desde una máquina remota. Consulte Sección 5.2, “El gestor de instancias de MySQL”.
- **Matemáticas de precisión**: Consulte Capítulo 23, *Matemáticas de precisión*.
- **Procedimientos almacenados**: Consulte Capítulo 19, *Procedimientos almacenados y funciones*.
- **Modo estricto y tratamiento de errores estándar**: Consulte Sección 5.3.2, “El modo SQL del servidor” y Capítulo 26, *Manejo de errores en MySQL*.
- **Disparadores**: Consulte Capítulo 20, *Disparadores (triggers)*.
- Tipo de datos **VARCHAR**: Soporte nativo `VARCHAR`. La longitud máxima de `VARCHAR` es 65,532 bytes ahora, y no se cortan espacios en blanco consecutivos. Consulte Sección 11.4.1, “Los tipos `CHAR` y `VARCHAR`”.
- **Vistas**: Consulte Capítulo 21, *Vistas (Views)* y Sección 1.7.5.6, “Vistas”.

La sección Novedades de este manual incluye una lista más en profundidad de características. Consulte Sección C.1, “Cambios en la entrega 5.0.x (Desarrollo)”.

Para los que deseen consultar las últimas novedades de MySQL, tenemos nuestro repositorio BitKeeper para MySQL disponible públicamente. Consulte Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”.

1.6. Fuentes de información acerca de MySQL

1.6.1. Listas de correo de MySQL

Esta sección presenta las listas de correo MySQL y proporciona guías sobre cómo deben usarse las listas. Cuando se suscribe a una lista de correo, recibe todos los mensajes como mensajes electrónicos. Puede enviar sus propias preguntas y respuestas a la lista.

1.6.1.1. Las listas de correo de MySQL

Para suscribirse o borrarse de cualquiera de las listas descritas en esta sección, visite http://lists.mysql.com/. Para la mayoría de ellos, puede seleccionar la versión normal de la lista en la que recibe mensajes individuales, o una versión resumida en la que recibe un gran mensaje al día.

Por favor *no* envíe mensajes para suscribirse o borrarse a ninguna de las listas de correo, ya que dichos mensajes se distribuyen automáticamente a miles de usuarios..

Su sitio local puede tener muchos suscriptores a una lista de correo MySQL. En ese caso, puede tener una lista de correo local, de forma que los mensajes enviados de `lists.mysql.com` a su sitio se propagan a la lista local. En estos casos, por favor contacte con su administrador de sistemas para ser añadido o borrado de la lista MySQL local.

Si desea tener el tráfico de una lista de correo en un buzón de correo separado en su programa de correo, cree un filtro basado en las cabeceras del mensaje. Puede usar las cabeceras `List-ID:` o `Delivered-To:` para identificar los mensajes de la lista.

Las listas de correo de MySQL son las siguientes:

- `anuncios`

Esta lista es para anuncios de nuevas versiones de MySQL y programas relacionados. Esta es una lista de tráfico bajo y a la que todos los usuarios de MySQL deberían suscribirse.

- `mysql`

  Esta es la lista principal para discusión sobre MySQL en general. Por favor tenga en cuenta que algunas cuestiones es mejor discutirlas en listas más especializadas. Si postea en una lista equivocada, puede no obtener respuesta.

- `bugs`

  Esta lista es para gente que desee estar informada sobre cuestiones reportadas desde la última versión de mySQL o que deseen estar activamente implicadas en el proceso de buscar bugs y arreglarlos. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

- `temas internos`

  Esta lista es para gente que trabaja en el código de MySQL. Este también es el fórum para discutir acerca del desarrollo de MySQL y para publicar parches.

- `mysqldoc`

  Esta lista es para gente que trabaja en la documentación de MySQL: gente de MySQL AB, traductores, y otros miembros de la comunidad.

- `pruebas de rendimiento`

  Esta lista es para cualquiera interesado en temas de rendimiento. La discusión se concentra en rendimiento de bases de datos (no sólo de MySQL), pero también incluye categorías más amplias como rendimiento del kernel, sistema de ficheros, tipos de discos, etc.

- `empaquetadores`

  Esta lista es para discusiones acerca de empaquetar y distribuir MySQL. Este es el fórum usado por mantenedores de distribuciones para intercambiar ideas sobre empaquetar MySQL y asegurar que MySQL parece tan similar como sea posible en todas las plataformas y sistemas operativos soportados.

- `java`

  Esta lista es para discusionse acerca de MySQL Server y Java. Normalmente se usa para discutir acerca de JDBC, incluyendo el connector/J de MySQL.

- `win32`

  Esta lista es para todos los temas acerca del software MySQL en sistemas operativos Microsoft, tales como Windows 9x, Me, NT, 2000, SP y 2003..

- `myodbc`

  Esta lista es para todos los tópicos acerca de conectar al MySQL Server con ODBC.

- `herramientas gui`

  Esta lista es para todos los temas acerca de herramientas GUI MySQL, incluyendo `MySQL Administrator` y el cliente gráfico `MySQL Control Center`.

- `cluster`

Esta lista es para discusión acerca de MySQL Cluster.

- `dotnet`

  Esta lista es para discusión acerca de MySQL Server y la plataforma .NET. La mayoría de discusiones es acerca del Connector/NET MySQL.

- `plusplus`

  Esta lista es para tópicos acerca de programación con la API C++ para MySQL.

- `perl`

  Esta lista es para tópicos acerca de soporte Perl para MySQL con `DBD::mysql`.

Si no es capaz de obtener una respuesta a su pregunta de ninguna lista MySQL, una opción es adquirir soporte de MySQL AB. Esto le pondrá en contacto directo con los desarrolladores.

La siguiente tabla muestra algunas listas de correo MySQL en idiomas distintos al inglés. Estas listas no son operadas por MySQL AB.

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

  Lista de correo francesa.

- `<list@tinc.net>`

  Lista de correo Koreana. Envíe un correo a `subscribe mysql your@email.address` para subscribirse a la lista.

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

  Lista de correo Alemana. Envíe un correo a `subscribe mysql-de your@email.address` para subscribirse a la lista. Puede encontrar información acerca de esta lista en http://www.4t2.com/mysql/.

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

  Lista de correo Portuguesa. Envíe un correo a `subscribe mysql-br your@email.address` para subscribirse a la lista.

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

  Lista de correo Española. Envíe un correo a `subscribe mysql your@email.address` para subscribirse a la lista.

1.6.1.2. Hacer preguntas y reportar bugs

Antes de reportar un bug o cuestión, por favor haga lo siguiente:

- Busque en el manual en línea en http://dev.mysql.com/doc/. Intentamos mantener el manual actualizado añadiendo soluciones a nuevos problemas frecuentemente. El historial de cambios (http://dev.mysql.com/doc/mysql/en/News.html) puede ser particularmente útil ya que es bastante posible que versiones más actuales tengan soluciones a su problema.

- Busque en la base de datos de bugs en http://bugs.mysql.com/ para ver si el bug ha sido reportado y solucionado.

- Busque el archivo de las listas de correo MySQL en http://lists.mysql.com/.

- También puede usar http://www.mysql.com/search/ para buscar en todas las páginas web (incluyendo el manual) que se encuentran en la web de MySQL AB.

Si no puede encontrar una solución en el manual o los archivos, pruebe con su experto local en MySQL. Si tampoco puede encontrar una solución a su pregunta, por favor siga las guías para enviar un correo a las listas de correo MySQL, explicado en la siguiente sección, antes de contactar con nosotros.

1.6.1.3. Cómo informar de bugs y problemas

El sitio normal en el que reportar bugs es http://bugs.mysql.com/, que es la dirección de nuestra base de datos de bugs. Esta base de datos es pública, y puede ser consultada por cualquiera. Si entra en el sistema, puede añadir nuevos reportes.

Para escribir un buen reporte de error se necesita paciencia, pero hacerlo correctamente por primera vez nos ahorra tiempo tanto a nosotros como a usted mismo. Un buen reporte de bug que contenga un testeo completo del bug, hace que sea muy probable que se arregle para la siguiente versión. Esta sección muestra cómo escribir un reporte correctamente de forma que no pierda su tiempo haciendo cosas que no nos ayudan en absoluto.

Animamos a todo el mundo a usar el script `mysqlbug` para generar un reporte de bug (o un reporte acerca de cualquier problema). `mysqlbug` puede encontrarse en el directorio `scripts` (distribución fuente) y en el directorio `bin` en el directorio de instalación (distribución binaria). Si no es posible usar `mysqlbug` (por ejemplo, si utiliza Windows), es vital que incluya toda la información necesaria que aparece en esta sección (y lo más importante, una descripción del sistema operativo y la versión de MySQL).

El script `mysqlbug` le ayuda a generar un reporte determinando la mayoría de la información automáticamente, pero si falta algo importante, por favor inclúyalo en su mensaje. Por favor, lea esta sección con cuidado y asegúrese que toda la información descrita aquí se incluye en su reporte.

Preferiblemente, debe testear el problema usando la última versión de producción o desarrollo de MySQL server antes de postear. Cualquiera debería ser capaz de repetir el bug usando `mysql test< script_file` en el caso de test incluído o ejecutando el script de consola o Perl incluído en el reporte de bug.

Todos los bugs posteados en la base de datos de bugs en http://bugs.mysql.com/ se corrigen o documentan en la siguiente actualización de MySQL. Si sólo se necesitan cambios menores en el código para arreglarlo, podemos postear un parche para arreglarlo.

Si encuentra un fallo importante de seguridad en MySQL, puede enviar un correo a `<security@mysql.com>`.

Si tiene un reporte de bug repetible, por favor envíelo a la base de datos de bugs en http://bugs.mysql.com/. Tenga en cuenta que incluso en este caso es bueno ejecutar el script `mysqlbug` antes para reunir información de su sistema. Cualquier bug que seamos capaces de reproducir tiene una alta probabilidad de arreglarse en la siguiente versión de MySQL.

Para reportar otros problemas, puede usar cualquiera de las listas de correo de MySQL.

Recuerde que nos es posible responder un mensaje que contenga demasiada información, pero no si no contiene suficiente. Normalmente se omiten los hechos porque se piensa que se conoce la causa del problema y se asume que algunos detalles no importan. Un buen principio es el siguiente: si duda acerca de explicar algo, hágalo. Es más rápido y menos problemático escribir un par de líneas extra en su reporte que esperar a la respuesta si debemos preguntar algo que no se incluya en el reporte inicial.

Los errores más comunes en los reportes de error son (a) no incluir el número de versión de la distribución MySQL Server usada, y (b) no escribir completamente la plataforma en la está instalado MySQL Server

(incluyendo el tipo de plataforma y número de versión). Esta es información altamente relevante, y en el 99% de los casos el reporte de bug es inútil sin ella. Muy a menudo nos preguntas "¿Porqué no me funciona a mí?" Entonces encontramos que la característica reportada no estaba implementada en esa versión de MySQL. A veces el error depende de la plataforma; en esos casos es casi imposible para nosotros arreglar nada sin saber el sistema operativo y el número de versión de la plataforma.

Si ha compilado MySQL del código fuente, recuerde en proporcionar información acerca del compilador, si está relacionada con el problema. A menudo la gente encuentra fallos en los compiladores y cree que el problema está relacionado con MySQL. La mayoría de los compiladores están bajo desarrollo contínuo y mejoran versión a versión, necesitamos saber qué compilador usa. Tenga en cuenta que cada cada problema de compilación debe ser considerado como un bug y reportado como tal.

Es más útil cuando se incluye una buena descripción del problema junto al reporte del bug. Esto es dar un buen ejemplo de todo lo que conduce al problema y describir exactamente el problema en sí. El mejor reporte es el que incluye un ejemplo incluyendo cómo reproducir el problema o bug. Consulte Sección D.1.6, “Crear un caso de prueba tras haber encontrado una tabla corrupta”.

Si un programa produce un mensaje de error es muy importante incluirlo en el reporte. Si tratamos de buscar algo de los archivos usando programas, es mejor que el mensaje de error coincida exactamente con el producido por el programa (incluso es importante respetar mayúsculas y minúsculas). Nunca debe intentar reproducir de memoria el mensaje de error; en lugar de ello, copie el mensaje entero en el reporte.

Si tiene algún problema con el Connector/ODBC (MyODBC), por favor trate de generar un fichero de traza y enviarlo con su reporte. Consulte Sección 25.1.7.2, “How to Report Connector/ODBC Problems or Bugs”.

Por favor, recuerde que mucha gente que lea su reporte lo hará con un monitor de 80 columnas. Cuando genere reportes o ejemplos usando la columna de línea de comandos `mysql` debe usar la opción `--vertical` (o el terminador de comando `\G`) para salida que excedería el ancho disponible para tales monitores (por ejemplo, con el comando `EXPLAIN SELECT`; (vea el ejemplo al final de esta sección).

Por favor, incluya la siguiente información en su reporte:

- El número de la versión de la distribución de MySQL que usa (por ejemplo, MySQL 4.0.12). Puede consultar la versión que está usando ejecuntado `mysqladmin version`. El programa `mysqladmin` puede encontrarse en el directorio `bin` bajo el directorio de instalación.
- El fabricante y modelo de la máquina en la que experimenta el problema.
- Nombre del sistema operativo y versión. Si trabaja con Windows, puede obtener el nombre y número de versión haciendo doble click en el icono Mi PC y consultando el menú "Ayuda/Acerca de Windows". Para la mayoría de sistemas Unix puede obtener esta información con el comando `uname -a`.
- En ocasiones la cantidad de memoria (real y virtual) es relevante. Si lo duda, incluya estos valores.
- Si usa una distribución fuente del software MySQL, el nombre y número de versión del compilador usado es necesario.Si usa una distribución binaria, necesita el nombre de la distribución.
- Si el problema ocurre durante la compilación, incluya el mensaje de error exacto y unas cuantas líneas de contexto alrededor del código en el fichero donde ocurre el error.
- Si `mysqld` cae, incluya la consulta que hizo caer `mysqld`. Normalmente puede consultarlo ejecutando `mysqld` con el log de consultas activado, y luego consultando el log tras la caída de `mysqld`. Consulte Sección D.1.5, “El uso de registros (logs) para encontrar la causa de errores de `mysqld`”.
- Si una base de datos está relacionada con el problema, incluya la salida del comando `mysqldump --no-data db_name tbl_name`. Este es un método sencillo y poderoso para obtener información

acerca de cualquier tabla en una base de datos. La información nos ayuda a crear una situación similar a la que ha provocado el fallo.

- Para bugs relacionados con rendimiento o problemas con consultas `SELECT`, siempre debe incluir la salida de `EXPLAIN SELECT ...`, y como mínimo el número de filas que el comando `SELECT` produce. También puede incluir la salida de `SHOW CREATE TABLE tbl_name` para cada tabla implicada. Mientras más información tengamos acerca de la situación, es más posible que podamos ayudar..

  El siguiente es un ejemplo de un reporte muy bueno. Debe ser posteado con el script `mysqlbug`. El ejemplo usa la herramienta por líneas de comando `mysql`. Tenga en cuenta el terminador de comandos `\G` para comandos cuya salida exceda las de un monitor de 80 columnas.

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
         <salida de SHOW COLUMNS>
  mysql> EXPLAIN SELECT ...\G
         <salida de EXPLAIN>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
         <Una pequeña descripción de la salida del SELECT,
         incluyendo el tiempo empleado en ejecutar la consulta>
  mysql> SHOW STATUS;
         <salida de SHOW STATUS>
  ```

- Si ocurre un problema al ejecutar `mysqld`, trate de proporcionar un script de entrada que reproduzca la anomalía. Este script debe incluir cualquier fichero fuente necesario. Mientras más fielmente reproduzca el script la situación, mejor. Si puede hacer un caso de test reproducible, debe postearlo en http://bugs.mysql.com/ para un tratamiento prioritario.

  Si no puede proporcionar un script, debe como mínimo incluir la salida de `mysqladmin variables extended-status processlist` en su correo para proporcionar algo de información sobre cómo se está comportando el sistema.

- Si no puede proporcionar un caso de test con unas pocas líneas, o si la tabla de test es demasiado grande para ser enviada a la lista de correo (más de 10 filas), debe dumpear sus tablas usando `mysqldump` y crear un ficero `README` que describa su problema.

  Cree un fichero comprimido con sus ficheros usando `tar` y `gzip` o `zip`, y use FTP para transferir el archivo a ftp://ftp.mysql.com/pub/mysql/upload/. Después introduzca el problema en nuestra base de datos en http://bugs.mysql.com/.

- Si cree que el servidor MySQL produce un resultado extraño de una consulta, incluya no sólo el resultado, sino también su opinión sobre cuál debería ser el resultado correcto, y una citación describiendo las bases de su opinión.

- Cuando de un ejemplo del problema, es mejor usar los nombres de variables, de tablas, etc. que existan en la situación en lugar de usar nuevos nombres. El problema puede estar relacionado con el nombre de una variable o tabla. Estos casos son raros, pero es mejor estar seguro que arrepentirse luego. Después de todo, debería ser más fácil dar un ejemplo que use la situación real y esto es mejor para nosotros. En caso que tenga datos que no quiera enseñar, puede usar FTP para transferirlo a ftp://ftp.mysql.com/pub/mysql/upload/. Si la información es realmente secreta y no quiere enseñárnosla, entonces puede proporcionarnos un ejemplo usando otros nombres, pero por favor, considérelo como la última opción.

- Incluya todas las opciones introducidas en los programas relevantes, si es posible. Por ejemplo, indique las opiones que usa cuando inicia el servidor `mysqld` así como las opciones que usa cuando ejecuta cualquier programa cliente MySQL. Las opciones de dichos programas, tales como `mysqld` y `mysql`, y al script `configure` , son a menudo claves para obtener una respuesta y muy relevantes. Nunca

es mala idea incluirlas. Si usa cualquier módulo, como Perl o PHP, por favor incluya los número de versiones de los mismos también.

- Si su pregunta está relacionada con el sistema de privilegios, por favor incluya la salida de `mysqlaccess`, la salida de `mysqladmin reload`, y todos los mensajes de error que obtenga cuando intente conectar. Cuando testee sus privilegios, debe ejecutar el comando `mysqlaccess`. Después, ejecute `mysqladmin reload version` y trate de conectar con el comando que le causa problemas. `mysqlaccess` se encuentra en el directorio `bin` bajo el directorio de instalación de MySQL.

- Si tiene un parche para un bug, inclúyalo. Pero no asuma que el parche es todo lo que necesitamos o que podemos usarlo, sin proporcionar alguna información necesaria como casos de test mostrando el problema que corrige el parche. Podemos encontrar problemas en su parche o puede que no lo entendamos en absoluto; en ese caso no podemos usarlo.

  Si no podemos verificar exactamente el propósito del parche, no lo usaremos. Los casos de test nos ayudan en este punto. Nos muestran que el parche puede tratar todas las situaciones que puedan ocurrir. Si encontramos un caso extremo (incluso uno raro) donde el parche no funcione, será inútil.

- Suposiciones acerca de la naturaleza del bug, porqué ocurre o de qué depende, suelen ser incorrectas. Incluso el equipo de MySQL no puede adivinar estas cosas sin usar un debugger para determinar la causa real de un bug.

- Indique en su reporte que ha chequeado el manual de referencia y el archivo de correo, de forma que otros sepan que ha intentado solucionar el problema por sí mismo.

- Si obtiene un `parse error`, por favor chequee su sintaxis con cuidado. Si no puede encontrar nada incorrecto en ella, es muy posible que su versión de MySQL Server no soporte la sintaxis que utiliza. Si está usando la versión actual y el manual en http://dev.mysql.com/doc/ no cubre la versión que usa, MySQL Server no soporta su consulta. En ese caso, sus únicas opciones son implementar la sintaxis usted mismo o enviar un mail a `<licensing@mysql.com>` y pedir una oferta para implementarlo.

  Si el manual cubre la sintaxis que está usando, pero tiene una versión más antigua de MySQL, compruebe el historial de cambios de MySQL para ver si la sintaxis ha sido implementada. En ese caso, tiene la opción de actualizar a una nueva versión de MySQL Server. Consulte Apéndice C, *Historial de cambios de MySQL*.

- Si su problema es que los datos parecen estar corruptos u obtiene errores al acceder a una tabla en particular, debe chequear y tratar de arreglar las tablas con `CHECK TABLE` y `REPAIR TABLE` o con `myisamchk`. Consulte Capítulo 5, *Administración de bases de datos*.

  Si está utilizando Windows, verifique que `lower_case_table_names` es 1 o 2 con `SHOW VARIABLES LIKE 'lower_case_table_names'`.

- Si tiene problemas con tablas corruptas a menudo, debe tratar de encontrar cuándo y porqué ocurre. En este caso, el log de errores en el directorio de datos de MySQL puede contener información acerca de qué ha ocurrido. (Éste es el fichero con el sufijo `.err` en el nombre.) Consulte Sección 5.10.1, “El registro de errroes (Error Log)”. Por favor, incluya cualquier información relevante de este fichero en su reporte. Normalmente `mysqld` *nunca* debería corromper una tabla si nada muere durante una actualización. Si puede encontrar la causa de un `mysqld` muriendo, es mucho más fácil para nosotros encontrar una solución al problema. Consulte Sección A.1, “Cómo determinar a qué es debido un problema”.

- Si es posible, descargue e instale la versión más reciente de MySQL Server y compruebe si resuelve su problema. Todas las versiones del software MySQL son testeadas duramente y deberían funcionar sin problemas. Creemos en hacer todo tan compatible con versiones anteriores como sea posible, y debería cambiar entre versiones de MySQL sin dificultades. Consulte Sección 2.1.2, “Escoger la distribución MySQL a instalar”.

Si es un cliente con soporte, por favor envíe el reporte de error a <mysql-support@mysql.com> para tratamiento de alta prioridad, así como a la lista de correo apropiada para ver si alguien más ha experimentado (y quizás resuelto) el problema.

Para información acerca de reportar errores en MyODBC, consulte Sección 25.1.7.2, “How to Report Connector/ODBC Problems or Bugs”.

Para soluciones de problemas más comunes, consulte Apéndice A, *Problemas y errores comunes*.

Cuando se envían soluciones a usted individualmente y no a la lista de correo, se considera una buena práctica resumir las respuestas y enviar el resumen a la lista de correo, de forma que otros puedan beneficiarse de las respuestas que ha recibido y que le han ayudado a resolver su problema.

1.6.1.4. Guía para la contestación de preguntas en las listas de correo

Si considera que su respuesta tiene interés general, puede postearla en la lista de correo en lugar de responder directamente al indivíduo que la preguntó. Trate de hacer su respuesta general para que otras personas a parte de quién hizo la pregunta, se puedan beneficiar con la respuesta otros usuarios. Cuando postee a la lista, aségurese que su respuesta no es una duplicación de otra respuesta.

Trate de resumir la parte esencial de la pregunta en su respuesta; no se sienta obligado a anotar el mensaje original entero.

Por favor, no postee un mensaje desde su navegador con el modo HTML activado. Muchos usuarios no leen el correo con un navegador

1.6.2. Soporte por IRC (Internet Relay Chat) de la comunidad MySQL

Adicionalmente a las listas de correo MySQL, puede encontrar una comunidad experimentada en `IRC` (`Internet Relay Chat`). Estos son los mejores canales que conocemos hoy día:

- **freenode** (consulte http://www.freenode.net/ para servidores)
  - `#mysql` Básicamentes preguntas sobre MySQL , pero también de otras bases de datos y preguntas generales sobre SQL. Preguntas sobre PHP, Perl o C en combinación con MySQL son frecuentes.

Si busca un cliente IRC para conectar a un canal IRC, consulte `X-Chat` (http://www.xchat.org/). X-Chat (GPL licensed) está disponible para Unix así como para Windows (una implementación libre para Windows sobre X-Chat está disponible en http://www.silverex.org/download/).

1.6.3. Soporte por parte de la comunidad en los foros de MySQL

El último recurso de soporte para la comunidad son los foros en http://forums.mysql.com.

Hay una variedad de foros disponibles, agrupados en las siguientes categorías generales:

- Migración
- Uso de MySQL
- Conectores MySQL
- Tecnología MySQL
- Negocios

1.7. Cumplimiento de los estándares por parte de MySQL

Esta sección describe cómo MySQL se relaciona con el estándar ANSI/ISO SQL. MySQL Server tiene varias extensiones del estándar SQL, y aquí puede encontrar cuáles son y cómo usarlas. También puede encontrar información sobre lagunas de funcionalidades en MySQL Server, y cómo tratar algunas diferencias.

El estándar SQL ha ido evolucionando desde 1986 y existen varias versiones. En este manual, "SQL-92" se refiere al estándar publicado en 1992, "SQL:1999" se refiere al estándar publicado en 1999, y "SQL:2003" se refiere a la versión actual del estándar. Usamos la frase "el estándar SQL" para referirnos a la versión actual del estándar SQL en cualquier momento.

Nuestro objetivo es no restringir la usabilidad de MySQL ningún uso sin una muy buena razón para ello. Incluso si no tenemos los recursos para hacer un desarrollo para cada uso posible, estamos siempre deseando ayudar y ofrecer sugerencias a gente que intenta usar MySQL en nuevos campos.

Uno de nuestros fines principales con el producto es continuar el trabajo hacia el cumplimiento del estándar SQL, pero sin sacrificar velocidad o fiabilidad. No tememos añadir extensiones a SQL o soporte para funcionalidades no SQL si esto aumenta la usabilidad de MySQL Server para un gran segmento de nuestra base de usuarios. La interfaz `HANDLER` en MySQL Server 4.0 es un ejemplo de esta estrategia. Consulte Sección 13.2.3, “Sintaxis de HANDLER”.

Continuamos soportando bases de datos transaccionales y no transaccionales para satisfacer uso crítico 24/7 y uso pesado en entornos Web o log.

MySQL Server fue diseñado originalmente para trabajar con bases de datos de tamaño medio (de 10 a 100 millones de regitros, o unas 100MB por tabla) en máquinas pequeñas. Hoy MySQL Server soporta bases de datos de tamaño de terabytes, pero el código todavía puede compilarse en una versión reducida adecuada para dispositivos hand-held o incrustados. El diseño compacto de MySQL Server hace el desarrollo en ambas direcciones posible sin ningún conflicto en el árbol fuente.

Actualmente, no tratamos soporte en tiempo real, aunque la capacidad de replicación en MySQL ofrece funcionalidades significativas.

Existe soporte para clusters de bases de datos a través de soluciones de terceras partes, así como la integración de tecnología NDB Cluster, disponible desde la versión 4.1.2. Consulte Capítulo 16, *MySQL Cluster*.

También estamos mirando de proveer de soporte XML en el servidor de base de datos.

1.7.1. Estándares utilizados por MySQL

Estamos intentando soportar en estándar ANSI/ISO completamente, pero sin hacer concesiones a la velocidad y calidad del código.

Niveles ODBC 0-3.51.

1.7.2. Selección de modos SQL

MySQL Server puede operar en distintos modos SQL y puede aplicar dichos modos de forma diferente para distintos clientes. Esto permite a una aplicación adaptar el funcionamiento del servidor a sus propios requerimientos.

Los modos definen la sintaxis que MySQL debe soportar y qué clase de validaciones debe efectuar a los datos. Esto hace más fácil usar MySQL en un conjunto de entornos diferentes y usar MySQL junto con otros servidores de bases de datos.

Puede inicializar el modo SQL por defecto inicializando `mysqld` con la opición `--sql-mode="modes"`. Empezando en MySQL 4.1., se puede cambiar el modo tras inicializar mediante la variable `sql_mode` con un comando `SET [SESSION|GLOBAL] sql_mode='modes'`.

Para más información acerca de especificar el modo del servidor, consulte Sección 5.3.2, “El modo SQL del servidor”.

1.7.3. Ejecutar MySQL en modo ANSI

Puede decirle a `mysqld` que use el modo ANSI con la opción `--ansi` al arrancar. Consulte Sección 5.3.1, “Opciones del comando `mysqld`”.

Ejecutar el servidor en modo ANSI es lo mismo que inicializarlo con las siguientes opciones (especifique el valor de `--sql_mode` en una única línea):

```
--transaction-isolation=SERIALIZABLE
--sql-mode=REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,
IGNORE_SPACE
```

En MySQL 4.1, puede conseguir el mismo efecto con los siguientes 2 comandos (especifique el valor de `sql_mode` en una única línea):

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,
IGNORE_SPACE';
```

Consulte Sección 1.7.2, “Selección de modos SQL”.

En MySQL 4.1.1, la opción `sql_mode` puede inicializarse con el siguiente comando:

```
SET GLOBAL sql_mode='ansi';
```

En ese caso, el valor de la variable `sql_mode` se especifica para todas las opciones que son relevantes en el modo ANSI. Puede comprobar el resultado de la siguiente manera:

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

1.7.4. Extensiones MySQL al estándar SQL

MySQL Server incluye algunas extensiones que probablemente no encontrará en otras bases de datos SQL. Tenga en cuenta que si lo usa, su código no será portable a otros servidores SQL. En algunos casos, puede escribir código que incluya extensiones MySQL, pero siendo portable, mediante comentarios de la forma `/*! ... */`. En ese caso, MySQL parsea y ejecuta el código dentro de los comentarios como si fuera cualquier otro comando de MySQL, pero otros servidores SQL ignorarán la extensión. Por ejemplo:

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

Si añade un número de versión tras el carácter '`!`', la sintaxis dentro del comentario se ejecuta sólo si el número de versión de MySQL es igual o mayor que el especificado:

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

Eso significa que si tiene la Versión 3.23.02 o posterior, MySQL Server usa la palabra clave `TEMPORARY`.

La siguiente lista describe las extensiones MySQL, organizadas por categorías.

- Organización de los datos en disco

  MySQL Server mapea cada base de datos a un directorio bajo el directorio de datos de MySQL, y las tablas dentro de cada directorio como ficheros. Esto tiene algunas implicaciones:

  -

  - Puede usar comandos de sistema estándar para hacer copia de seguridad, renombrar, mover, borrar y copiar tablas de tipo `MyISAM` o `ISAM`. Por ejemplo, para renombrar el nombre de una tabla `MyISAM` renombre los archivos `.MYD`, `.MYI`, y `.frm` que correspondan a la tabla.

  Nombres de bases de datos, tablas, índices, columnas o alias pueden empezar con un dígito (pero no pueden consistir únicamente de dígitos).

- Sintaxis general del lenguaje

  - Cadenas de caracteres deben limitarse por '"' o '`'`, no sólo por '`'`'.

  - Use '`\`' como un carácter de escape en cadenas de caracteres.

  - En comandos SQL, puede acceder a tablas de distintas bases de datos con la sintaxis *`db_name.tbl_name`*. Algunos servidores SQL proporcionan la misma fucnionalidad, pero lo llaman `User space`. MySQL Server no soporta espacios de tablas como los usados en comandos como: `CREATE TABLE ralph.my_table...IN my_tablespace`.

- Sintaxis de comandos SQL

  - Los comandos `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, y `REPAIR TABLE`.

  - Los comandos `CREATE DATABASE` y `DROP DATABASE`. Consulte Sección 13.1.3, “Sintaxis de `CREATE DATABASE`”.

  - El comando `DO`.

  - `EXPLAIN SELECT` para obtener una descripción de cómo las tablas se usan.

  - Los comandos `FLUSH` y `RESET`.

  - El comando `SET`. Consulte Sección 13.5.3, “Sintaxis de `SET`”.

  - El comando `SHOW`. Consulte Sección 13.5.4, “Sintaxis de `SHOW`”.

  - Uso de `LOAD DATA INFILE`. En muchos casos, esta sintaxis es compatible con el comando de Oracle `LOAD DATA INFILE`. Consulte Sección 13.2.5, “Sintaxis de `LOAD DATA INFILE`”.

  - Uso de `RENAME TABLE`. Consulte Sección 13.1.9, “Sintaxis de `RENAME TABLE`”.

  - Uso de `REPLACE` en lugar de `DELETE` + `INSERT`. Consulte Sección 13.2.6, “Sintaxis de `REPLACE`”.

  - Uso de `CHANGE col_name`, `DROP col_name`, o `DROP INDEX`, `IGNORE` o `RENAME` en un comando `ALTER TABLE`. Uso de múltiples `ADD`, `ALTER`, `DROP`, o `CHANGE` cláusulas en un comando `ALTER TABLE`. Consulte Sección 13.1.2, “Sintaxis de `ALTER TABLE`”.

  - Uso de nombres de índices, índices sobre el prefijo de un cambio, y uso de `INDEX` o `KEY` en un comando `CREATE TABLE`. Consulte Sección 13.1.5, “Sintaxis de `CREATE TABLE`”.

  - Uso de `TEMPORARY` o `IF NOT EXISTS` con `CREATE TABLE`.

  - Uso de `IF EXISTS` con `DROP TABLE`.

  - Puede borrar varias tablas con un único comando `DROP TABLE`.
  - Las cláusulas `ORDER BY` y `LIMIT` de los comandos `UPDATE` y `DELETE`.
  - Sintaxis de `INSERT INTO ... SET col_name = ...`.
  - La cláusula `DELAYED` de los comandos `INSERT` y `REPLACE`.
  - La cláusula `LOW_PRIORITY` de los comandos `INSERT`, `REPLACE`, `DELETE`, y `UPDATE`.
  - Uso de `INTO OUTFILE` y `STRAIGHT_JOIN` en un comando `SELECT` . Consulte Sección 13.2.7, “Sintaxis de SELECT”.
  - La opción `SQL_SMALL_RESULT` en un comando `SELECT`.
  - No necesita nombrar todas las columnas seleccionadas en la parte `GROUP BY`. Esto proporciona mejor rendimiento en algunas consultas muy específicas pero bastante normales. Consulte Sección 12.10, “Funciones y modificadores para cláusulas GROUP BY”.
  - Puede especificar `ASC` y `DESC` con `GROUP BY`.
  - La habilidad para inicializar variables en un comando con el operador `:=`:

```
mysql> SELECT @a:=SUM(total),@b=COUNT(*),@a/@b AS avg
    -> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

- Tipos de columnas
  - Los tipos de columnas `MEDIUMINT`, `SET`, `ENUM`, y los distintos tipos `BLOB` y `TEXT`.
  - Los atributos de columnas `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, y `ZEROFILL`.
- Funciones y operadores
  - Para facilitar a los usuarios que vienen de otros entornos SQL, MySQL Server soporta alias para varias funciones. Por ejemplo, todas las funciones de cadenas de caracteres soportan sintaxis estándar SQL y ODBC.
  - MySQL Server entiende los operadores `||` y `&&` para OR lógica y AND, como en el lenguaje de programación C. En MySQL Server, `||` y `OR` son sinónimos, como lo son `&&` y `AND`. Debido a esta sintaxis, MySQL Server no soporta el operador estándar SQL `||` para concatenar cadenas de caracteres; use en su lugar `CONCAT()`. Como `CONCAT()` toma cualquier número de argumentos, es fácil adaptarse al uso del operador `||` a MySQL Server.
  - Uso de `COUNT(DISTINCT list)` donde `list` tiene más de un elemento.
  - Todas las comparaciones de cadenas de caracteres son case-insensitive por defecto, con la ordenación determinada por el conjunto de caracteres actual (ISO-8859-1 Latin1 por defecto). Si no quiere que sea así, puede declarar las columnas con el atributo `BINARY` o usar la conversión `BINARY`, que hace que las comparaciones se hagan usando el código de caracteres subyacente en lugar del orden léxico.
  - El operador `%` es sinónimo de `MOD()`. Esto es que `N % M` es equivalente a `MOD(N,M)`. `%` se soporta para programadores C y por compatibilidad con PostgreSQL.

- Los operadores `=`, `<>`, `<=` ,`<`, `>=`,`>`, `<<`, `>>`, `<=>`, `AND`, `OR`, o `LIKE` se pueden usar en comparaciones de columnas a la izquierda del `FROM` en comandos `SELECT`. Por ejemplo:

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

- La función `LAST_INSERT_ID()` retorna el valor `AUTO_INCREMENT` más reciente. Consulte Sección 12.9.3, “Funciones de información”.
- `LIKE` se permite en columnas numéricas.
- Los operadores de expresiones regulares extendidos `REGEXP` y `NOT REGEXP`.
- `CONCAT()` o `CHAR()` con un argumento o más de dos argumentos. (En MySQL Server, estas funciones pueden tomar cualquier número de argumentos.)
- Las funciones `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `MD5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()`, y `WEEKDAY()`.
- Uso de `TRIM()` para eliminar espacios en substrings. Funciones estándar sólo SQL soportan eliminar caracteres simples.
- Las funciones `GROUP BY STD()`, `BIT_OR()`, `BIT_AND()`, `BIT_XOR()`, y `GROUP_CONCAT()`. Consulte Sección 12.10, “Funciones y modificadores para cláusulas `GROUP BY`”.

1.7.5. Diferencias en MySQL del estándar SQL

Intentamos que MySQL Server siga los estándares ANSI SQL y el estándar ODBC SQL, pero MySQL Server ejecuta operaciones de forma distinta en algunos casos:

- Para columnas `VARCHAR`, los espacios finales se eliminan cuando el valor se guarda. (Arreglado en MySQL 5.0.3). Consulte Sección A.8, “Problemas conocidos en MySQL”.
- En algunos casos, las columnas de tipo `CHAR` se convierten en columnas `VARCHAR` cuando define una tabla o altera su estructura. (Arreglado en MySQL 5.0.3). Consulte Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”.
- Los privilegios para una tabla no se eliminan automáticamente cuando se borra una tabla. Debe usar explícitamente un comando `REVOKE` para quitar los privilegios de una tabla. Consulte Sección 13.5.1.3, “Sintaxis de `GRANT` y `REVOKE`”.
- La función `CAST()` no soporta conversión a `REAL` o `BIGINT`. Consulte Sección 12.8, “Funciones y operadores de cast”.
- SQL estándar necesita que las cláusulas `HAVING` en un comando `SELECT` puedan referirse a columnas en la cláusula `GROUP BY`. Esto no se permite antes de la versión MySQL 5.0.2.

1.7.5.1. Subconsultas

MySQL 4.1 soporta sub-consultas y tablas derivadas. Una "sub-consulta" es un comando `SELECT` anidado en otro comando. Una tabla "derivada" (una vista sin nombre) es una subconsulta en la cláusula `FROM` de otra consulta. Consulte Sección 13.2.8, “Sintaxis de subconsultas”.

Para versiones MySQL anteriores a la 4.1, la mayoría de subconsultas pueden reescribirse usando joins u otros métodos. Consulte Sección 13.2.8.11, “Re-escribir subconsultas como joins en versiones de MySQL anteriores” para ejemplos que muestren cómo hacerlo.

1.7.5.2. `SELECT INTO TABLE`

MySQL Server no soporta la sintaxis de extensiones Sybase SQL: `SELECT ... INTO TABLE ...`. En su lugar, MySQL Server soporta la sintaxis estándar SQL `INSERT INTO ... SELECT ...`, que básicamente es lo mismo. Consulte Sección 13.2.4.1, “Sintaxis de `INSERT ... SELECT`”.

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

Alternativamente, puede usar `SELECT INTO OUTFILE ...` o `CREATE TABLE ... SELECT`.

Para al versión 5.0, MySQL soporta `SELECT ... INTO` con variables de usuario. La misma sintaxis puede usarse dentro de procedimientos almacenados usando cursores y variables locales. Consulte Sección 19.2.9.3, “La sentencia `SELECT ... INTO`”.

1.7.5.3. Transacciones y operaciones atómicas

MySQL Server (versiones 3.23-max y todas las versiones 4.0 y posteriores) soportan transacciones con los motores trasaccionales `InnoDB` y `BDB` . `InnoDB` proporciona *completa* compatibilidad `ACID`. Consulte Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

Los otros motores no transaccionales en MySQL Server (como `MyISAM`) siguen un paradigma diferente para integridad de datos llamado "operaciones atómicas". En términos transaccionales, tablas `MyISAM` operan en modo `AUTOCOMMIT=1`. Operaciones atómicas a menudo ofrecen integridad comparable con mejor rendimiento.

MySQL Server soporta ambos paradigmas, puede decidir si su aplicación necesita la velocidad de operaciones atómicas o el uso de características transaccionales. Esta elección puede hacerse para cada tabla.

Como se ha dicho, el compromiso entre tipos de tablas transaccionales y no transaccionales reside principalmente en el rendimiento. Tablas transaccionales tienen requerimientos significativamente mayores para memoria y espacio de disco, y mayor carga de CPU. Por otra parte, tipos de tablas transaccionales como `InnoDB` también ofrece muchas características significativas. El diseño modular de MySQL Server permite el uso concurrente de distintos motores de almacenamiento para cumplir distintos requerimientos y mostrarse óptimo en todas las situaciones.

Pero, ¿cómo usar las características de MySQL Server para mantener integridad de forma rigurosa incluso en tablas no transaccionales como `MyISAM` , y cómo se comparan estas características con los tipos de tablas transaccionales?

1. Si su aplicación está escrita de forma que dependa en que pueda llamar a `ROLLBACK` en lugar de `COMMIT` en situaciones críticas, es preferible usar transacciones. Transacciones aseguran que actualizaciones no acabadas o actividades corruptas no se ejectuen en la base de datos; el servidor tiene la oportunidad de hacer un rollback automático para mantener la base de datos a salvo.

   Si usa tablas no transaccionales, MySQL Server le permite solucionar problemas potenciales en prácticamente todos los casos simplemente incluyendo chequeos antes de las actualizaciones y ejecutando scripts sencillos que comprueban que la consistencia de la base de datos, dando una advertencia o reparando automáticamente cualquier incosistencia. Simplemente usando el log de MySQL o añadiendo un log extra, normalmente puede arreglar tablas sin pérdida de integridad en los datos.

2. Normalmente, las actualizaciones transaccionales críticas pueden reescribirse como atómicas.Generalmente hablando, todos los problemas de integridad que resuelven las transacciones

pueden resolverse con `LOCK TABLES` o actualizaciones atómicas, asegurando que no se aborten automáticamente desde el servidor, el cuál es un problema habitual en sistemas de bases de datos transaccionales.

3. Para tener un entorno fiable de MySQL, usando tablas transaccionales o no, sólo necesita tener copias de seguridad y el log binario activado. Con ello, puede recuperarse de cualquier situación de la que pueda hacerlo con cualquier otro sistema transaccional. Siempre es bueno tener copias de seguridad, independientemente del sistema de bases de datos usado.

El paradigma transaccional tiene sus ventajas y desventajas. Muchos usuarios y desarrolladores de aplicaciones dependen en la facilidad con la que pueden solucionar problemas donde un aborto parece ser o es necesario. Sin embargo, incluso si el paradigma de operaciones atómicas le es desconocido o está más familiarizado con las transacciones, considere el beneficio de la velocidad que pueden ofrecer las tablas no transaccionales, que puede ser de tres a cinco veces más rápido que las más optimizadas tablas transaccionales.

En las situaciones en las que la integridad es de máxima importancia, MySQL Server ofrece integridad a nivel de transacción incluso para tablas no transaccionales. Si bloquea tablas con `LOCK TABLES`, todas las actualizaciones se bloquean hasta que se hacen las comprobaciones necesarias. Si obtiene un bloqueo `READ LOCAL` (el contrario a un bloqueo de escritura) para una tabla que permita inserciones concurrentes al final de la tabla, las lecturas están permitidas, así como las inserciones de otros clientes. Los registros insertados no puede verlos el cliente que tenga el bloqueo hasta que lo libere. Con `INSERT DELAYED`, puede encolar inserciones en una cola local, hasta que los bloqueos se liberan, sin tener que esperar el cliente a que acabe la inserción. Consulte Sección 13.2.4.2, “Sintaxis de `INSERT DELAYED`”.

"Atómico", en el sentido en que nos referimos, no es nada mágico. Se trata que puede asegurar que mientras cada actualización específica está ejecutándose, ningún otro usuario puede interferir con ellas, y que nunca puede haber un rollback automático (lo que puede ocurrir con tablas transaccionales si no se es muy cuidadoso). MySQL Server garantiza que no hay dirty reads (lecturas sucias).

A continación se presentan algunas técnicas para trabajar con tablas no transaccionales:

- Los bucles que necesiten transacciones normalmente pueden codificarse con la ayuda de `LOCK TABLES`, y no necesita cursores para actualizar registros en tiempo real.
- Para evitar usar `ROLLBACK`, puede usar la siguiente estrategia:

  1. Use `LOCK TABLES` para bloquear todas las tablas a las que quiere acceder.
  2. Compruebe las condiciones que deben darse antes de ejecutar la actualización.
  3. Actualice si todo es correcto.
  4. Use `UNLOCK TABLES` para liberar los bloqueos.

  Este es un método mucho más rápido que usar transacciones con posibles rollbacks, aunque no siempre. La única situación en que esta situación no funciona es cuando alguien mata el thread durante una actualización. En ese caso, todos los bloqueos se liberan pero algunas actualizaciones pueden no ejecutarse.

- Puede usar funciones para actualizar registros en una única operación. Puede obtener una aplicación muy eficiente usando las siguientes técnicas:

  - Modifique columnas con su valor actual.
  - Actualice sólo aquéllas que hayan cambiado.

Por ejemplo, cuando estamos actualizando la información de un cliente, sólo actualizamos los datos del cliente que han cambiado y comprobamos que los datos cambiados o datos que dependen de los datos cambiados, han cambiado respecto a los datos originales. El test para datos cambiados se hace con la cláusula `WHERE` en el comando `UPDATE` . Si el registro no se ha actualizado, mostramos un mensaje al cliente: "Algunos de los datos actualizados han sido cambiados por otro usuario". A continuación mostramos los registros viejos junto a los nuevos en una ventana para que el usuario pueda decidir qué versión del registro de usuario usar.

Esto nos da algo que es similar a bloqueo de columnas pero es incluso mejor ya que sólo actualizamos algunas de las columnas, usando valores que son relativos a sus valores actuales. Eso significa que el típico comando `UPDATE` será algo así:

```
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';
```

Esto es muy eficiente y funciona incluso si otro cliente ha cambiado los valores en las columnas `pay_back` o `money_owed_to_us`.

-

1.7.5.4. Procedimientos almacenados (stored procedures) y disparadores (triggers)

Los procedimientos almacenados se implementan desde la versión 5.0. Consulte Capítulo 19, *Procedimientos almacenados y funciones*.

Funcionalidad básica para disparadores se implementa en MySQL desde la versión 5.0.2, con desarrollo adicional planeado para MySQL 5.1. Consulte Capítulo 20, *Disparadores (triggers)*.

1.7.5.5. Claves foráneas (foreign keys)

En MySQL Server 3.23.44 y posteriores, el motor `InnoDB` soporta chequeo para restricciones de claves foráneas, incluyendo `CASCADE`, `ON DELETE`, y `ON UPDATE`. Consulte Sección 15.6.4, “Restricciones (constraints) `FOREIGN KEY`”.

Para otros motores diferentes a `InnoDB`, MySQL Server parsea la sintaxis de `FOREIGN KEY` en comandos `CREATE TABLE`, pero no lo usa ni almacena. En el futuro, la implemantación se extenderá para almacenar esta información en el fichero de especificaciones de las tablas de forma que puedan obtenerla `mysqldump` y ODBC. En una etapa posterior, restricciones de claves foráneas se implementarán para tablas `MyISAM`.

Restricciones de claves foráneas ofrecen distintos beneficios a los diseñadores de bases de datos:

- Suponiendo un diseño adecuado de las relaciones, las restricciones de claves foráneas hacen más difícil que un programador introduzca inconsistencias en la base de datos.
- Chequeo centralizado de restricciones por el servidor de base de datos hace que sea innecesario realizar esos chequeos en la parte de la aplicación, eliminando la posibilidad que distintas aplicaciones puedan no chequear todas las restricciones de la misma forma.

- Usando actualizaciones y borrados en cascada puede simplificarse el código de aplicación.
- Reglas diseñadas correctamente para claves foráneas pueden ayudar a documentar las relaciones entre tablas.

Tenga en cuenta que estos beneficios tienen el coste de un trabajo adicional para el servidor de base de datos para poder realizar todas las comprobaciones necesarias. Chequeos adicionales por parte del servidor afectan al rendimiento, lo que puede ser lo suficientemente malo para algunas aplicaciones como para evitarlo todo lo posible. (Algunas grandes aplicaciones comerciales han codificado la lógica de claves foráneas en el nivel de aplicación por esta razón.)

MySQL proporciona a diseñadores de bases de datos la posibilidad de elegir qué paradigma elegir. Si no necesita claves foráneas y quiere evitar la sobrecarga asociada con la integridad referencial, puede usar otro tipo de tabla como `MyISAM`. (Por ejemplo, el motor `MyISAM` ofrece muy buen rendimiento para aplicaciones que sólo realizan operaciones `INSERT` y `SELECT`, ya que las inserciones de pueden utilizar de forma concurrente con consultas. Consulte Sección 7.3.2, “Cuestiones relacionadas con el bloqueo (locking) de tablas”.)

Si elige no utilizar integridad referencial, tenga en cuenta las siguientes consideraciones:

- Sin un chequeo por parte del servidor de integridad referencial, la aplicación debe realizar este trabajo. Por ejemplo, debe tener cuidado de insertar registros en tablas en el orden apropiado, y evitar crear registros con hijos huérfanos. También debe ser capaz de recuperarse de errores que ocurran durante inserciones múltiples.
- Si `ON DELETE` es la única integridad referencial que necesita la aplicación, desde la versión 4.0 de MySQL Server puede usar comandos `DELETE` para borrar registros de distintas tablas con un único comando. Consulte Sección 13.2.1, “Sintaxis de DELETE”.
- Una forma de suplir la falta de `ON DELETE` es añadir el comando `DELETE` apropiado a su aplicación cuando borre registros de una tabla que no tenga clave foránea. En la práctica, esto es tan rápido como usar una clave foránea, y más portable.

Tenga en cuenta que el uso de claves foráneas puede provocar algunos problemas:

- El soporte de claves foráneas arregla muchas cuestiones relacionadas con la integridad, pero todavía es necesario diseñar las claves cuidadosamente para evitar reglas circulares o combinaciones incorrectas de borrados en cascada.
- Es posible crear una topología de relaciones que haga difícil restaurar tablas individuales de una copia de seguridad. (MySQL alivia esta dificultad permitiendo desactivar claves foráneas temporalmente al recargar una tabla que dependa de otras. Consulte Sección 15.6.4, “Restricciones (constraints) FOREIGN KEY”. Desde MySQL 4.1.1, `mysqldump` genera ficheros que utilizan esta características automáticamente al recargarse.)

Tenga en cuenta que las claves foráneas en SQL se usan para chequear y forzar integridad referencial, no para unir tablas. Si quiere obtener resultados de múltiples tablas a partir de un comando `SELECT`, debe usar un join entre ellas:

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id;
```

Consulte Sección 13.2.7.1, “Sintaxis de JOIN”. Consulte Sección 3.6.6, “Usar claves foráneas (foreign keys)”.

La sintaxis de `FOREIGN KEY` sin `ON DELETE ...` se usa a menudo por aplicaciones ODBC para producir cláusulas `WHERE` automáticamente.

1.7.5.6. Vistas

Vistas (incluyendo vistas actualizables) se implementan en la versión 5.0 de MySQL Server. Las vistas están disponibles en las versiones binarias a partir de la 5.0.1. Consulte Capítulo 21, *Vistas (Views)*.

Las vistas son útiles para permitir acceder a los usuarios a un conjunto de relaciones (tablas) como si fueran una sola, y limitar su acceso a las mismas. También se pueden usar las vistas para restringir el acceso a registros (un subconjunto de una tabla particular). Para control de acceso a columnas, puede usar el sofisticado sistema de privilegios de MySQL Server. Consulte Sección 5.6, “El sistema de privilegios de acceso de MySQL”.

Al diseñar la implementación de las vistas, nuestro ambicioso objetivo, dentro de los límites de SQL, ha sido la plena compatibilidad con la regla 6 de Codd para sistemas relacionales de bases de datos: "Todas las vistas que son actualizables en teoría, deben serlo en la práctica".

1.7.5.7. Empezar un comentario con '`--`'

Algunas otras bases de datos SQL utilizan '`--`' para comenzar comentarios. MySQL Server utiliza '`#`' como carácter para comenzar comentarios. Puede utilizar comentarios estilo C `/*this is a comment */` con MySQL Server. Consulte Sección 9.5, “Sintaxis de comentarios”.

MySQL Server 3.23.3 y posteriores permiten comentarios del estilo '`--`', mientras el comentario esté seguido de un carácter (o por un carácter de controlo como una nueva línea). Se necesita dicho espacio para evitar problemas con consultas SQL generadas automáticamente que usen algo parecido al código a continuación, donde se inserta automáticamente el valor de pago para `!payment!`:

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

Piense acerca de lo que ocurre si el valor de `payment` es un valor negativo como `-1`:

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

`credit--1` es una expresión legal en SQL, pero si `--` se interpreta como parte de un comentario, una parte de la expresión se descarta. El resultado es un comando que tiene un significado completamente distinto al deseado:

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

Este comando no produce ningún cambio en absoluto! Lo cual ilustra que permitir comentaros que empiecen con '`--`' puede tener serias consecuencias.

Usando la implementación de este método de comentarios en MySQL Server desde 3.23.3, `credit--1` es seguro.

Otra medida de seguridad es que el cliente de líneas de comando `mysql` elimina todas las líneas que empiecen con '`--`'.

La siguiente información sólo es relevante si utiliza versiones anteriores a la 3.23.3:

Si tiene un programa SQL en un fichero de texto que contenga comentarios del estilo '`--`', debería utilizar la utilidad `replace` para convertir los comentarios antiguos en caracteres '`#`' de la siguiente forma:

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

```
        | mysql db_name
```

en lugar del usual:

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

También puede editar el fichero "a mano" para cambiar los comentarios '--' a '#':

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

Vuelva a cambiarlos con el comando:

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

1.7.6. Cómo trata MySQL las restricciones (Constraints)

MySQL le permite trabajar con tablas transaccionales, que permiten hacer un rollback, y con tablas no transaccionales que no lo permiten. Es por ello que las restricciones son algo distintas en MySQL respecto a otras bases de datos. Debemos tratar el caso en el que se insertan o actualizan muchos registros en una tabla no transaccional en la que los cambios no pueden deshacerse cuando ocurre un error.

La filosofía básica es que MySQL Server trata de producir un error para cualquier cosa que detecte mientras parsea un comando que va a ejecutarse, y trata de recuperarse de cualquier error que ocurra mientras se ejecuta el comando. Lo hacemos en la mayoría de casos, pero todavía no en todos.

Las opciones en MySQL cuando ocurre un error son parar el comando en medio de la ejecución o recuperarse lo mejor posible del problema y continuar. Por defecto, el servidor utiliza esta última opción. Esto significa, por ejemplo, que el servidor puede cambiar algunos valores ilegales por el valor legal más próximo.

A partir de la versión 5.0.2 de MySQL, hay disponibles varios modos SQL para proporcionar un mayor control sobre cómo aceptar valores incorrectos y si continuar ejecutando el comando o abortarlo cuando ocurre el error. Usando estas opciones, puede configurar MySQL Server para actuar en un modo más tradicional como otros servidores de bases de datos que rechazan datos incorrectos. Los modos SQL pueden cambiarse en tiempo de ejecución, lo que permite a los clientes individuales seleccionar el comportamiento más apropiado para sus requerimientos. Consulte Sección 5.3.2, “El modo SQL del servidor”.

La siguiente sección describe qué ocurre para diferentes tipos de restricciones.

1.7.6.1. Restricciones (constraints) en los índices `PRIMARY KEY` y `UNIQUE`

Normalmente, un error ocurre cuando trata de ejecutar un `INSERT` o `UPDATE` en un registro que viole la clave primaria, clave única o clave foránea. Si usa un motor transaccional como `InnoDB`, MySQL automáticamente deshace el comando. Si usa un motor no transaccional, MySQL para de procesar el comando en el registro en el que ocurre el error y deja sin procesar el resto de registros.

Si desea ignorar este tipo de violaciones de claves, MySQL permite la palabra clave `IGNORE` para `INSERT` y `UPDATE`. En este caso, MySQL ignora cualquier violación de clave y continúa procesando el siguiente registro. Consulte Sección 13.2.4, “Sintaxis de `INSERT`”. See Sección 13.2.10, “Sintaxis de `UPDATE`”.

Puede obtener información acerca del número de registro insertados o actualizados realmente con la función de la API de C `mysql_info()`. Consulte Sección 24.2.3.32, “`mysql_info()`”. A partir de MySQL 4.1 puede usar el comando `SHOW WARNINGS`. Consulte Sección 13.5.4.22, “Sintaxis de `SHOW WARNINGS`”.

De momento, sólo las tablas `InnoDB` soportan claves foráneas. Consulte Sección 15.6.4, “Restricciones (constraints) `FOREIGN KEY`”. El soporte para claves foráneas para tablas `MyISAM` está previsto para implementarse en MySQL 5.1.

1.7.6.2. Restricciones (constraints) sobre datos inválidos

Antes de la versión 5.0.2 de MySQL, se permitía insertar valores ilegales convirtiéndolos en valores legales. A partir de la versión 5.0.2, sigue este compartimiento por defecto, pero puede elegir un tratamiento para valores incorrectos más tradicional, como no aceptarlos y abortar los comandos que los incluyen. Esta sección describe el comportamiento por defecto de MySQL (permisivo), así como el nuevo modo estricto SQL y en qué se diferencian.

Lo siguiente es cierto si no usa modo estricto. Si inserta un valor "incorrecto" en una columna, como `NULL` en una columna `NOT NULL` o una valor numérico demasiado grande en una columna numérica, MySQL cambia el valor al "mejor valor posible" para la columna en lugar de producir un error:

- Si trata de almacenar un valor fuera de rango en una columna numérica, MySQL Server en su lugar almacena cero, el menor valor posible, o el mayor valor posible en la columna.
- Para cadenas de caracteres, MySQL almacena una cadena vacía o tanto de la cadena de carácteras como quepa en la columna.
- Si trata de almacenar una cadena de caracteres que no empiece con un número en una columna numérica, MySQL Server almacena 0.
- MySQL le permite almacenar ciertos valores incorrectos en columnas `DATE` y `DATETIME` (tales como `'2000-02-31'` o `'2000-02-00'`). La idea es que no es el trabajo del servidor SQL validar fechas. Si MySQL puede almacenar una fecha y recuperarla fielmente, se almacena tal y como se da. Si la fecha es totalmente incorrecta (más allá de la capacidad del servidor para almacenarla), se almacena en su lugar el valor especial `'0000-00-00'`.
- Si intenta almacenar `NULL` en una columna que no admita valores `NULL` ocurre un error para los comandos `INSERT` de un solo registro. Para comandos `INSERT` de varios registros o para comandos `INSERT INTO... SELECT` , MySQL Server almacena el valor implícito para el tipo de datos de la columna. En general, es `0` para tipos numéricos, cadena vacía (`""`) para tipos de cadenas de caracteres, y el valor "cero" para tipos de fecha y tiempo. Los valores implícitos por defecto se discuten en Sección 13.1.5, “Sintaxis de `CREATE TABLE`”.
- Si un comando `INSERT` no especifica un valor para una columna, MySQL inserta su valor por defecto si la columna especifica un valor mediante la cláusula `DEFAULT`. Si la definición no tiene tal cláusula `DEFAULT` clause, MySQL inserta el valor por defecto implícito para el tipo de datos de la columna.

La razón para las reglas anteriores es que no podemos validar esas condiciones hasta que los comandos han empezado a ejecutarse. No podemos deshacer si encontramos un problema tras actualizar algunos registros, ya que el motor de almacenamiento puede no soportar rollback. La opción de terminar el comando no es siempre positiva; en este caso, la actualización quedaría "a medias", lo que posiblemente es la peor opción. En este caso, lo mejor es hacerlo "lo mejor posible" y continuar como si nada hubiera ocurrido.

A partir de MySQL 5.0.2, puede seleccionar un tratamiento de validación de datos de entrada más estricto usando los modos SQL `STRICT_TRANS_TABLES` o `STRICT_ALL_TABLES`. Consulte Sección 5.3.2, “El modo SQL del servidor”.

`STRICT_TRANS_TABLES` funciona así:

- Para motores de almacenamiento transaccionales, valores incorrectos en cualquier parte del comando provocan que se aborte el comando y se deshaga el mismo.

- Para motores no transaccionales, el comando aborta si ocurre un error en el primer registro a insertar o actualizar. (En este caso, el comando dejará la tabla intacta, igual que en una tabla transaccional.) Los errores en registros después del primero no abortan el comando. En lugar de ello, los valores incorrectos se ajustan y se generan advertencias en lugar de errores. En otras palabras, con `STRICT_TRANS_TABLES`, un valor incorrecto provoca que MySQL deshaga todas las actualizaciones hechas hasta el momento, si es posible.

Para chequeo estricto, active `STRICT_ALL_TABLES`. Es equivalente a `STRICT_TRANS_TABLES` excepto que en motores de almacenamiento no transaccionales, los errores abortan el comando incluso cuando hay datos incorrectos a partir del primer registro. Esto significa que si ocurre un error a medias de una inserción o actualización de varios registros en una tabla no transaccional, se produce una actualización parcial. Los primeros registros se insertan o actualizan, pero aquéllos a partir del punto en que ocurre el error no. Para evitar esto en tablas no transaccionales, use inserciones de un solo registro o use `STRICT_TRANS_TABLES` para obtener advertencias en lugar de errores. Para evitar problemas, no utilice MySQL para validar contenido de columnas. Es preferible (y a menudo más rápido) dejar que la aplicación se asegure de pasar sólo valores legales a la base de datos.

Con cualquiera de las opciones de modo estricto, puede hacer que se traten los errores como advertencias usando `INSERT IGNORE` o `UPDATE IGNORE` en lugar de `INSERT` o `UPDATE` sin `IGNORE`.

1.7.6.3. Restricciones `ENUM` y `SET`

Las columnas `ENUM` y `SET` proporcionan una manera eficiente de definir columnas que contienen un conjunto dado de valores. Sin embargo, antes de MySQL 5.0.2, `ENUM` y `SET` no son restricciones reales. Esto es por la misma razón que `NOT NULL` tampoco lo es. Consulte Sección 1.7.6.2, “Restricciones (constraints) sobre datos inválidos”.

Las columnas de tipo `ENUM` siempre tienen un valor por defecto. Si no especifica un valor por defecto, entonces será `NULL` para las columnas que permitan valores `NULL`, si no, se utiliza el primer valor de la enumeración como valor por defecto.

Si inserta un valor incorrecto en una columna `ENUM` o si fuerza insertar un valor en una columna `ENUM` con `IGNORE`, se inicializa al valor reservado para enumeraciones `0`, el cual se muestra como una cadena vacía en un contexto de cadenas de caracteres. Consulte Sección 11.4.4, “El tipo de columna `ENUM`”.

Si insertar un valor incorrecto en una columna `SET`, se ignora el valor incorrecto. Por ejemplo, si la columna puede contener los valores `'a'`, `'b'`, y `'c'`, un intento de insertar `'a,x,b,y'` resulta en un valor de `'a,b'`. Consulte Sección 11.4.5, “El tipo `SET`”.

En MySQL 5.0.2, puede configurar el servidor para que use el modo estricto de SQL. Consulte Sección 5.3.2, “El modo SQL del servidor”. Cuando el modo estricto está activado, la definición de columnas `ENUM` o `SET` no actúa como una restricción en valores insertados en la columna. Los valores que no satisfacen estas condiciones provocan un error:

- Un valor `ENUM` debe elegirse entre los listados en la definición de la columna, o el equivalente numérico interno. El valor no puede ser un valor de error (esto es 0 o la cadena vacía). Para una columna definida como `ENUM('a','b','c')`, valores tales como `""`, `'d'`, y `'ax'` son ilegales y rehusados.

- Un valor `SET` debe ser una cadena vacía o un valor consistente en uno o más valores listados en la definición de la columna separados por comas. Para una columna definida como `SET('a','b','c')`, valores tales como `'d'`, y `'a,b,c,d'` serían ilegales y, por lo tanto, rehusados.

Se pueden suprimir los errores derivados de valores inválidos en modo estricto usando `INSERT IGNORE` o `UPDATE IGNORE`. En ese caso, se genera una advertencia en lugar de un error. Para tipos `ENUM`, el valor se inserta como un miembro erróneo (`0`). Para tipo `SET`, el valor se inserta igual excepto que

se borra cualquier subcadena inválida. Por ejemplo, `'a,x,b,y'` se convertiría en `'a,b'`, como se ha descrito.

Capítulo 2. Instalar MySQL

Tabla de contenidos

En este capítulo se describe cómo obtener e instalar MySQL:

1. **Debe determinarse si la plataforma donde se desea hacer la instalación está soportada.** Nótese que no todos los sistemas soportados son igualmente adecuados para ejecutar MySQL. En algunas plataformas el funcionamiento será mucho más robusto y eficiente que en otras. Consulte Sección 2.1.1, “Sistemas operativos que MySQL soporta” para más detalles.

2. **Debe elegirse la distribución que se instalará.** Hay varias versiones de MySQL disponibles, y la mayoría lo están en varios formatos de distribución. Se puede elegir entre distribuciones prearmadas que contienen programas binarios (precompilados) o bien código fuente. En caso de duda, debe elegirse una distribución binaria. También se provee acceso público al código fuente para quienes deseen ver los desarrollos más recientes y colaborar en el testeo de código nuevo. Para establecer qué versión y tipo de distribución debería usarse, consulte Sección 2.1.2, “Escoger la distribución MySQL a instalar”.

3. **Descargar la distribución que se desea instalar.** Para ver una lista de sitios desde los cuales se puede obtener MySQL, consúltese Sección 2.1.3, “Cómo obtener MySQL”. Se puede verificar la integridad de la distribución como se describe en Sección 2.1.4, “Comprobar la integridad de paquetes con sumas de verificación MD5 o GnuPG”.

4. **Instalar la distribución.** Para instalar MySQL desde una dsitribución binaria, empleense las instrucciones en Sección 2.2, “Instalación MySQL estándar con una distribución binaria”. Para instalar MySQL a partir de una distribución de código fuente o desde el directorio de desarrollo actual, utilícense las instrucciones en Sección 2.8, “Instalación de MySQL usando una distribución de código fuente”.

   Nota: Si se planea actualizar una versión existente de MySQL a una versión más nueva en lugar de instalarlo por primera vez, consúltese Sección 2.10, “Aumentar la versión de MySQL” para obtener información acerca de procedimientos de actualización y de cuestiones que se deberían de considerar antes de actualizar.

   Si se encontrasen problemas durante la instalación, consúltese Sección 2.12, “Notas específicas sobre sistemas operativos” para obtener información sobre la resolución de problemas en plataformas específicas.

5. **Realizar cualquier ajuste que sea necesario con posterioridad a la instalación.** Luego de instalar MySQL, léase Sección 2.9, “Puesta en marcha y comprobación después de la instalación”. Esta sección contiene información importante acerca de cómo asegurarse de que el servidor de MySQL está funcionando adecuadamente. También describe cómo hacer seguras las cuentas de usuario iniciales de MySQL, *que no poseen contraseñas* hasta que se les hayan asignado. Esta sección es de aplicación si MySQL se ha instalado utilizando tanto una distribución binaria como una de código fuente.

6. Si se desea ejecutar los scripts para medir el rendimiento de MySQL, debe estar disponible el soporte de Perl para MySQL. Consúltese Sección 2.13, “Notas sobre la instalación de Perl”.

2.1. Cuestiones generales sobre la instalación

Antes de instalar MySQL, se debería hacer lo siguiente:

1. Determinarse si la plataforma donde se desea hacer la instalación está soportada.
2. Elegirse la distribución que se instalará.
3. Descargar la distribución que se desea instalar y verificar su integridad.

Esta sección contiene la información necesaria para llevar adelante estos pasos. Una vez ejecutados, se puede seguir las instrucciones de secciones posteriores del capítulo, para instalar la distribución elegida.

2.1.1. Sistemas operativos que MySQL soporta

En esta sección aparecen listados los sistemas operativos en los que es posible instalar MySQL.

Se ha utilizado GNU Autoconfig, de modo que es posible portar MySQL a todos los sistemas modernos que tengan un compilador de C++ y una implementación funcional de subprocesos (threads) POSIX. (El soporte de subprocesos es necesario para el servidor. Para compilar únicamente el código del cliente, no se requiere más que el compilador de C++). Nosotros mismos desarrollamos y utilizamos el software ante todo en Linux (SuSE y Red Hat), FreeBSD, y Sun Solaris (Versiones 8 y 9),

MySQL ha sido compilado correctamente en las siguientes combinaciones de sistemas operativos y paquetes de subprocesos. Nótese que, para varios sistemas operativos, el soporte nativo de subprocesos funciona solamente en las versiones más recientes.

- AIX 4.x, 5.x con subprocesos nativos. Consulte Sección 2.12.5.3, “Notas sobre IBM-AIX”.
- Amiga.
- BSDI 2.x with con el paquete MIT-pthreads. Consulte Sección 2.12.4.5, “Notas sobre BSD/OS Version 2.x”.
- BSDI 3.0, 3.1 y 4.x con subprocesos nativos. Consulte Sección 2.12.4.5, “Notas sobre BSD/OS Version 2.x”.
- Digital Unix 4.x con subprocesos nativos. Consulte Sección 2.12.5.5, “Notas Alpha-DEC-UNIX (Tru64)”.
- FreeBSD 2.x con el paquete MIT-pthreads. Consulte Sección 2.12.4.1, “Notas sobre FreeBSD”.
- FreeBSD 3.x and 4.x con subprocesos nativos. Consulte Sección 2.12.4.1, “Notas sobre FreeBSD”.
- FreeBSD 4.x con LinuxThreads. Consulte Sección 2.12.4.1, “Notas sobre FreeBSD”.
- HP-UX 10.20 con el paquete DCE threads o MIT-pthreads. Consulte Sección 2.12.5.1, “Notas sobre HP-UX Version 10.20”.
- HP-UX 11.x con subprocesos nativos. Consulte Sección 2.12.5.2, “Notas sobre HP-UX Version 11.x”.
- Linux 2.0+ con LinuxThreads 0.7.1+ o `glibc` 2.0.7+ para varias arquitecturas de CPU. Consulte Sección 2.12.1, “Notas sobre Linux”.
- Mac OS X. Consulte Sección 2.12.2, “Notas sobre Mac OS X”.
- NetBSD 1.3/1.4 Intel y NetBSD 1.3 Alpha (requiere GNU make). Consulte Sección 2.12.4.2, “Notas sobre NetBSD”.
- Novell NetWare 6.0. Consulte Sección 2.6, “Instalar MySQL sobre NetWare”.

- OpenBSD > 2.5 con subprocesos nativos. OpenBSD < 2.5 con el paquete MIT-pthreads. Consulte Sección 2.12.4.3, “Notas sobre OpenBSD 2.5”.
- OS/2 Warp 3, FixPack 29 y OS/2 Warp 4, FixPack 4. Consulte Sección 2.12.6, “Notas sobre OS/2”.
- SCO OpenServer 5.0.X con una versión del paquete FSU Pthreads recientemente portada. Consulte Sección 2.12.5.8, “Notas sobre SCO UNIX y OpenServer 5.0.x”.
- SCO UnixWare 7.1.x. Consulte Sección 2.12.5.9, “Notas sobre SCO UnixWare 7.1.x y OpenUNIX 8.0.0”.
- SCO Openserver 6.0.x. Consulte Sección 2.12.5.10, “Notas sobre SCO OpenServer 6.0.x”.
- SGI Irix 6.x con subprocesos nativos. Consulte Sección 2.12.5.7, “Notas sobre SGI Irix”.
- Solaris 2.5 y posteriores con subprocesos nativos en SPARC y x86. Consulte Sección 2.12.3, “Notas sobre Solaris”.
- SunOS 4.x con el paquete MIT-pthreads package. Consulte Sección 2.12.3, “Notas sobre Solaris”.
- Tru64 Unix. Consulte Sección 2.12.5.5, “Notas Alpha-DEC-UNIX (Tru64)”.
- Windows 9x, Me, NT, 2000, XP, y 2003. Consulte Sección 2.3, “Instalar MySQL en Windows”.

No todas las plataformas son igualmente aptas para ejecutar MySQL. Los siguientes factores determinan si una plataforma está más o menos bien preparada para un servidor MySQL con alto volumen de carga y para misiones crítica:

- Estabilidad general de la biblioteca de subprocesos. Una plataforma puede tener una excelente reputación en otras situaciones, pero MySQL es estable como lo sea la biblioteca de subprocesos que utiliza la plataforma, aun cuando cualquier otro aspecto sea perfecto.
- La capacidad del núcleo o kernel del sistema operativo y de la biblioteca de subprocesos para aprovechar sistemas de multiprocesamiento simétrico (SMP). En otras palabras, cuando un proceso crea un subproceso, éste debería poderse ejecutar en una CPU diferente a la del proceso original.
- La capacidad del núcleo o kernel del sistema operativo y de la biblioteca de subprocesos para ejecutar varios subprocesos que bloquean y liberan mutexes frecuentemente en una pequeña región crítica sin excesivos cambios de contexto. Si la implementación de `pthread_mutex_lock()` es muy proclive a consumir tiempo de CPU, esto afectará en gran manera a MySQL. Si no se previene este problema, añadir más CPUs hará todavía más lento a MySQL.
- El rendimiento y la estabilidad general del sistema de ficheros.
- Si se emplean grandes tablas, la capacidad del sistema de ficheros para gestionar eficientemente archivos de gran tamaño.
- El nivel de experiencia que los desarrolladores de MySQL AB posean sobre una determinada plataforma. Si la conocen bien, habilitan optimizaciones específicas y soluciones en tiempo de compilación. Además pueden proporcionar consejos sobre cómo configurar el sistema en forma óptima para MySQL.
- El volumen de pruebas realizadas por MySQL AB sobre configuraciones similares.
- La cantidad de usuarios que han ejecutado MySQL con éxito en la misma plataforma y en configuraciones similares. Si este número es alto, las probabilidades de encontrar sorpresas específicas de la plataforma son mucho menores.

En base a estos criterios, las mejores plataformas para ejecutar MySQL en este momento son x86 con SuSE Linux (kernel versión 2.4 o 2.6), y ReiserFS (o cualquier distribución de Linux similar) y SPARC

con Solaris (2.7-9). FreeBSD aparece en tercer lugar, pero es de esperar que se integre al lote principal cuando se mejore la biblioteca de subprocesos. También las otras plataformas donde MySQL se compila y ejecuta en la actualidad podrian ser incluidas en la categoria principal, pero no con el mismo nivel de estabilidad y rendimiento. Esto requiere un esfuerzo por parte de los desarrolladores de MySQL en cooperación con los desarrolladores de los sistemas operativos y de bibliotecas de componentes de las que depende MySQL. Si Usted está interesado en mejorar alguno de estos componentes, está en posición de influir en su desarrollo, y necesita información más detallada acerca de lo que MySQL requiere para funcionar mejor, envíe un mensaje de correo electrónico a la lista de correo `internals` de MySQL. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”.

El propósito de la anterior comparación no es afirmar que un sistema es, en términos generales, mejor o peor que otro. Se trata solamente de la elección de un sistema operativo con el objetivo de ejecutar MySQL. Por lo tanto, el resultado de la comparación podría ser diferente si se consideraran otros factores. En algunos casos, la razón de que un sistema operativo sea mejor que otros podría residir simplemente en que los desarrolladores de MySQL han podido dedicar más esfuerzos a la prueba y optimización sobre una plataforma en particular. Lo aquí manifestado son las observaciones de estos desarrolladores a fin de ayudar al usuario a decidir la plataforma sobre la que ejecutar MySQL.

2.1.2. Escoger la distribución MySQL a instalar

Como parte de los preparativos para instalar MySQL, debe decidirse qué versión se utilizará. El desarrollo de MySQL se divide en entregas (releases) sucesivas, y el usuario puede decidir cuál es la que mejor satisface sus necesidades. Después de haber elegido la versión a instalar, se debe optar por un formato de distribución. Las entregas están disponibles en formato binario o código fuente.

2.1.2.1. Escoger la versión de MySQL a instalar

La primera decisión a tomar es si se desea emplear una entrega "en producción" (estable) o una entrega de desarrollo. En el proceso de desarrollo de MySQL coexisten múltiples entregas, cada una con un diferente estado de madurez:

- MySQL 5.1 es la próxima serie de entregas de desarrollo, y en ella se implementarán las nuevas características. En breve se pondrán a disposición de los usuarios interesados en hacer pruebas integrales las entregas Alfa.
- MySQL 5.0 es la serie de entregas estables (para producción). Solamente se liberan nuevas entregas para corrección de errores, no se añaden nuevas características que pudieran afectar a la estabilidad.
- MySQL 4.1 es la anterior serie de entregas estables (para producción). Se liberarán nuevas entregas para solucionar problemas de seguridad o errores críticos. En esta serie no se agregarán nuevas caracteristicas de importancia.
- MySQL 4.0 y 3.23 son las antiguas series de entregas estables (para producción). Estas versiones están discontinuadas, de modo que solamente se liberarán nuevas entregas para solucionar errores de seguridad extremadamente críticos.

Los desarrolladores de MySQL no son partidarios de la "congelación" total del código de una versión, puesto que anula la posibilidad de introducir soluciones a errores. Cuando se habla de algo “congelado” se quiere expresar que no se harán más que pequeñas modificaciones que no deberían afectar a la forma en que funciona actualmente una entrega en un entorno de producción. Naturalmente, los errores que se corrigen en una serie se propagan a las siguientes si son relevantes.

Si el usuario está comenzando a emplear MySQL por primera vez o intentando su implementación en un sistema para el que no hay una distribución binaria, es recomendable instalar una entrega perteneciente a una serie en producción. Actualmente, MySQL 5.0. Todas las entregas de MySQL, aun aquellas

pertenecientes a una serie en desarrollo, se verifican con las pruebas de rendimiento de MySQL y se prueban extensamente antes de liberarse.

Si se está ejecutando una versión más antigua y se desea actualizar, pero se quiere evitar un cambio brusco con el consiguiente riesgo de incompatibilidades, debería actualizarse a la última versión dentro de la misma serie de entregas que se utiliza actualmente (es decir, aquélla donde únicamente la última parte del número de versión es más nueva que la actual). En dicha versión los desarrolladores de MySQL han tratado de corregir solamente errores fatales y realizar cambios pequeños, relativamente “seguros”.

Si se desea emplear características nuevas que no están en las entregas para ambientes de producción, habrá que utilizar una versión perteneciente a una serie de entregas en desarrollo. Hay que tener en cuenta que las entregas de desarrollo no son tan estables como las que están en producción.

Si lo que se desea es emplear el código fuente más actualizado disponible, que contenga todas las correcciones y esté libre de bugs, se debería emplear uno de los repositorios BitKeeper, que no son “entregas” propiamente dichas, pero es el código en el que se basarán las entregas futuras.

El esquema de denominaciones de MySQL emplea para las entregas nombres consistentes en tres números y un sufijo; por ejemplo, **mysql-5.0.9-beta**. Los números dentro del nombre de la entrega se interpretan como sigue:

- El primer número (**5**) es la versión principal y describe el formato de fichero. Todas las entregas de la versión 5 comparten el mismo formato para sus ficheros.
- El segundo número (**0**) es el nivel de entrega. En conjunto, la versión principal y el nivel de entrega constituyen el número de la serie.
- El tercer número (**9**) es el número de versión dentro de la serie. Se incrementa para cada nueva entrega. Usualmente es deseable poseer la última versión dentro de la serie que se está usando.

Para los cambios menores, el que se incrementa es el último número en la denominación de la versión. Cuando se adicionan características de importancia o aparecen incompatibilidades menores con versiones precedentes, se incrementa el segundo número. Cuando cambia el formato de los ficheros, se incrementa el primer número.

Las denominaciones de las entregas también incluyen un sufijo para indicar el grado de estabilidad. Una entrega progresa a través de un conjunto de sufijos a medida que mejora su estabilidad. Los posibles sufijos son:

- **alpha** indica que la entrega contiene características nuevas que no han sido plenamente probadas. Asimismo, en la sección "Novedades" deberían estar documentados los errores conocidos, aunque usualmente no los hay. Consulte Apéndice C, *Historial de cambios de MySQL*. Por lo general, en cada entrega alpha se implementan nuevos comandos y extensiones, y es la etapa donde puede producirse la mayor cantidad de cambios en el código. Sin embargo, debido a las pruebas realizadas, no deberían existir errores conocidos.
- **beta** significa que la entrega está destinada a poseer sus características completas y que se probó todo el código nuevo. No se agregan características de importancia, y no deberían existir errores críticos. Una versión cambia de alpha a beta cuando no se han descubierto errores fatales durante al menos un mes, y no hay planes de agregar características que pudieran comprometer la fiabilidad del código existente.

  Todas las APIs, las estructuras visibles externamente y las columnas para comandos SQL no se modificarán en las futuras entregas, sean beta, candidatas, o de producción.
- **rc** es una entrega candidata; o sea, una beta que ha estado funcionando un intervalo de tiempo y parece hacerlo bien. Solamente podrían ser necesarias correcciones menores. (Una entrega candidata es formalmente conocida como una entrega gamma.)

- Si no hay un sufijo, significa que la versión se ha estado utilizando por un tiempo en diferentes sitios sin que se informaran errores críticos reproducibles, más allá de los específicos de una plataforma. Esto es lo que se llama una entrega de producción (estable) o “General Availability” (GA).

MySQL utiliza un esquema de denominaciones ligeramente diferente a muchos otros productos. En general, se considera segura para usar una versión que ha durado un par de semanas sin ser reemplazada por una nueva dentro de la misma serie de entregas.

La totalidad de las entregas de MySQL se someten a pruebas de fiabilidad y rendimiento (estándares dentro de MySQL) para cerciorarse de que son relativamente seguras de utilizar. Puesto que las pruebas estándar son ampliadas cada vez para que incluyan todos los errores anteriormente descubiertos, el conjunto de pruebas se mejora continuamente.

Cada entrega se prueba al menos con:

- Un conjunto interno de pruebas

  El directorio `mysql-test` contiene un amplio conjunto de casos de prueba. En MySQL, prácticamente cada versión binaria del servidor pasa por estas pruebas. Consulte Sección 27.1.2, “El paquete de pruebas MySQL Test” para más información sobre este conjunto de pruebas.

- El conjunto de pruebas de rendimiento de MySQL

  Este conjunto ejecuta una serie de consultas comunes. Es también una manera de verificar que las últimas optimizaciones realizadas hacen efectivamente más rápido el código. Consulte Sección 7.1.4, “El paquete de pruebas de rendimiento (benchmarks) de MySQL”.

- La prueba `crash-me`

  Esta prueba intenta determinar las características soportadas por la base de datos y cuáles son sus capacidades y limitaciones. Consulte Sección 7.1.4, “El paquete de pruebas de rendimiento (benchmarks) de MySQL”.

Otra prueba consiste en utilizar la versión más reciente del servidor en el entorno de producción de MySQL, en al menos un ordenador. Se dispone de más de 100GB de datos para este fin.

2.1.2.2. Escoger un formato de distribución

Después de haber decidido qué versión de MySQL instalar, se debe elegir entre una distribución binaria o una de código fuente. Probablemente la elección más frecuente sea la distribución binaria, si existe una para la plataforma en cuestión. Hay distribuciones binarias disponibles en formato nativo para muchas plataformas, como los ficheros RPM para Linux, paquetes de instalación DMG para Mac OS X, y ficheros comprimidos Zip y `tar`.

Algunas razones a favor de la elección de una distribución binaria:

- Es más fácil de instalar que una distribución de código fuente.
- Para satisfacer distintos requerimientos de usuarios, se facilita dos versiones binarias diferentes: una que contiene motores de almacenamiento no transaccionales (más pequeña y rápida) y una configurada con las más importantes opciones, como por ejemplo tablas transaccionales. Ambas versiones se compilan a partir de la misma distribución de código fuente. Todos los clientes MySQL nativos pueden conectarse a ambas versiones indistintamente.

  La versión binaria extendida de MySQL está señalada con el sufijo `-max` y está configurada con las mismas opciones que `mysqld-max`. Consulte Sección 5.1.2, “El servidor extendido de MySQL `mysqld-max`”.

Si se desea utilizar `MySQL-Max` en formato RPM, primero debe instalarse el RPM de `MySQL-server` estándar.

Bajo ciertas circunstancias, puede ser mejor instalar MySQL a partir de una distribución de código fuente:

- Cuando se desea instalar MySQL en una ubicación especial. Las distribuciones binarias estándar están listas para ejecutarse en cualquier sitio, pero podria ser necesaria aún más flexibilidad en la elección de la ubicación de los componentes.
- Cuando se desea configurar `mysqld` con algunas características adicionales que no se encuentran incluídas en las distribuciones binarias estándar. La siguiente es una lista de las opciones adicionales más comunes:
  - `--with-innodb` (habilitado por defecto en todas las entregas binarias de la serie 5.0 de MySQL)
  - `--with-berkeley-db` (no está disponible en todas las plataformas)
  - `--with-libwrap`
  - `--with-named-z-libs` (en algunas distribuciones binarias ya está incluido)
  - `--with-debug[=full]`
- Cuando se desea excluir de `mysqld` algunas características presentes en las distribuciones binarias estándar. Por ejemplo, estas distribuciones se compilan normalmente con soporte para todos los conjuntos de caracteres. Si se deseara un servidor MySQL más liviano, se lo puede recompilar con soporte solamente para el conjunto de caracteres que se necesita.
- Cuando se posee un compilador especial (como `pgcc`) o se desea utilizar opciones de compilación optimizadas para un determinado procesador. Las distribuciones binarias se compilan con opciones que deberían funcionar en diversos procesadores de la misma familia.
- Cuando se desea emplear la última versión de código fuente desde un repositorio BitKeeper, para acceder a modificaciones recientes. Por ejemplo, si se detecta un error y se comunica al equipo de desarrollo de MySQL, la corrección se realiza sobre el código fuente, que queda almacenado en el repositorio. La primera entrega con esta corrección será la siguiente.
- Cuando se desea leer (o modificar) el código en C y C++ que conforma MySQL. Para este fin, se debería poseer una distribución de código fuente, ya que es la documentación más actualizada.
- Las distribuciones de código fuente contienen más pruebas y ejemplos que las distribuciones binarias.

2.1.2.3. Cómo y cuándo se entregan las actualizaciones

MySQL evoluciona con rapidez, y sus desarrolladores desean compartir el desarrollo con los usuarios. Se intenta producir una entrega cada vez que se incorpora nuevas características que pueden ser útiles para otros.

También se escucha a los usuarios que solicitan características sencillas de implementar. Se toma nota de lo que los usuarios con licencia desean, y especialmente de lo que solicitan los clientes de soporte, intentando actuar al respecto.

No es necesario descargar una entrega para conocer sus características, puesto que se puede dilucidar si una entrega posee determinada característica en la sección Novedades. Consulte Apéndice C, *Historial de cambios de MySQL*.

MySQL se rige por la siguiente política de actualizaciones:

- Las entregas se liberan dentro de cada serie. Para cada entrega, el último número en la versión es uno más que en la entrega anterior dentro de la misma serie.
- Las entregas de producción (estables) tienden a aparecer 1 o 2 veces por año. Sin embargo, de hallarse pequeños bugs, se libera una entrega con solamente correcciones.
- Las entregas de corrección para viejas entregas tienden a aparecer cada 4 u 8 semanas.
- De cada entrega principal Mysql AB realiza distribuciones binarias para algunas plataformas. Otros sujetos pueden realizar distribuciones binarias para otros sistemas, pero probablemente con menos frecuencia.
- Las correcciones están disponibles tan pronto se identifican errores pequeños o no críticos, pero que igualmente afectan al uso normal de MySQL. Se colocan en el repositorio público BitKeeper, y se incluyen en la entrega siguiente.
- Si por cualquier motivo se descubre un error fatal en una entrega, la política de MySQL es corregirlo mediante una nueva entrega, tan pronto como sea posible. (¡Y veríamos con agrado que otras compañías hicieran lo mismo!).

2.1.2.4. Filosofía de las entregas—No hay bugs conocidos en las entregas

Se dedica gran cantidad de tiempo y esfuerzo en producir entregas libres de errores. Que se tenga conocimiento, no se ha liberado una sola versión de MySQL con errores fatales reproducibles *conocidos*. (Un error “fatal” es uno que provoca la terminación abrupta de MySQL bajo condiciones de uso normales, que produce respuestas incorrectas para consultas normales, o que tiene problemas de seguridad).

Se han documentado todos los problemas, errores y cuestiones que dependen de decisiones de diseño. Consulte Sección A.8, “Problemas conocidos en MySQL”.

La intención de los desarrolladores es corregir todo lo que tenga solución sin afectar a la estabilidad de una versión estable de MySQL. En ciertos casos, esto significa que se puede corregir un error en las versiones en desarrollo, pero no en la versión estable (de producción). De todos modos estos errores se documentan para que los usuarios estén al tanto de ellos.

El proceso de desarrollo comprende las siguientes etapas:

- Se recolectan informes de errores desde la lista de soporte técnico, desde la base de datos de errores en http://bugs.mysql.com/, y desde las listas de correo externas.
- Todos los errores hallados en versiones con soporte, se introducen en la base de datos de errores.
- Cuando se corrige un error, se intenta crear un caso de prueba e incluirlo en el sistema de pruebas, para tener seguridad de que el error no vuelva a ocurrir sin ser detectado. (Cerca del 90% de los errores corregidos tienen un caso de prueba).
- Se crean casos de prueba para cada nueva característica que se agrega a MySQL.
- Antes de crear una entrega, se verifica que todos los errores reproducibles informados para esa versión de MySQL (3.23.x, 4.0.x, 4.1.x, 5.0.x, etc.) están solucionados. Si alguno no pudiera corregirse (debido a una decisión de diseño) esto se documenta en el manual. Consulte Sección A.8, “Problemas conocidos en MySQL”.
- Se hace una compilación para cada plataforma para la que se brinda una distribución binaria (más de 15) y se ejecutan pruebas de fiabilidad y rendimiento en todas ellas.
- No se publica una distribución binaria para una plataforma en la que fallaron las pruebas de fiabilidad o rendimiento. Si el problema se debe a un error en el código fuente, se resuelve, y para todas las plataformas se vuelve a compilar y probar.

- El proceso de compilación y prueba dura entre 2 y 3 días. Si durante el proceso se descubre un error fatal (por ejemplo, uno que genere un fichero de volcado del núcleo), se corrige el error y el proceso recomienza.
- Después de publicar la distribución binaria en http://dev.mysql.com/, se envía un mensaje con la novedad a las listas de correo `mysql` y `announce`. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”. El mensaje contiene una lista con todos los cambios y problemas conocidos que contiene la entrega. La sección **Known Problems** (problemas conocidos) solo ha sido necesaria en una pequeña cantidad de entregas.
- Para que los usuarios accedan rápidamente a las nuevas características de MySQL, se produce una entrega nuevo cada 4 a 8 semanas. El código fuente se prepara diariamente y se pone a disposición en http://downloads.mysql.com/snapshots.php.
- Si, a pesar de los esfuerzos realizados, se toma conocimiento de un error o problema crítico específicos de una plataforma después de que una entrega haya sido liberada, se genera una nueva entrega `'a'` con la corrección para la plataforma afectada. Gracias a la gran base de usuarios, cualquier problema se detecta y resuelve muy rápidamente.
- El trabajo del equipo de desarrollo en la generación de entregas estables es bastante bueno. De las últimas 150 entregas, se han debido rehacer menos de diez. En tres de estos casos, el error se debió a defectos en la biblioteca `glibc` en uno de los ordenadores de desarrollo, que llevó tiempo descubrir.

2.1.2.5. Binarios de MySQL compilados por MySQL AB

Uno de los servicios que MySQL AB ofrece es proveer un conjunto de distribuciones binarias compiladas en sistemas propios o amablemente proporcionados por adeptos de MySQL.

A parte de las distribuciones binarias provistas en formatos específicos de algunas plataformas, se ofrecen distribuciones binarias para una serie de plataformas en forma de ficheros comprimidos `tar` (ficheros `.tar.gz`). Consulte Sección 2.2, “Instalación MySQL estándar con una distribución binaria”.

Para distribuciones Windows, consulte Sección 2.3, “Instalar MySQL en Windows”.

Estas distribuciones se generan empleando el script `Build-tools/Do-compile`, que compila el código fuente y crea el fichero binario `tar.gz` empleando `scripts/make_binary_distribution`.

Estos binarios están configurados y compilados con los siguientes compiladores y opciones de compilación. Esta información también puede obtenerse observando las variables `COMP_ENV_INFO` y `CONFIGURE_LINE` dentro del script `bin/mysqlbug` de cada fichero de distribución binaria `tar`.

Los siguientes binarios se compilan en los sistemas de desarrollo de MySQL AB:

- Linux 2.4.xx x86 con `gcc` 2.95.3:

```
CFLAGS="-O2 -mcpu=pentiumpro" CXX=gcc CXXFLAGS="-O2 -mcpu=pentiumpro -
felide-constructors" ./configure --prefix=/usr/local/mysql --with-extra-
charsets=complex --enable-thread-safe-client --enable-local-infile --enable-
assembler --disable-shared --with-client-ldflags=-all-static --with-mysqld-
ldflags=-all-static
```

- Linux 2.4.x x86 con `icc` (compilador Intel C++ 8.1 o posterior):

```
CC=icc CXX=icpc CFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" CXXFLAGS="-O3
-unroll2 -ip -mp -no-gcc -restrict" ./configure --prefix=/usr/local/mysql --
localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --with-
extra-charsets=complex --enable-thread-safe-client --enable-local-infile --
```

```
enable-assembler --disable-shared --with-client-ldflags=-all-static --with-
mysqld-ldflags=-all-static --with-embedded-server --with-innodb
```

Obsérvese que las versiones 8.1 y posteriores del compilador Intel tienen drivers separados para C 'puro' (`icc`) y c++ (`icpc`); si se utiliza `icc` versión 8.0 o anterior para compilar MySQL, será necesario establecer `CXX=icc`.

- Linux 2.4.xx Intel Itanium 2 con `ecc` (Compilador Intel C++ Itanium 7.0):

```
CC=ecc CFLAGS="-O2 -tpp2 -ip -nolib_inline" CXX=ecc CXXFLAGS="-O2 -tpp2
-ip -nolib_inline" ./configure --prefix=/usr/local/mysql --with-extra-
charsets=complex --enable-thread-safe-client --enable-local-infile
```

- Linux 2.4.xx Intel Itanium con `ecc` (Compilador Intel C++ Itanium 7.0):

```
CC=ecc CFLAGS=-tpp1 CXX=ecc CXXFLAGS=-tpp1 ./configure --prefix=/usr/local/
mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-
local-infile
```

- Linux 2.4.xx alpha con `ccc` (Compaq C V6.2-505 / Compaq C++ V6.3-006):

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx CXXFLAGS="-fast -arch generic -
noexceptions -nortti" ./configure --prefix=/usr/local/mysql --with-extra-
charsets=complex --enable-thread-safe-client --enable-local-infile --with-
mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared --disable-shared
```

- Linux 2.x.xx ppc con `gcc` 2.95.4:

```
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 --localstatedir=/usr/local/mysql/data --libexecdir=/
usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-embedded-server --with-innodb
```

- Linux 2.4.xx s390 con `gcc` 2.95.3:

```
CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors" ./configure --
prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-
client --enable-local-infile --disable-shared --with-client-ldflags=-all-
static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.xx x86_64 (AMD64) con `gcc` 3.2.1:

```
CXX=gcc ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
```

- Sun Solaris 8 x86 con `gcc` 3.2.3:

```
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 --localstatedir=/usr/local/mysql/data --libexecdir=/
usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-innodb
```

- Sun Solaris 8 SPARC con `gcc` 3.2:

```
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 --enable-assembler --with-named-z-libs=no --
with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 8 SPARC 64-bit con `gcc` 3.2:

```
CC=gcc CFLAGS="-O3 -m64 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -m64
-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 --with-named-z-libs=no --with-named-
curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 SPARC con `gcc` 2.95.3:

```
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 --enable-assembler --with-named-curses-libs=-
lcurses --disable-shared
```

- Sun Solaris 9 SPARC con `cc-5.0` (Sun Forte 5.0):

```
CC=cc-5.0 CXX=CC ASFLAGS="-xarch=v9" CFLAGS="-Xa -xstrconst -mt -D_FORTEC_ -
xarch=v9" CXXFLAGS="-noex -mt -D_FORTEC_ -xarch=v9" ./configure --prefix=/
usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-z-libs=no --enable-
thread-safe-client --disable-shared
```

- IBM AIX 4.3.2 ppc con `gcc` 3.2.3:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many " CXX=gcc CXXFLAGS="-O2 -mcpu=powerpc
-Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./configure --
prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-
client --enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 4.3.3 ppc con `xlC_r` (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192" CXX=xlC_r
CXXFLAGS ="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192" ./configure --
prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/
usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --disable-shared --with-innodb
```

- IBM AIX 5.1.0 ppc con `gcc` 3.3:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2 -mcpu=powerpc
-Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./configure --
prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-
client --enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 5.2.0 ppc con `xlC_r` (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192" CXX=xlC_r
CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192" ./configure --prefix=/
usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/
local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --
```

```
enable-local-infile --with-named-z-libs=no --disable-shared --with-embedded-
server --with-innodb
```

- HP-UX 10.20 pa-risc1.1 con `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
```

- HP-UX 11.00 pa-risc con `aCC` (HP ANSI C++ B3910B A.03.50):

```
CC=cc CXX=aCC CFLAGS=+DAportable CXXFLAGS=+DAportable ./configure --prefix=/
usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/
local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --
enable-local-infile --disable-shared --with-embedded-server --with-innodb
```

- HP-UX 11.11 pa-risc2.0 64bit con `aCC` (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure --prefix=/usr/local/
mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-
local-infile --disable-shared
```

- HP-UX 11.11 pa-risc2.0 32bit con `aCC` (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./configure --
prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/
usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-innodb
```

- HP-UX 11.22 ia64 64bit con `aCC` (HP aC++/ANSI C B3910B A.05.50):

```
CC=cc CXX=aCC CFLAGS="+DD64 +DSitanium2" CXXFLAGS="+DD64 +DSitanium2" ./
configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex --enable-
thread-safe-client --enable-local-infile --disable-shared --with-embedded-
server --with-innodb
```

- Apple Mac OS X 10.2 powerpc con `gcc` 3.1:

```
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
```

- FreeBSD 4.7 i386 con `gcc` 2.95.4:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql --with-
extra-charsets=complex --enable-thread-safe-client --enable-local-infile --
enable-assembler --with-named-z-libs=not-used --disable-shared
```

- FreeBSD 4.7 i386 empleando LinuxThreads con `gcc` 2.95.4:

```
CFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT -D_THREAD_SAFE -
I/usr/local/include/pthread/linuxthreads" CXXFLAGS="-DHAVE_BROKEN_REALPATH
-D__USE_UNIX98 -D_REENTRANT -D_THREAD_SAFE -I/usr/local/include/pthread/
linuxthreads" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/
```

```
local/mysql/data --libexecdir=/usr/local/mysql/bin --enable-thread-safe-
client --enable-local-infile --enable-assembler --with-named-thread-libs="-
DHAVE_GLIBC2_STYLE_GETHOSTBYNAME_R -D_THREAD_SAFE -I /usr/local/include/
pthread/linuxthreads -L/usr/local/lib -llthread -llgcc_r" --disable-shared --
with-embedded-server --with-innodb
```

- QNX Neutrino 6.2.1 i386 con `gcc` 2.95.3qnx-nto 20010315:

```
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
```

Las siguientes distribuciones binarias están compiladas sobre sistemas de terceros, facilitados por otros usuarios a MySQL AB. Se facilitan solamente por cortesía, ya que MySQL AB no tiene control total sobre estos sistemas, por lo que sólo puede proporcionar un soporte limitado sobre las distribuciones compiladas en ellos.

- SCO Unix 3.2v5.0.7 i386 con `gcc` 2.95.3:

```
CFLAGS="-O3 -mpentium" LDFLAGS=-static CXX=gcc CXXFLAGS="-O3 -mpentium -
felide-constructors" ./configure --prefix=/usr/local/mysql --with-extra-
charsets=complex --enable-thread-safe-client --enable-local-infile --with-
named-z-libs=no --enable-thread-safe-client --disable-shared
```

- SCO UnixWare 7.1.4 i386 con `CC` 3.2:

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql --with-extra-
charsets=complex --enable-thread-safe-client --enable-local-infile --with-
named-z-libs=no --enable-thread-safe-client --disable-shared --with-readline
```

- SCO OpenServer 6.0.0 i386 con `CC` 3.2:

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql --with-extra-
charsets=complex --enable-thread-safe-client --enable-local-infile --with-
named-z-libs=no --enable-thread-safe-client --disable-shared --with-readline
```

- Compaq Tru64 OSF/1 V5.1 732 alpha con `cc/cxx` (Compaq C V6.3-029i / DIGITAL C++ V6.1-027):

```
CC="cc -pthread" CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed -
speculate all" CXX="cxx -pthread" CXXFLAGS="-O4 -ansi_alias -fast -inline
speed -speculate all -noexceptions -nortti" ./configure --prefix=/usr/local/
mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-
local-infile --with-named-thread-libs="-lpthread -lmach -lexc -lc" --disable-
shared --with-mysqld-ldflags=-all-static
```

- SGI Irix 6.5 IP32 con `gcc` 3.0.1:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" 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
```

- FreeBSD/sparc64 5.0 con `gcc` 3.2.1:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql --
localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --with-
```

```
extra-charsets=complex --enable-thread-safe-client --enable-local-infile --
disable-shared --with-innodb
```

Las siguientes opciones de compilación han sido empleadas en distribuciones binarias en el pasado. Estas distribuciones ya no reciben actualizaciones, pero las opciones de compilación se listan para referencia.

- Linux 2.2.xx SPARC con `egcs` 1.1.2:

  ```
  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 --enable-assembler --disable-shared
  ```

- Linux 2.2.x con x686 con `gcc` 2.95.2:

  ```
  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 --disable-shared --with-
  extra-charsets=complex
  ```

- SunOS 4.1.4 2 sun4c con `gcc` 2.7.2.1:

  ```
  CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors" ./configure --prefix=/usr/
  local/mysql --disable-shared --with-extra-charsets=complex --enable-assembler
  ```

- SunOS 5.5.1 (y posteriores) sun4u con `egcs` 1.0.3a o 2.90.27 o

  `gcc` 2.95.2 y posteriores: `CC=gcc CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-low-memory --with-extra-charsets=complex --enable-assembler`

- SunOS 5.6 i86pc con `gcc` 2.8.1:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql --with-low-
  memory --with-extra-charsets=complex
  ```

- BSDI BSD/OS 3.1 i386 con `gcc` 2.7.2.1:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql --with-
  extra-charsets=complex
  ```

- BSDI BSD/OS 2.1 i386 con `gcc` 2.7.2:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql --with-
  extra-charsets=complex
  ```

- AIX 4.2 con `gcc` 2.7.2.2:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql --with-
  extra-charsets=complex
  ```

Si alguien tiene opciones más efectivas para cualquiera de las configuraciones listadas, puede enviarlas por correo electrónico a la lista de correo MySQL `internals`. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”.

Las distribuciones RPM para entregas de MySQL 5.0 disponibles en el sitio web de MySQL están generadas por MySQL AB.

Si desea compilar una versión de depuración de MySQL, debería agregar `--with-debug` o `--with-debug=full` a los comandos `configure` anteriores, y quitar cualquier opción `-fomit-frame-pointer`.

2.1.3. Cómo obtener MySQL

Consulte la página de descargas de MySQL (http://dev.mysql.com/downloads/) para obtener información acerca de la versión más actualizada e instrucciones de descarga. Para obtener una lista actualizada de los sitios de replicación que también ofrecen descargas de MySQL, consulte http://dev.mysql.com/downloads/mirrors.html. Encontrará información acerca de cómo constituir un sitio de replicación y de cómo informar sobre un sitio de replicación que esté funcionando mal o esté desactualizado.

El principal sitio de replicación se encuentra en http://mirrors.sunsite.dk/mysql/.

2.1.4. Comprobar la integridad de paquetes con sumas de verificación MD5 o GnuPG

Después de descargar la distribución de MySQL que se adecúe a las necesidades del caso y antes de proceder a su instalación, se debería verificar su integridad. MySQL AB ofrece tres posibles formas de hacerlo:

- Sumas de verificación (checksums) MD5
- Firmas criptográficas empleando `GnuPG`, el GNU Privacy Guard.
- Para paquetes RPM, el mecanismo de verificación de integridad que incorporan estos paquetes.

Las siguientes secciones describen cómo emplear estos métodos.

Si se advierte que la suma de verificación MD5 o la firma GPG no coinciden, en primer lugar debe intentarse con una nueva descarga del paquete, quizá desde otro sitio de replicación. Si la verificación de la integridad del paquete fracasa repetidas veces, se debe notificar el incidente a MySQL AB, informando del nombre completo del paquete y del sitio de donde se descargó, a las direcciones `<webmaster@mysql.com>` o `<build@mysql.com>`. No debe utilizarse el sistema de informe de errores para comunicar problemas de descarga.

2.1.4.1. Comprobar la suma de verificación MD5

Después de haber descargado un paquete MySQL, se debería estar seguro de que su suma de verificación (checksum) MD5 concuerda con la provista en la página de descarga. Cada paquete tiene una suma de verificación individual, que se puede verificar mediante el siguiente comando, donde `package_name` es el nombre del paquete descargado:

```
shell> md5sum package_name
```

Ejemplo:

```
shell> md5sum mysql-standard-5.0.9-beta-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.0.9-beta-linux-i686.tar.gz
```

Se debería verificar que la suma de verificación resultante (la cadena de dígitos hexadecimales) concuerda con la que se muestra en la página de descargas inmediatamente debajo del paquete correspondiente.

Nota: lo que se debe comprobar es la suma de verificación del **fichero comprimido** (por ejemplo, el fichero `.zip` o `.tar.gz`) y no de los ficheros contenidos dentro del comprimido.

Hay que notar que no todos los sistemas operativos soportan el comando `md5sum`. En algunos, se llama simplemente `md5` y otros, directamente no lo poseen. En Linux forma parte del paquete **GNU Text Utilities**, que está disponible para una gran variedad de plataformas. El código fuente puede bajarse desde http://www.gnu.org/software/textutils/. Si OpenSSL está instalado, también puede emplearse el comando `openssl md5 package_name`. Una versión para DOS/Windows del comando `md5` se halla disponible en http://www.fourmilab.ch/md5/.

2.1.4.2. Verificación de firmas utilizando GnuPG

Otro método para verificar la integridad y autenticidad de un paquete es utilizar firmas criptográficas. Esta manera es más fiable que las sumas de verificación MD5, pero requiere más trabajo.

MySQL AB firma los paquetes de MySQL 5.0 con GnuPG (GNU Privacy Guard). GnuPG es una alternativa de código abierto frente a la conocida Pretty Good Privacy (PGP) de Phil Zimmermann. Consulte http://www.gnupg.org/ para más información acerca de GnuPG y de cómo obtenerlo e instalarlo en su sistema. La mayoría de las distribuciones Linux incluyen GnuPG instalado por defecto. Para mayor información acerca de GnuPG consulte http://www.openpgp.org/.

A fin de verificar la firma de un paquete específico, antes se debe obtener una copia de la clave pública para GPG de MySQL AB. Se puede descargar de http://www.keyserver.net/. La clave está identificada como `build@mysql.com`. Alternativamente, puede cortarse y copiarse la clave directamente del siguiente texto:

```
Key ID:
pub  1024D/5072E1F5 2003-02-03
     MySQL Package signing key (www.mysql.com) <build@mysql.com>
Fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5

Public Key (ASCII-armored):

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q7TXlTUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkgPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQUCPj6jDAUJCWYBgAULBwoDBAMVAwIDFgIBAheAAAoJEIxxjTtQ
cuH1cY4AnilUwTXn8MatQOiG0a/bPxrvK/gCAJ4oinSNZRYTnblChwFaazt7PF3q
zIhMBBMRAgAMBQI+PqPRBYMJZgC7AAoJEElQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/pECUmppiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRAgAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEAoIhpp6BozKI8p6eaabzF5MlJH58pAKCu/ROofK8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWsEN/l
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLmRDRi
Rjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8Ilbf5vSYHbuE5p
/1oIDznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+Lwqq
a8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOW
I39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKHt92
6s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZ
Whe70YGNPw1yjWJT1IhMBBgRAgAMBQI+PqMdBQkJZgGAAAoJEIxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFApmr1+0uUq/SlsQ==
```

```
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

Para incorporar esta clave dentro del GPG en uso, se emplea el comando `gpg --import`. Por ejemplo, si la clave estuviese guardada en un fichero llamado `mysql_pubkey.asc`, el comando de importación tomaría esta forma:

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

Debe consultarse la documentación de GPG para obtener más información sobre el manejo de claves públicas.

Después de haber descargado e importado la clave pública, debe descargarse el paquete MySQL deseado y la correspondiente firma, que también se encuentra en la página de descargas. El fichero de firma tiene el mismo nombre que el fichero de distribución, con una extensión `.asc`. Por ejemplo:

Fichero de distribución	`mysql-standard-5.0.9-beta-linux-i686.tar.gz`
Fichero de firma	`mysql-standard-5.0.9-beta-linux-i686.tar.gz.asc`

Se debe verificar que ambos ficheros se encuentran en el mismo directorio y entonces ejecutar el siguiente comando para verificar la firma del fichero de distribución:

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

Ejemplo:

```
  shell> gpg --verify mysql-standard-5.0.9-beta-linux-i686.tar.gz.asc
  gpg: Signature made Tue 12 Jul 2005 23:35:41 EST using DSA key ID 5072E1F5
  gpg: Good signature from "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

El mensaje `Good signature` indica que todo resultó correctamente. Puede ignorarse cualquier mensaje del tipo `insecure memory` que se obtenga.

2.1.4.3. Verificar firmas utilizando RPM

No existe una firma por separado para paquetes RPM. Estos paquetes tienen incorporadas la firma GPG y la suma de verificación MD5. Para verificar un paquete RPM se utiliza el siguiente comando:

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

Ejemplo:

```
shell> rpm --checksig MySQL-server-5.0.9-0.i386.rpm
MySQL-server-5.0.9-0.i386.rpm: md5 gpg OK
```

Nota: si está utilizando RPM 4.1 y emite mensajes de error del tipo `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, aun cuando se haya incorporado la clave pública dentro de las claves reconocidas por el GPG (keyring), se necesitará importar primero la clave pública dentro de las claves reconocidas (keyring) del RPM. RPM 4.1 ya no utiliza las claves reconocidas (keyring) personales (o GPG en sí mismo). En lugar de ello, mantiene su propio repositorio de claves (keyring) ya que constituye una aplicación a nivel de sistema, en tanto que el repositorio público de claves (keyring) de GPG es un fichero específico del usuario. Para importar la clave pública de MySQL dentro del repositorio de claves (keyring) del RPM, primero debe obtenerse la clave tal como se describe en la sección anterior. A continuación debe utilizarse `rpm --import` para importar la clave. Por ejemplo, si la clave pública se encuentra en un fichero llamado `mysql_pubkey.asc`, se importa utilizando el siguiente comando:

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

Para obtener la clave pública de MySQL, consulte: Sección 2.1.4.2, “Verificación de firmas utilizando GnuPG”.

2.1.5. Conformación de la instalación

Esta sección describe la conformación por defecto de los directorios creados por el instalador binario y por las distribuciones de código fuente provistas por MySQL AB. Si se instala una distribución obtenida de otro proveedor, esta conformación podría variar.

En MySQL 5.0 para Windows, el directorio de instalación por defecto es `C:\Program Files\MySQL\MySQL Server 5.0`. (Algunos usuarios de Windows prefieren realizar la instalación en el antiguo directorio por defecto, `C:\mysql`. De todos modos, la conformación de directorios permanece sin cambios). El directorio de instalación contiene los siguientes subdirectorios:

Directorio	Contenido
`bin`	Programas cliente y el servidor `mysqld`
`data`	Ficheros de registro (logs), bases de datos
`Docs`	Documentación
`examples`	Programas y scripts de ejemplo
`include`	Ficheros de inclusión
`lib`	Bibliotecas
`scripts`	Scripts de utilidades.
`share`	Ficheros con mensajes de error

Las instalaciones que se crean a partir de distribuciones RPM para Linux generadas por MySQL AB generan archivos bajo los siguientes directorios deL sistema:

Directorio	Contenido
`/usr/bin`	Programas cliente y scripts
`/usr/sbin`	El servidor `mysqld`
`/var/lib/mysql`	Ficheros de registro (logs), bases de datos
`/usr/share/doc/packages`	Documentación
`/usr/include/mysql`	Ficheros de inclusión
`/usr/lib/mysql`	Bibliotecas
`/usr/share/mysql`	Ficheros con mensajes de error y conjuntos de caracteres
`/usr/share/sql-bench`	Pruebas de rendimiento

En Unix, un fichero binario de distribución `tar` se instala descomprimiéndolo en la ubicación que se escoja para la instalación (generalmente `/usr/local/mysql`) y crea los siguientes directorios en dicha ubicación:

Directorio	Contenido
`bin`	Programas cliente y el servidor `mysqld`
`data`	Ficheros de registro (logs), bases de datos.
`docs`	Documentación, registro de modificaciones.

`include`	Ficheros de inclusión
`lib`	Bibliotecas
`scripts`	`mysql_install_db`
`share/mysql`	Ficheros con mensajes de error
`sql-bench`	Pruebas de rendimiento

Una distribución de código fuente se instala después de haberla configurado y compilado. Por defecto, la etapa de instalación crea ficheros bajo `/usr/local`, en los siguientes subdirectorios:

Directorio	Contenido
`bin`	Programas cliente y scripts
`include/mysql`	Ficheros de inclusión
`info`	Documentación en formato "Info"
`lib/mysql`	Bibliotecas
`libexec`	El servidor `mysqld`
`share/mysql`	Ficheros con mensajes de error
`sql-bench`	Pruebas de rendimiento y `crash-me`
`var`	Bases de datos y ficheros de registro (logs)

Dentro de su directorio de instalación, la conformación de una instalación de código fuente difiere de una binaria en los siguientes aspectos:

- El servidor `mysqld` se instala en el directorio `libexec` en lugar de en el directorio `bin`.
- El directorio para los datos es `var` en lugar de `data`.
- `mysql_install_db` se instala en el directorio `bin` en lugar de `scripts`.
- Los directorios de ficheros de inclusión y bibliotecas son `include/mysql` y `lib/mysql` en lugar de `include` y `lib`.

Se puede crear una instalación binaria propia a partir de una distribución de código fuente compilada si se ejecuta el script `scripts/make_binary_distribution` desde el directorio principal de la distribución de código fuente.

2.2. Instalación MySQL estándar con una distribución binaria

Las siguientes secciones cubren la instalación de MySQL en plataformas para las que se ofrecen paquetes que utilizan el formato de paquete de instalación respectivo de cada una. (Esto también es conocido como “instalación binaria”). Sin embargo, hay disponibles distribuciones binarias de MySQL para muchas otras plataformas. Consulte Sección 2.7, “Instalación de MySQL en otros sistemas similares a Unix” para encontrar instrucciones de instalación genéricas para estos paquetes que se aplican a todas las plataformas.

Consulte Sección 2.1, “Cuestiones generales sobre la instalación” para más información sobre otras distribuciones binarias y cómo conseguirlas.

2.3. Instalar MySQL en Windows

Desde la versión 3.21, MySQL AB proporciona una versión Windows nativa de MySQL, que representa un apreciable porcentaje de las descargas diarias de MySQL. Esta sección describe el proceso para instalar MySQL en Windows.

El instalador para la versión Windows de MySQL 5.0, en conjunto con un asistente de configuración dotado de interfaz gráfica, instala automáticamente MySQL, crea un fichero de opciones, inicia el servidor, y otorga seguridad a las cuentas de usuario por defecto.

Si se está actualizando una instalación existente de MySQL anterior a la versión 4.1.5, deben observarse los siguientes pasos:

1. Obtener e instalar la distribución.
2. Establecer un fichero de opciones, de ser necesario.
3. Seleccionar el servidor que se desea emplear.
4. Iniciar el servidor.
5. Colocar contraseñas a las cuentas MySQL creadas inicialmente.

Este proceso también debe realizarse con instalaciones nuevas de MySQL, cuando el paquete de instalación no incluya un instalador.

MySQL 5.0 para Windows está disponible en tres formatos de distribución:

- La distribución binaria contiene un programa de instalación que instala cada elemento necesario para iniciar el servidor inmediatamente.
- La distribución de código fuente contiene todo el código y ficheros de soporte para generar ejecutables utilizando el compilador de VC++ 6.0

Siempre que sea posible debería emplearse la distribución binaria. Es más simple que las otras y no se necesita ninguna herramienta adicional para poner en funcionamiento el servidor MySQL.

Esta sección explica cómo instalar MySQL para Windows utilizando una distribución binaria. Para realizar la instalación a partir de una distribución de código fuente, consulte Sección 2.8.6, “Instalar MySQL desde el código fuente en Windows”.

2.3.1. Requisitos de Windows

Para ejecutar MySQL para Windows, se necesita lo siguiente:

- Un sistema operativo Windows de 32 bits, tal como 9x, Me, NT, 2000, XP, o Windows Server 2003.

  Se recomienda fuertemente el uso de un sistema operativo Windows basado en NT (NT, 2000, XP, 2003) puesto que éstos permiten ejecutar el servidor MySQL como un servicio. Consulte Sección 2.3.12, “Arrancar MySQL como un servicio de Windows”.

- Soporte para protocolo TCP/IP.
- Una copia de la distribución binara de MySQL para Windows, que se puede descargar de http://dev.mysql.com/downloads/. Consulte Sección 2.1.3, “Cómo obtener MySQL”.

  Nota: Si se descarga la distribución a través de FTP, se recomienda el uso de un cliente FTP adecuado que posea la característica de reanudación (resume) para evitar la corrupción de ficheros durante el proceso de descarga.

- Una herramienta capaz de leer ficheros `.zip`, para descomprimir el fichero de distribución.
- Suficiente espacio en disco rígido para descomprimir, instalar, y crear las bases de datos de acuerdo a sus requisitos. Generalmente se recomienda un mínimo de 200 megabytes.

También podrían necesitarse los siguientes ítems opcionales:

- Si se planea conectarse al servidor MySQL a través de ODBC, se deberá contar con un driver Connector/ODBC. Consulte Sección 25.1, “MySQL Connector/ODBC”.
- Si se necesitan tablas con un tamaño superior a 4GB, debe instalarse MySQL en un sistema de ficheros NTFS o posterior. Al crear las tablas no debe olvidarse el uso de `MAX_ROWS` y `AVG_ROW_LENGTH`. Consulte Sección 13.1.5, “Sintaxis de `CREATE TABLE`”.

2.3.2. Elección de un paquete de instalación

En la versión 5.0 de MySQL hay tres paquetes de instalación para elegir cuando se instala MySQL para Windows. Son los siguientes:

- **El paquete Essentials**: Tiene un nombre de fichero similar a `mysql-essential-5.0.9-beta-win32.msi` y contiene los ficheros mínimamente necesarios para instalar MySQL en Windows, incluyendo el asistente de configuración. Este paquete no incluye componentes opcionales como el servidor incrustado (embedded) y el conjunto de pruebas de rendimiento (benchmarks).
- **El paquete Complete (Completo)**: Tiene un nombre de fichero similar a `mysql-5.0.9-beta-win32.zip` y contiene todos los archivos necesarios para una instalación completa bajo Windows, incluyendo el asistente de configuración. Este paquete incluye componentes opcionales como el servidor incrustado (embedded) y el conjunto de pruebas de rendimiento (benchmarks).
- **El Paquete Noinstall (Noinstall Archive)**: Tiene un nombre de fichero similar a `mysql-noinstall-5.0.9-beta-win32.zip` y contiene todos los ficheros contenidos en el paquete Complete, a excepción del asistente de configuración. Este paquete no incluye un instalador automatizado, y debe ser instalado y configurado manualmente.

El paquete Essentials es el recomendado para la mayoría de los usuarios.

El proceso de instalación que se siga depende del paquete de instalación escogido. Si se opta por instalar ya sea el paquete Complete o Essentials, consúltese Sección 2.3.3, “Instalación de MySQL con un instalador automático”. Si se opta por instalar MySQL a partir del paquete Noinstall, consúltese Sección 2.3.6, “Instalar MySQL partiendo de un archivo Zip Noinstall”.

2.3.3. Instalación de MySQL con un instalador automático

Los usuarios nuevos de MySQL 5.0 pueden emplear el asistente de instalación y el asistente de configuración para instalar MySQL en Windows. Éstos están diseñados para instalar y configurar MySQL de tal forma que los usuarios nuevos pueden comenzar a utilizar MySQL inmediatamente.

Los asistentes de instalación y configuración se encuentran disponibles en los paquetes Essentials y Complete, y están recomendados para la mayoría de las instalaciones estándar de MySQL. Las excepciones incluyen a usuarios que necesitan implementar múltiples instancias de MySQL en un único servidor y a usuarios avanzados que desean un control completo de la configuración del servidor.

2.3.4. Usar el asistente de instalación de MySQL

2.3.4.1. Introducción

El asistente de instalación es un instalador para el servidor MySQL que emplea las últimas tecnologías de instalador para Microsoft Windows. El Asistente de Instalación de MySQL, en combinación con el asistente de configuración, le permite a un usuario instalar y configurar un servidor MySQL que esté listo para el uso inmediatamente a continuación de la instalación.

El asistente de instalación MySQL es el instalador estándar para todas las distribuciones del Servidor MySQL 5.0. Los usuarios de versiones anteriores de MySQL deberán detener y desinstalar sus servidores existentes antes de realizar una instalación con el asistente de instalación de MySQL. Consulte

Sección 2.3.4.7, “Aumentar la versión MySQL” para más información acerca de actualizar una versión anterior.

Microsoft incluyó una versión mejorada de su Microsoft Windows Installer (Instalador de Microsoft Windows, MSI) en las versiones recientes de Windows. MSI se ha convertido en el estándar de facto para la instalación de aplicaciones bajo Windows 2000, Windows XP, y windows Server 2003. El asistente de instalación MySQL emplea esta tecnología para proporcionar un proceso de instalación más flexible y amigable.

El motor del instalador de Microsoft Windows fue actualizado en Windows XP; quienes utilicen una versión previa de Windows pueden remitirse a este artículo de la Base de Conocimiento de Microsoft para obtener información sobre cómo actualizar a la última versión de MSI.

Adicionalmente, Microsoft introdujo recientemente el conjunto de herramientas WiX (Instalador de Windows XML). Éste es el primer proyecto Open Source reconocido de Microsoft. Los desarrolladores de MySQL han optado por WiX porque es un proyecto Open Source y les permite manejar el proceso completo de instalación en Windows de una manera flexible, utilizando scripts.

Las mejoras al asistente de instalación MySQL dependen del soporte y comentarios recibidos de los usuarios. Si Usted descubre que el asistente de instalación está dejando de lado alguna característica que le resulte importante, o si halla un error, por favor emplee nuestro sistema de errores MySQL para solicitar características o informar sobre problemas.

2.3.4.2. Bajarse y arrancar el asistente de instalación de MySQL

Los paquetes de instalación del servidor MySQL pueden descargarse desde http://dev.mysql.com/downloads/. Si el paquete a descargar está contenido en un fichero ZIP, se deberá descomprimir el fichero antes.

El procedimiento para ejecutar el asistente de instalación depende del contenido del paquete descargado. Si existe un fichero `setup.exe` o `.msi`, al hacerles doble click comenzará la instalación.

2.3.4.3. Escoger un tipo de instalación

Hay disponibles tres tipos de instalación: **típica**, **completa**, y **personalizada**.

La instalación **típica** instala el servidor MySQL, el cliente de línea de comandos `mysql`, y las utilidades de línea de comandos. Los clientes y utilidades incluyen `mysqldump`, `myisamchk`, y otras herramientas que ayudan a administrar el servidor MySQL.

La instalación **completa** instala todos los componentes incluidos en el paquete. El paquete completo incluye componentes como el servidor incrustado (embedded), el conjunto de pruebas de rendimiento (benchmarks), scripts de mantenimiento, y documentación.

La instalación **personalizada** otorga un control completo sobre los paquetes que se desea instalar y el directorio de instalación que se utilizará. Consulte Sección 2.3.4.4, “La ventana de diálogo de instalación personalizada” para más información sobre cómo llevar a cabo una instalación personalizada.

Si se escoge la instalación **típica** o la **completa**, al hacer click sobre el botón Siguiente se avanza a la pantalla de confirmación para verificar las opciones y comenzar la instalación. Si se escoge la instalación **personalizada**, al hacer click sobre el botón Siguiente, se avanza al cuadro de diálogo de instalación personalizada, descrito en Sección 2.3.4.4, “La ventana de diálogo de instalación personalizada”

2.3.4.4. La ventana de diálogo de instalación personalizada

Si se desea cambiar el directorio de instalación o los componentes que se instalarán, se deberá elegir el tipo de instalación **personalizada**.

Todos los componentes disponibles se encuentran en un diagrama de árbol en el lado izquierdo del cuadro de diálogo de instalación personalizada. Los componentes que no serán instalados tienen un icono X rojo; los componentes que se instalarán tienen un icono gris. Para indicar si un componente se instalará o no, debe hacerse click en su icono y elegir una opción de la lista desplegable que aparece.

Se puede cambiar el directorio de instalación por defecto haciendo click en el botón Cambiar... a la derecha del directorio de instalación que se muestra.

Después de elegir los componentes a instalar y el directorio de instalación, hacer click en el botón Siguiente hará avanzar al cuadro de diálogo de confirmación.

2.3.4.5. La ventana de diálogo de confirmación

Una vez que se elige un tipo de instalación y los componentes a instalar, se avanza al cuadro de diálogo de confirmación. Se muestran el tipo y el directorio de instalación se para ser confirmados.

Para instalar MySQL una vez que se está conforme con la configuración, debe hacerse click en el botón Instalar. Para cambiar la configuración, debe hacerse click en el botón Retroceder. Para abandonar el asistente de instalación sin terminar la instalación de MySQL, debe hacerse click en el botón Cancel.

Una vez que la instalación está completa, se porporciona la opción de registrarse en el sitio web de MySQL. El registro otorga acceso para publicar mensajes en los foros de Mysql, en forums.mysql.com, junto con la posibilidad de informar errores en bugs.mysql.com y suscribirse al boletín electrónico. La pantalla final del instalador brinda un resumen de la instalación y la opción de ejecutar el asistente de configuración MySQL, que se utiliza para crear un fichero de configuración, instalar el servicio MySQL, y establecer la seguridad.

2.3.4.6. Cambios que realiza el asistente de instalación MySQL

Una vez que se hace click en el botón Instalar, el asistente de instalación MySQL comienza el proceso de instalación y realiza ciertos cambios en el sistema, que se describen en la siguiente sección.

Cambios al Registro

El asistente de instalación crea una clave de registro, durante una situación típica de instalación, localizada en `HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB`.

El asistente de instalación crea una clave cuyo nombre es el número de versión principal (número de la serie) del servidor que se encuentra instalado, tal como `MySQL Server 5.0`. Contiene dos valores de cadena, `Location` y `Version`. La cadena `Location` contiene el directorio de instalación. En una instalación corriente, contiene `C:\Program Files\MySQL\MySQL Server 5.0\`. La cadena `Version` contiene el número de entrega (release). Por ejemplo, para una instalación de MySQL Server 5.0.9 la clave contiene el valor `5.0.9`.

Estas claves del registro son útiles para ayudar a herramientas externas a identificar la ubicación en la que se instaló el servidor MySQL, evitando un rastreo completo del disco para descubrirla. Las claves del registro no son necesarias para la ejecución del servidor y no se crean cuando se usa el fichero Zip `noinstall`

Cambios en el menú Inicio

El asistente de instalación crea una nueva entrada en el menú Inicio de Windows, bajo una opción cuyo nombre es el número de versión principal (número de la serie) del servidor que se encuentra instalado. Por ejemplo, si se instala MySQL 5.0, se crea una sección MySQL Server 5.0 en el menú Inicio.

Se crean las siguientes entradas dentro de la nueva sección del menú Inicio:

- MySQL Command Line Client : Es un atajo al cliente de línea de comandos `mysql` y está configurado para iniciar sesión como usuario `root`. El atajo pregunta por una contraseña perteneciente a un usuario `root` cuando se conecta.
- MySQL Server Instance Config Wizard : Es un atajo al asistente de configuración. Utilice este atajo para configurar un servidor recientemente instalado o reconfigurar uno existente.
- MySQL Documentation : Es un vínculo a la documentación del servidor MySQL, que se almacena localmente en el directorio de instalación de MySQL. Esta opción no está disponible cuando el servidor MySQL fue instalado con el paquete Essentials.

Cambios en el sistema de ficheros

El asistente de instalación MySQL, por defecto instala el servidor MySQL en `C:\Program Files\MySQL\MySQL Server 5.0`, donde *`Program Files`* es el directorio de aplicaciones por defecto del sistema, y *`5.0`* es el número de versión principal (número de la serie) de servidor MySQL instalado. Ésta es la nueva ubicación donde se recomienda instalar MySQL, en sustitución de la antigua ubicación por defecto, `c:\mysql`.

Por defecto, todas las aplicaciones MySQL se almacenan en un directorio común localizado en `C:\Program Files\MySQL`, donde *`Program Files`* es el directorio de aplicaciones por defecto del sistema. Una instalación típica de MySQL en el ordenador de un desarrollador podría verse así:

```
C:\Program Files\MySQL\MySQL Server 5.0
C:\Program Files\MySQL\MySQL Administrator 1.0
C:\Program Files\MySQL\MySQL Query Browser 1.0
```

Esta proximidad entre los distintos directorios facilita la administración y el mantenimiento de todas las aplicaciones MySQL instaladas en un sistema en particular.

2.3.4.7. Aumentar la versión MySQL

El asistente de instalación MySQL puede llevar a cabo actualizaciones del servidor automáticamente, empleando la capacidad de actualización de MSI. Ello significa que no es necesario desinstalar manualmente una versión previa antes de instalar una nueva entrega (release). El instalador, automáticamente, detiene y quita el servicio MySQL antiguo antes de instalar la nueva versión.

Las actualizaciones automáticas se hallan disponibles solamente cuando se actualiza entre instalaciones que tienen el mismo número de versión principal (número de la serie). Por ejemplo, se puede hacer una actualización automática desde MySQL 4.1.5 a MySQL 4.1.6, pero no desde MySQL 4.1 a MySQL 5.0.

Consulte Sección 2.3.15, “Aumentar la versión de MySQL en Windows”.

2.3.5. Utilización del asistente de configuración

2.3.5.1. Introducción

El asistente de configuración MySQL automatiza el proceso de configurar el servidor bajo Windows. Crea un fichero `my.ini` personalizado, realizando una serie de preguntas y aplicando las respuestas a una plantilla para generar un fichero `my.ini` apropiado para la instalación en curso.

El asistente de configuración MySQL se incluye con la versión 5.0 de MySQL, y por el momento sólo está disponible para usuarios de Windows.

El asistente de configuración es en gran parte el resultado de los comentarios que MySQL AB recibió de muchos usuarios en un período de varios años. Sin embargo, si Usted descubre que el asistente de instalación está dejando de lado alguna característica que le resulte importante, o si halla un error, por favor emplee nuestro sistema de errores MySQL para solicitar características o informar sobre problemas.

2.3.5.2. Arrancar el asistente de configuración de MySQL

El asistente de configuración generalmente se ejecuta a continuación del asistente de instalación, ni bien éste finaliza. También puede iniciarse haciendo click en la entrada MySQL Server Instance Config Wizard de la sección MySQL del menú Inicio.

Adicionalmente, es posible dirigirse al directorio `bin` de la instalación MySQL y ejecutar directamente el fichero `MySQLInstanceConfig.exe`.

2.3.5.3. Escoger una opción de mantenimiento

Si el asistente de configuración MySQL detecta un fichero `my.ini` preexistente, se tiene la opción de reconfigurar el servidor o quitar la instancia borrando el fichero `my.ini` y deteniendo y quitando el servicio MySQL.

Para reconfigurar un servidor existente, debe escogerse la opción Re-configure Instance y hacer click en el botón Next. El fichero `my.ini` actual será renombrado como `my`*`timestamp`*`.ini.bak`, donde *`timestamp`* es la fecha y hora en que el fichero `my.ini` existente se creó. Para quitar la instancia del servidor actual, debe seleccionarse la opción Remove Instance y hacer click en el botón Next.

Si se selecciona la opción Remove Instance , se continúa con una ventana de confirmación. Al hacer click en el botón Execute, el asistente de configuración MySQL detendrá y quitará el servico MySQL, tras lo cual borrará el fichero `my.ini`. Los ficheros del servidor, incluyendo el directorio `data`, no se eliminarán.

Si se opta por Re-configure Instance , se continúa hacia el cuadro de diálogo Configuration Type donde puede elegirse el tipo de instalación a configurar.

2.3.5.4. Escoger un tipo de configuración

Cuando se inicia el asistente de configuración MySQL para una instalación nueva o se escoge la opción Re-configure Instance para una configuración existente, se avanza hacia el cuadro de diálogo Configuration Type.

Hay disponibles dos tipos de configuración: Configuración detallada (Detailed Configuration) y Configuración estándar (Standard Configuration) . La Configuración estándar está orientada a usuarios nuevos que deseen comenzar rápidamente con MySQL sin tener que tomar varias decisiones relativas a la configuración del servidor. La Configuración detallada está dirigida a usuarios avanzados que deseen un control más preciso sobre la configuración del servidor.

Si se trata de un usuario nuevo de MySQL y necesita un servidor configurado para un ordenador de desarrollo con un único usuario, la Configuración estándar debería cubrir sus necesidades. Al elegir la Configuración estándar el asistente de configuración MySQL establece todas las opciones de configuración automáticamente, a excepción de Opciones de servicio (Service options) y Opciones de seguridad (Security options) .

La Configuración estándar establece opciones que pueden ser incompatibles con sistemas donde existen instalaciones de MySQL previas. Si se posee una instalación de MySQL anterior además de la que se está configurando, se recomienda optar por la Configuración detallada (Detailed configuration)

Para completar la Configuración estándar , hay que remitirse a las secciones sobre Opciones de servicio (Service options) y Opciones de seguridad (Security options) , en Sección 2.3.5.11, “La ventana de diálogo de las opciones de servicio” y Sección 2.3.5.12, “La ventana de diálogo de las opciones de seguridad”, respectivamente.

2.3.5.5. La ventana de diálogo del tipo de servidor

Hay tres tipos de servidor distintos para elegir, y el tipo que se escoja afectará a las decisiones que el asistente de configuración MySQL tomará en relación al uso de memoria, disco y procesador.

- Developer machine (Ordenador de desarrollo) : Esta opción se aplica a ordenadores de escritorio donde MySQL está orientado a un uso personal solamente. Se asume que se estarán ejecutando varias otras aplicaciones, por lo que el servidor MySQL se configura para utilizar una cantidad mínima de recursos del sistema.
- Server machine (Servidor) : Esta opción se aplica a servidores donde MySQL se ejecuta junto con otras aplicaciones de servidor como son FTP, correo electrónico, y servidores web. MySQL se configura para utilizar una cantidad moderada de recursos del sistema.
- Dedicated MySQL Server Machine (Servidor MySQL dedicado) : Esta opción se aplica a ordenadores donde solamente se ejecuta el servidor MySQL. Se asume que no hay otras aplicaciones ejecutándose. El servidor MySQL se configura para utilizar todos los recursos disponibles en el sistema.

2.3.5.6. La ventana de diálogo Base de datos

El cuadro de diálogo Uso de la base de datos (Database usage) permite indicar los gestores de tablas que se planea utilizar al crear tablas de MySQL. La opción que se escoja determinará si el motor de almacenamiento `InnoDB` estará disponible y qué porcentaje de los recursos de servidor estarán disponibles para `InnoDB`

- Base de datos polifuncional (Multifunctional database) : Esta opción habilita tanto el motor de almacenamiento `InnoDB` como `MyISAM` y reparte los recursos uniformemente entre ambos. Se recomienda para usuarios que emplearán los dos motores de almacenamiento en forma habitual.
- Base de datos transaccional exclusiva (Transactional database only) : Esta opción habilita tanto el motor de almacenamiento `InnoDB` como `MyISAM`, pero destina más recursos del servidor al motor `InnoDB`. Se recomienda para usuarios que emplearán `InnoDB` casi exclusivamente, y harán un uso mínimo de `MyISAM`
- Base de datos no-transaccional exclusiva (Non-transactional database only) : Esta opción deshabilita completamente el motor de almacenamiento `InnoDB` y destina todos los recursos del servidor al motor `MyISAM`. Recomendado para usuarios que no utilizarán `InnoDB`.

2.3.5.7. La ventana de diálogo del espacio de tablas InnoDB

Algunos usuarios pueden querer ubicar los ficheros `InnoDB` en una ubicación diferente al directorio de datos del servidor MySQL. Esto puede ser deseable si el sistema tiene disponible un dispositivo de almacenamiento con mayor capacidad o mayor rendimiento, como un sistema RAID.

Para modificar la ubicación por defecto de los ficheros `InnoDB`, debe elegirse una nueva unidad de disco en la lista desplegable de letras de unidades y elegir una nueva ruta en la lista desplegable de rutas. Haciendo click en el botón ... podrá crearse una ruta personalizada,

Si se está modificando la configuración de un servidor preexistente, debe hacerse click en el botón Modify antes de cambiar la ruta. En dicho caso habrá que desplazar manualmente los ficheros InnoDB existentes hacia la nueva ubicación antes de iniciar el servidor.

2.3.5.8. La ventana de diálogo de conexiones concurrentes

Es importante establecer un límite para las conexiones simultáneas que se podrán establecer con el servidor MySQL, para evitar que éste se quede sin recursos. El cuadro de diálogo Conexiones simultáneas (Concurrent connections) permite indicar el uso que se planea darle al servidor, y establecer en consecuencia el límite de conexiones simultáneas. También es posible introducir manualmente el límite.

- Soporte de decisiones (Decision support (DSS)/OLAP) : Debe escogerse esta opción si el servidor no necesitará una gran cantidad de conexiones simultáneas. El número máximo de conexiones se establece en 100, asumiéndose un promedio de 20 conexiones simultáneas.

- Proceso de transacciones en línea (Online transaction processing (OLTP)) : Debe escogerse esta opción si el servidor necesitará un gran número de conexiones simultáneas. El número máximo de conexiones se establece en 500.
- Configuración manual (Manual setting) : Debe escogerse esta opción para establecer manualmente el número máximo de conexiones simultáneas que admitirá el servidor. El número deseado puede elegirse de una lista desplegable o teclearse si no figura en ella.

2.3.5.9. La ventana de diálogo de redes

El cuadro de diálogo Opciones de red (Networking options) permite activar o desactivar el protocolo TCP/IP y modificar el número de puerto por el que se accederá al servidor MySQL.

El protocolo TCP/IP está activado por defecto. Para desactivarlo debe quitarse la marca de la casilla al lado de la opción Activar TCP/IP (Enable TCP/IP networking)

Por defecto se utiliza el puerto 3306 para acceder a MySQL. Para modificar este valor, el número deseado puede elegirse de una lista desplegable o teclearse si no figura en la lista. Si el puerto indicado ya se encuentra en uso, se solicitará la confirmación de la elección.

2.3.5.10. La ventana de diálogo del conjunto de caracteres

El servidor MySQL soporta múltiples conjuntos de caracteres, y es posible establecer uno por defecto, que se aplicará a todas las tablas, columnas y bases de datos, a menos que se sustituya. Debe emplearse el cuadro de diálogo Character set para cambiar en el servidor el conjunto de caracteres por defecto.

- Juego de caracteres estándar (Standard character set) : Esta opción establecerá a `Latin1` como el juego de caracteres por defecto en el servidor. `Latin1` se usa para el Inglés y muchos idiomas de Europa Occidental.
- Soporte multilingüe mejorado (Best support for multilingualism) : Esta opción establece a `UTF8` como el conjunto de caracteres por defecto en el servidor. `UTF8` puede almacenar caracteres de muchos idiomas diferentes en un único juego.
- Selección manual del conjunto de caracteres por defecto / colación (Manual selected default character set / collation) : Esta opción se emplea cuando se desea elegir manualmente el juego de caracteres por defecto del servidor, a través de una lista desplegable.

2.3.5.11. La ventana de diálogo de las opciones de servicio

En plataformas basadas en Windows NT, el servidor MySQL puede instalarse como un servicio. De ese modo, se iniciará automáticamente durante el inicio del sistema, e incluso será reiniciado automáticamente por Windows en caso de producirse un fallo en el servicio.

El asistente de configuración MySQL instala por defecto el servidor MySQL como un servicio, utilizando el nombre de servicio `MySQL`. Si se desea evitar la instalación del servicio, debe vaciarse la casilla al lado de la opción Instalar como servicio Windows (Install as Windows service) . Se puede modificar el nombre del servicio eligiendo un nuevo nombre o tecleándolo en la lista desplegable provista.

Para instalar el servidor MySQL como un servicio pero que no se ejecute al iniciarse Windows, debe vaciarse la casilla al lado de la opción Ejecutar el servidor MySQL automáticamente (Launch the MySQL server automatically) .

2.3.5.12. La ventana de diálogo de las opciones de seguridad

Se recomienda fuertemente que se establezca una contraseña para el usuario `root` del servidor MySQL. El asistente de configuración MySQL la solicita por defecto. Si no se desea establecer una contraseña,

debe vaciarse la casilla al lado de la opción Modificar configuración de seguridad (Modify security settings) .

Para establecer la contraseña del usuario `root`, se debe introducir tanto en el cuadro de texto Nueva contraseña de root (New root password) como en Confirmar (Confirm) . Si se está reconfigurando un servidor existente, también será necesario introducir la contraseña en vigencia dentro del cuadro de texto Contraseña de root actual (Current root password) .

Para evitar que el usuario `root` inicie sesión desde cualquier punto de la red, debe marcarse la casilla al lado de la opción Root sólo puede conectarse en modo local (Root may only connect from localhost) . Esto fortalece la seguridad de la cuenta de `root`.

Para crear una cuenta de usuario anónimo, debe marcarse la casilla al lado de la opción Crear una cuenta de anónimo (Create An Anonymous Account) . No se recomienda crear un usuario anónimo porque puede disminuir la seguridad del servidor y ocasionar dificultades de inicio de sesión y de permisos.

2.3.5.13. La ventana de diálogo de confirmación

El último cuadro de diálogo del asistente de configuración MySQL es el de Confirmación (Confirmation dialog). Para concretar el proceso de configuración, debe hacerse click en el botón Ejecutar (Execute). Para volver a un cuadro de diálogo anterior, debe hacerse click en el botón Atrás (Back). Para abandonar el asistente de configuración sin cambiar la configuración del servidor, debe hacerse click en el botón Cancelar (Cancel).

Después de hacer click en el botón Ejecutar (Execute), el asistente de configuración MySQL llevará a cabo una serie de tareas cuyo avance se mostrará en la pantalla a medida que cada etapa termine.

El asistente de configuración MySQL determina en primer lugar las opciones del fichero de configuración, basándose en las preferencias del usuario, y empleando una plantilla confeccionada por desarrolladores e ingenieros de MySQL AB. Esta plantilla se llama `my-template.ini` y se localiza en el directorio de instalación del servidor.

Luego, el asistente de configuración guarda dichas opciones en el fichero `my.ini`. La ubicación final de este fichero se muestra al lado de la tarea **Guardar fichero de configuración (Write configuration file)**.

Si se optó por crear un servicio de Windows para el servidor MySQL, el asistente de configuración creará e iniciará el servicio. Si se está reconfigurando un servicio existente, el asistente de configuración reiniciará el servicio para que tomen efecto los cambios realizados.

Si se optó por establecer una contraseña para el usuario `root`, el asistente de configuración MySQL se conectará al servidor, establecerá la nueva contraseña para `root`, y aplicará cualquier otra opción de seguridad que se haya seleccionado.

Después de que el asistente de configuración MySQL haya completado sus tareas, se mostrará un resumen. Haciendo click en el botón Terminar (Finish) se abandonará el asistente.

2.3.5.14. Dónde está el fichero my.ini

El asistente de configuración MySQL coloca el fichero `my.ini` en el directorio de instalación del servidor MySQL. De este modo se asocian los ficheros de configuración con distintas instancias del servidor.

Para asegurarse de que el servidor MySQL sabe dónde buscar el fichero `my.ini`, durante la instalación del servicio se pasa al servidor un argumento similar a este: `--defaults-file="C:\Program Files\MySQL\MySQL Server 5.0\my.ini"`, donde *`C:\Program Files\MySQL\MySQL Server 5.0`* se reemplaza con la ruta de instalación del servidor MySQL.

`--defaults-file` le indica al servidor MySQL que lea el fichero especificado en busca de opciones de configuración.

2.3.5.15. Editar el fichero my.ini

Para modificar el fichero `my.ini`, se debe abrir con un editor de texto y realizar cualquier cambio necesario. También se puede modificar con la utilidad http://www.mysql.com/products/administrator/

Los programas cliente y las utilidades de MySQL, como el cliente de línea de comandos `mysql` y `mysqldump`, no son capaces de localizar el fichero `my.ini` ubicado en el directorio de instalación del servidor. Para configurar las aplicaciones cliente y de utilidades, debe crearse un nuevo fichero `my.ini` en el directorio `C:\Windows` o `C:\WINNT`, según corresponda a la versión de Windows que se esté ejecutando.

2.3.6. Instalar MySQL partiendo de un archivo Zip Noinstall

Los usuarios que hayan optado por instalar desde el paquete Noinstall, pueden servirse de las instrucciones en esta sección para instalar manualmente MySQL. El proceso para instalar MySQL desde un fichero ZIP es el siguiente:

1. Extraer el contenido del fichero dentro del directorio de instalación deseado.
2. Crear un fichero de opciones.
3. Elegir un tipo de servidor MySQL
4. Iniciar el servidor MySQL.
5. Establecer la seguridad de las cuentas de usuario por defecto.

El proceso completo se describe en las secciones siguientes.

2.3.7. Descomprimir el fichero de instalación

Para instalar MySQL manualmente, debe hacerse lo siguiente:

1. Si se está actualizando desde una versión anterior, se debe consultar Sección 2.3.15, “Aumentar la versión de MySQL en Windows” antes de comenzar el proceso de actualización.
2. Si se está utilizando un sistema operativo basado en Windows NT, como Windows NT, Windows 2000, Windows XP o Windows Server 2003, se debe iniciar sesión con un usuario con privilegios de administrador.
3. Debe elegirse una ubicación para la instalación. Tradicionalmente, el servidor MySQL se ha venido colocando en `C:\mysql`, y el asistente de instalación lo hace en `C:\Program Files\MySQL`. Si no se instala en `C:\mysql`, se debe indicar el directorio de instalación al iniciar el servidor o en un fichero de opciones. Consulte Sección 2.3.8, “Creación de un fichero de opciones”.
4. Utilizando una aplicación capaz de expandir ficheros comprimidos, se debe extraer el contenido del paquete dentro de la ubicación elegida para la instalación. Algunas aplicaciones extraen el contenido del fichero dentro de una carpeta que crean en la ubicación que se les indica. Si este es el caso, debe moverse el contenido de dicha subcarpeta y colocarlo en la ubicación elegida.

2.3.8. Creación de un fichero de opciones

Si es necesario especificarle opciones al servidor durante su inicio, esto puede hacerse desde la línea de comandos o bien colocando las opciones en un fichero de opciones. Aquellas opciones que se usarán cada vez que se inicie el servidor, es conveniente colocarlas en un fichero. Esto es especialmente cierto en las siguiente circunstancias:

- El directorio de instalación o de datos son diferentes de los usados por defecto (`C:\Archivos de Programa\MySQL\MySQL Server 5.0` y `C:\Archivos de Programa\MySQL\MySQL Server 5.0\data`).
- Es necesario afinar la configuración del servidor

Cuando el servidor MySQL para Windows se inicia, busca opciones en dos ficheros: en `my.ini` en el directorio de Windows, y en `C:\my.cnf`. El directorio de Windows generalmente es `C:\WINDOWS` o `C:\WINNT`. Se puede verificar el valor exacto consultando la variable de entorno `WINDIR` por medio del siguiente comando:

```
C:\> echo %WINDIR%
```

MySQL buscará opciones primero en el fichero `my.ini` y luego en `my.cnf`. Sin embargo, para evitar confusiones, es mejor emplear un solo fichero. Si el ordenador utiliza un gestor de arranque donde `C:` no es la unidad de inicio, la única opción será `my.ini`. Cualquiera que sea el fichero de opciones empleado, deberá estar en texto plano.

Otra posibilidad es utilizar como base los ficheros de opciones incluidos como ejemplo en la distribución de MySQL. Éstos se encuentran en el directorio de instalación y tienen nombres como `my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, y `my-huge.cnf`. Para utilizarlos como base de la configuración basta renombrarlos y copiarlos en la ubicación apropiada.

Un fichero de opciones puede crearse y modificarse con cualquier editor de textos, como el Bloc de Notas o Notepad. Por ejemplo, si MySQL está instalado en `E:\mysql` y el directorio de datos es `E:\mydata\data`, se puede crear un fichero de opciones que contenga una sección `[mysqld]` para especificar los valores que tendrán los parámetros `basedir` y `datadir`:

```
[mysqld]
# coloca en basedir el directorio de instalación
basedir=E:/mysql
# coloca en datadir el directorio de datos
datadir=E:/mydata/data
```

Debe tenerse en cuenta que las rutas de directorio, aun en Windows, deben escribirse en los ficheros de opciones con barras invertidas (/) en lugar de las habituales. Si se desea emplear estas últimas, deben colocarse en forma doble:

```
[mysqld]
# coloca en basedir el directorio de instalación
basedir=E:\\mysql
# coloca en datadir el directorio de datos
datadir=E:\\mydata\\data
```

En Windows, el instalador de MySQL coloca el directorio de datos directamente bajo el directorio donde se instala MySQL. Si se deseara tener el directorio de datos en una ubicación diferente, se debería copiar el contenido completo del directorio `data` en la nueva ubicación. Por ejemplo, si MySQL se instala en `C:\Program Files\MySQL\MySQL Server 5.0`, el directorio de datos estará por defecto en `C:\Program Files\MySQL\MySQL Server 5.0\data`. Si se quiere que el directorio de datos sea `E:\mydata` deben hacerse dos cosas:

1. Desplazar el directorio data y todo su contenido desde `C:\Program Files\MySQL\MySQL Server 5.0\data` hasta `E:\mydata`.
2. Emplear la opción `--datadir` para especificar la nueva ubicación del directorio data cada vez que se inicia el servidor.

2.3.9. Seleccionar un tipo de servidor MySQL

La siguiente tabla muestra los servidores MySQL 5.0 disponibles para Windows:

Ejecutable	Descripción
`mysqld-debug`	Compilado con el máximo de funciones de depuración y control automático de asignación de memoria, así como con soporte para tablas `InnoDB` y `BDB`.
`mysqld`	Ejecutable optimizado con soporte para `InnoDB`
`mysqld-nt`	Ejecutable optimizado para Windows NT, 2000, y XP con soporte para named pipes.
`mysqld-max`	Ejecutable optimizado con soporte para tablas `InnoDB` y `BDB`.
`mysqld-max-nt`	Similar a `mysqld-max`, pero compilado con soporte para named pipes.

Todos los ejecutables mencionados están optimizados para los modernos procesadores Intel, pero deberían funcionar en cualquier procesador Intel de tipo i386 o superior.

En MySQL 5.0, todos los servidores Windows tienen soporte para vínculo simbólico de directorios de bases de datos.

MySQL tiene soporte para TCP/IP en todas las plataformas Windows. Los servidores `mysqld-nt` y `mysql-max-nt` tienen soporte para named pipes en Windows NT, 2000, XP y 2003. Sin embargo, lo habitual es emplear TCP/IP sin tener en cuenta la plataforma. (Las named pipes son más lentas que TCP/IP en muchas configuraciones de Windows).

El uso de named pipes está sujeto a estas condiciones:

- Las named pipes están habilitadas solamente si se inicia el servidor con la opción `--enable-named-pipe`. Esto es necesario porque algunos usuarios han experimentado problemas al detener el servidor MySQL cuando las estaban utilizando.
- Las conexiones con named pipes están permitidas solamente en los servidores `mysqld-nt` o `mysqld-max-nt`, y siempre que la versión de Windows utilizada las soporte (Windows NT, 2000, XP, 2003).
- Estos servidores pueden ejecutarse en Windows 98 o Me, pero sólo si el protocolo TCP/IP está instalado; las conexiones con named pipe no pueden utilizarse.
- Estos servidores no pueden ejecutarse en Windows 95.

Nota: la mayor parte de los ejemplos de este manual emplean `mysqld` como nombre de servidor. Si se opta por emplear un servidor diferente, como `mysqld-nt`, deben hacerse los reemplazos de nombre adecuados en los comandos de los ejemplos.

2.3.10. Arrancar el servidor la primera vez

La información de esta sección se aplica principalmente si se está instalando MySQL con la versión `Noinstall`, o si se desea configurar y probar MySQL manualmente en lugar de usar las herramientas con interfaz gráfica.

En Windows 95, 98, o Me, los clientes MySQL siempre se conectan al servidor utilizando TCP/IP. (Esto le permite a cualquier ordenador de la red conectarse al servidor MySQL). Debido a esto, hay que asegurarse de que TCP/IP esté soportado en el ordenador antes de iniciar MySQL. El protocolo TCP/IP se encuentra en el CD-ROM de Windows.

Es importante advertir de que si se está utilizando una versión antigua de Windows 95 (por ejemplo, OSR2), es muy probable que se disponga de un paquete Winsock antiguo; MySQL necesita Winsock 2.

Puede descargarse el último paquete Winsock desde http://www.microsoft.com/. Windows 98 ya tiene la nueva biblioteca Winsock 2, de modo que no es necesario actualizarla.

En sistemas basados en NT, como Windows NT, 2000, XP, o 2003, los clientes tienen dos opciones. Pueden utilizar TCP/IP, o utilizar conexiones con named pipe si están soportadas por el servidor. Para lograr que MySQL trabaje con TCP/IP cuando se usa Windows NT 4, debe instalarse el service pack 3 (o posterior).

MySQL 5.0 para Windows también soporta conexiones de memoria compartida (shared-memory) si se lo inicia con la opción `--shared-memory`. Los clientes pueden conectarse a través de memoria compartida (shared memory) si usan la opción `--protocol=memory`.

Para más información sobre qué servidor ejecutar, consulte Sección 2.3.9, “Seleccionar un tipo de servidor MySQL”.

Esta sección brinda una visión de conjunto del arranque del servidor MySQL. Las siguientes secciones proporcionan información más específica para ejecutar el servidor MySQL desde la línea de comandos o como un servicio de Windows.

Los ejemplos de estas secciones asumen que MySQL está instalado en la ubicación por defecto: `C:\Program Files\MySQL\MySQL Server 5.0`. Las rutas de directorio mostradas en los ejemplos deben modificarse si MySQL está instalado en una ubicación diferente.

Las pruebas se realizan mejor desde el indicador del sistema en una ventana de consola (o “ventana DOS”). De este modo, los mensajes mostrados por el servidor permanecen en la ventana, donde son más sencillos de leer. Si algo funciona mal en la configuración, estos mensajes facilitan la identificación y solución de los problemas.

Para iniciar el servidor, se emplea este comando:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld --console
```

En servidores que incluyen soporte para `InnoDB`, se deberían mostrar los siguientes mensajes a medida que el servidor se inicia:

```
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
```

Cuando el servidor finaliza su secuencia de inicio, se debería ver un mensaje similar al siguiente, que indica que el servidor está listo para dar servicio a conexiones de clientes:

```
mysqld: ready for connections
Version: '5.0.9-beta'  socket: ''  port: 3306
```

El servidor continúa con la emisión por pantalla de cualquier otro mensaje de diagnóstico que se genere. Puede abrirse una nueva ventana de consola en la cual ejecutar programas cliente.

Si se omite la opción `--console`, el servidor dirige la información de diagnóstico hacia el registro de errores en el directorio de datos (por defecto, `C:\Program Files\MySQL\MySQL Server 5.0\data`). El registro de errores es el fichero con extensión `.err`.

Nota: Las cuentas de usuario que aparecen inicialmente en las tablas de permisos de MySQL no están protegidas por contraseña. Después de iniciar el servidor, se deberían establecer las contraseñas para estas cuentas empleando las instrucciones que se hallan en Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

2.3.11. Arrancar MySQL desde la línea de comandos de Windows

El servidor MySQL puede ser iniciado manualmente desde la línea de comandos. Esto es válido en cualquier versión de Windows.

Para iniciar el servidor `mysqld` desde la línea de comandos, se debería abrir una ventana de consola (o “ventana DOS ”) e ingresar este comando:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld
```

La ruta empleada en el ejemplo anterior puede variar según la ubicación de la instalación de MySQL en el sistema.

En versiones no NT de Windows, esto ejecutará `mysqld` en segundo plano. Esto significa que luego de que el servidor se inicia, puede verse otra ventana de comandos. Si se inicia el servidor de esta manera pero en Windows NT, 2000, XP o 2003, el mismo se ejecuta en segundo plano sin que aparezca ningún indicador del sistema hasta que el servidor finaliza. Debido a esto, se deberá abrir otra ventana de consola para correr programas cliente mientras el servidor se ejecuta.

El siguiente comando detendrá al servidor MySQL:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root shutdown
```

Esto invoca la utilidad administrativa de MySQL, `mysqladmin`, para conectarse al servidor y transmitirle la orden de finalización. El comando se conecta como el usuario `root` de MySQL, el cual es la cuenta administrativa por defecto en el sistema de permisos de MySQL. Debe advertirse que los usuarios en este sistema son enteramente independientes de cualquier usuario de inicio de sesión perteneciente a Windows.

Si `mysqld` no se inicia, debe verificarse el registro de errores para ver si el servidor generó cualquier mensaje que indique la causa del problema. El registro de errores se localiza en el directorio `C:\Program Files\MySQL\MySQL Server 5.0\data`. Es el fichero con extensión `.err`. También puede intentarse iniciar el servidor con el comando `mysqld --console`; en este caso se podrá obtener alguna información en pantalla que permita resolver el problema.

La última opción es ejecutar `mysqld` con `--standalone --debug`. En este caso, `mysqld` guardará un fichero de registro llamado `C:\mysqld.trace` el cual debería contener la razón por la cual `mysqld` no se inicia. Consulte Sección D.1.2, “Crear ficheros de traza”.

El comando `mysqld --verbose --help` sirve para mostrar todas las opciones que `mysqld` es capaz de comprender.

2.3.12. Arrancar MySQL como un servicio de Windows

En la familia NT (Windows NT, 2000, XP, 2003), la manera recomendada de ejecutar MySQL es instalarlo como un servicio del sistema operativo, de modo que se inicie y detenga automáticamente cuando

Windows lo haga. Un servidor MySQL instalado como servicio también puede controlarse desde la línea de comandos empleando los comandos `NET`, o con la utilidad gráfica `Services`.

La utilidad `Services` (el `Administrador de Servicios de Windows (Service Control Manager)`) puede encontrarse en el Panel de Control (bajo Administrative Tools en Windows 2000, XP, y Server 2003). Es aconsejable cerrar la utilidad `Services` mientras se lleven a cabo operaciones de instalación o remoción del servidor desde la línea de comandos. Esto evita una cantidad de errores.

Antes de instalar MySQL como un servicio Windows, se debería detener primero el servidor -si está en ejecución- mediante el siguiente comando:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root shutdown
```

Nota: si la cuenta de usuario MySQL `root` está protegida por una contraseña, la forma de invocar este comando será `C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root -p shutdown` y porporcionando la contraseña cuando sea solicitada.

Esto invoca la utilidad administrativa de MySQL, `mysqladmin`, para conectarse al servidor y transmitirle la orden de finalización. El comando se conecta como el usuario `root` de MySQL, el cual es la cuenta administrativa por defecto en el sistema de permisos de MySQL. Debe advertirse que los usuarios en este sistema son enteramente independientes de cualquier usuario de inicio de sesión perteneciente a Windows.

Este comando instalará el servidor como un servicio:

```
C:\> mysqld --install
```

Si se producen problemas al instalar `mysqld` como un servicio usando sólo el nombre del servidor, debe intentarse indicando la ruta completa. Por ejemplo:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld --install
```

La ruta al directorio `bin` de MySQL puede agregarse a la variable de entorno de Windows `PATH`:

- En el Escritorio de Windows, hacer click derecho en el ícono Mi PC y seleccionar Propiedades
- A continuación, seleccionar la pestaña Opciones Avanzadas de la ventana Propiedades del Sistema, y hacer click en el botón Variables de Entorno.
- Bajo **Variables del Sistema**, seleccionar Path , y hacer click en el botón Modificar. Aparecerá el cuadro de diálogo Modificar Variable del Sistema.
- Debe colocarse el cursor al final del texto mostrado en el espacio denominado **Valor de la Variable**. (Presionando la tecla **Fin (End)** se puede tener seguridad que el cursor quede realmente al final del texto.) Luego debe ingresarse la ruta completa al directorio `bin` de MySQL (por ejemplo, `C:\Program Files\MySQL\MySQL Server 5.0\bin`). Si había un texto anterior, debe haber un punto y coma separando aquel y esta nueva ruta. Cerrar todos los cuadros de diálogo haciendo click en OK. Ahora debería poderse invocar cualquier programa ejecutable de MySQL simplemente tipeando su nombre en el indicador de sistema desde cualquier directorio, sin tener que indicar la ruta completa. Esto incluye a los servidores, el cliente `mysql`, y todas las utilidades de línea de comandos tal como `mysqladmin` y `mysqldump`.
- No se debería agregar el directorio `bin` de MySQL al `PATH` de Windows si se están ejecutando múltiples servidores MySQL en el mismo ordenador.

Advertencia: debe tenerse mucho cuidado al editar manualmente la variable de sistema `PATH`; el borrado o modificación accidental de cualquier parte podría dejar al sistema funcionando mal o incluso inutilizable.

El comando de instalación como servicio no inicia el servidor. Las instrucciones para hacerlo se dan luego en esta sección.

MySQL 5.0 soporta argumentos adicionales cuando se lo instala como servicio:

- Puede indicarse un nombre para el servicio inmediatamente a continuación de la opción `--install`. El nombre por defecto es `MySQL`.
- Si se indica un nombre de servicio, solamente puede especificarse una opción a continuación. Por convención, esta debería ser `--defaults-file=file_name` para indicar el nombre de un fichero de opciones que el servidor debería leer cuando se inicia.

  Es posible emplear otra opción en vez de `--defaults-file`, pero no se recomienda. `--defaults-file` es más flexible porque posibilita especificar múltiples opciones de inicio para el servidor, colocándolas en el fichero indicado. Además, en MySQL 5.0, el uso de una opción diferente a `--defaults-file` no está soportado hasta la versión 5.0.3.
- A partir de la versión 5.0.1, puede especificarse la opción `--local-service` a continuación del nombre del servicio. Esto provoca que el servidor se ejecute empleando la cuenta `LocalService` de Windows, que tiene privilegios de sistema limitados. Esta cuenta existe solamente en Windows XP y posteriores. Si ambas opciones `--defaults-file` y `--local-service` son colocadas a continuación del nombre del servicio, pueden estar en cualquier orden.

Para un servidor MySQL que se instaló como un servicio de Windows, las siguientes reglas determinan el nombre de servicio y los ficheros de opciones que utilizará:

- Si el comando de instalación como servicio no especificó un nombre de servicio o el nombre por defecto (`MySQL`) a continuación de la opción `--install`, el servidor tomará el nombre de servicio `MySQL` y leerá opciones desde el grupo `[mysqld]` en los ficheros de opciones estándar.
- Si el comando de instalación como servicio especifica un nombre de servicio distinto a `MySQL` luego de la opción `--install`, el servidor empleará ese nombre de servicio. Leerá opciones en el grupo que tenga el mismo nombre que el servicio, en los ficheros de opciones estándar.

  El servidor también leerá opciones desde el grupo `[mysqld]` de los ficheros de opciones estándar. Esto permite usar el grupo `[mysqld]` para opciones que deban ser utilizadas por todos los servicios MySQL, y un grupo de opciones con el mismo nombre del servicio para ser usadas sólo por aquel.
- Si el comando de instalación del servicio especifica una opción `--defaults-file` después del nombre del servicio, el servidor leerá opciones solamente desde el grupo `[mysqld]` del fichero suministrado e ignorará los ficheros de opciones estándar.

A modo de un ejemplo más complejo, considérese el siguiente comando:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld --install MySQL --defaults-file=C:\my-opts.cnf
```

Aquí, el nombre de servicio por defecto (`MySQL`) se suministró a continuación de la opción `--install`. Si no se hubiera indicado la opción `--defaults-file`, este comando hubiese tenido como efecto que el servidor leyera el grupo `[mysqld]` de los ficheros de opciones estándar. No obstante, debido a que la opción `--defaults-file` se encuentra presente, el servidor leerá las opciones del grupo `[mysqld]`, pero sólo del fichero indicado.

También es posible especificar opciones como Parámetros de Inicio (Start parameters) en la utilidad `Services` de Windows antes de iniciar el servicio MySQL.

Una vez que el servidor MySQL ha sido instalado como servicio, será iniciado automáticamente luego del arranque de Windows. El servicio también puede iniciarse desde la utilidad `Services`, o empleando el comando `NET START MySQL`. El comando `NET` no es case sensitive.

Cuando se ejecuta como servicio, `mysqld` no tiene acceso a una ventana de consola, por lo que no puede mostrar mensajes. Si `mysqld` no se inicia, debe consultarse el registro de errores para ver si el servidor ha dejado allí mensajes que indiquen la causa del problema. El registro de errores se encuentra en el directorio de datos de MySQL (por ejemplo, `C:\Program Files\MySQL\MySQL Server 5.0\data`). Es el fichero con extensión `.err`.

Cuando un servidor MySQL se instala como servicio, se detendrá automáticamente si estaba en ejecución al momento de cerrar Windows. También puede detenerse manualmente, ya sea a través de la utilidad `Services`, del comando `NET STOP MySQL`, o del comando `mysqladmin shutdown`.

También existe la opción de instalar el servidor como un servicio de inicio manual, si no se desea que el servicio se inicie en cada arranque de Windows. Para esto, debe emplearse la opción `--install-manual` en lugar de `--install`:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld --install-manual
```

Para cancelar un servidor que fue instalado como servicio, primero se lo debe detener, si está en ejecución, por medio del comando `NET STOP MYSQL`. Luego de esto se usará la opción `--remove` para cancelarlo:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld --remove
```

Si `mysqld` no se está ejecutando como un servicio, se lo puede iniciar desde la línea de comandos. Consulte Sección 2.3.11, “Arrancar MySQL desde la línea de comandos de Windows” para más instrucciones.

Consulte Sección 2.3.14, “Resolución de problemas en la instalación de MySQL bajo Windows” si se producen problemas durante la instalación.

2.3.13. Comprobar la instalación de MySQL Installation

Cualquiera de los siguientes comandos permitirá comprobar si el servidor MySQL está en funcionamiento:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow -u root mysql
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin version status proc
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql test
```

Si `mysqld` responde con lentitud a las conexiones TCP/IP provenientes de programas cliente, probablemente haya un problema con el DNS. En este caso, hay que iniciar `mysqld` con la opción `--skip-name-resolve` y utilizar solamente `localhost` y números de IP en la columna `Host` de las tablas de permisos de MySQL.

Puede forzarse a un cliente MySQL a utilizar una conexión named pipe en lugar de TCP/IP especificando la opción `--pipe` o `--protocol=PIPE`, o indicando `.` (punto) como nombre de host. La opción `--socket` se utilizará para especificar el nombre del pipe.

2.3.14. Resolución de problemas en la instalación de MySQL bajo Windows

Cuando se instala y ejecuta MySQL por primera vez, es posible encontrar ciertos errores que evitan el inicio del servidor. El propósito de esta sección es brindar auxilio en el diagnóstico y corrección de algunos de estos errores.

El primer recurso a considerar durante la resolución de problemas en el servidor es el registro de errores. El servidor MySQL utiliza este registro para guardar información relevante acerca del error que está impidiendo su inicio. El registro de errores se encuentra en el directorio de datos especificado en el

fichero `my.ini`. La ubicación por defecto es `C:\Program Files\MySQL\MySQL Server 5.0\data`. Consulte Sección 5.10.1, “El registro de errroes (Error Log)”.

Otra fuente de información relativa a posibles errores son los mensajes mostrados en la consola cuando el servicio MySQL se está iniciando. Empleando el comando `NET START mysql` en la línea de comandos luego de instalar `mysqld` como un servicio, se podrá ver cualquier mensaje de error relativo al inicio del servicio MySQL. Consulte Sección 2.3.12, “Arrancar MySQL como un servicio de Windows”.

A continuación se brindan ejemplos de algunos de los más comunes errores que pueden ocurrir cuando se instala MySQL y se inicia el servidor por primera vez:

- ```
  System error 1067 has occurred.
  Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
  ```

  Este mensaje se emite cuando el servidor MySQL no puede encontrar la base de datos `mysql` u otros ficheros vitales para su funcionamiento. A menudo sucede cuando el directorio base o el directorio de datos de MySQL se instalan en ubicaciones distintas a las predeterminadas (`C:\mysql` y `C:\Program Files\MySQL\MySQL Server 5.0\data`, respectivamente).

  Una situación en la que puede ocurrir esto es cuando se instala una actualización de MySQL en una nueva ubicación, pero el fichero de configuración no se modifica para reflejar el nuevo directorio. Accesoriamente puede suceder que haya ficheros de configuración antiguos y nuevos en conflicto. Al actualizar MySQL hay que asegurarse de borrar o renombrar los ficheros de configuración existentes.

  Si se ha instalado MySQL en un directorio diferente a `C:\Program Files\MySQL\MySQL Server 5.0` es necesario asegurarse de que el servidor MySQL está al tanto de esto a través del uso de un fichero de configuración (`my.ini`). El fichero `my.ini` debe estar localizado en el directorio de Windows, generalmente `C:\WINNT` o `C:\WINDOWS`. Se puede determinar su ubicación exacta a partir de la variable de entorno `WINDIR` si se ordena lo siguiente en la línea de comandos:

  ```
  C:\> echo %WINDIR%
  ```

  Cualquier editor de texto, como Notepad, sirve para crear y modificar un fichero de opciones. Por ejemplo, si MySQL se instala en `E:\mysql` y el directorio de datos es `D:\MySQLdata`, se puede crear el fichero de opciones y establecer una sección llamada `[mysqld]` para indicar los valores de los parámetros basedir y datadir:

  ```
  [mysqld]
  # Coloca en basedir el directorio de instalación
  basedir=E:/mysql
  # Coloca en datadir el directorio de datos
  datadir=D:/MySQLdata
  ```

  Debe tenerse en cuenta que las rutas de directorio, aún en Windows, deben escribirse en los ficheros de opciones con barras invertidas (/) en lugar de las habituales. Si se desea emplear estas últimas, deben colocarse en forma doble:

  ```
  [mysqld]
  # Coloca en basedir el directorio de instalación
  basedir=C:\\Program Files\\MySQL\\MySQL Server 5.0
  # Coloca en datadir el directorio de datos
  datadir=D:\\MySQLdata
  ```

  Consulte Sección 2.3.8, “Creación de un fichero de opciones”.

- ```
  ```

```
Error: Cannot create Windows service for MySql. Error: 0
```

Este error ocurre cuando se reinstala o actualiza MySQL utilizando el Asistente de Configuración y sin detener y quitar primero el servicio MySQL existente. Sucede debido a que cuando el Asistente de Configuración intenta instalar el servicio, halla el anterior con el mismo nombre.

Una solución es escoger un nombre de servicio diferente a `mysql` cuando se emplea el Asistente de Configuración. Esto le permitirá al nuevo servicio instalarse correctamente, pero aún seguirá existiendo el servicio anterior. Aunque no es nocivo, es mejor remover los servicios que no están en uso.

Para quitar permanentemente el antiguo servicio mysql, debe emplearse el siguiente comando en la línea de comandos, dentro de un usuario que tenga privilegios de administrador:

```
C:\>sc delete mysql
[SC] DeleteService SUCCESS
```

Si la versión de Windows que se está utilizando no posee la utilidad `sc`, puede descargarse la utilidad `delsrv` desde http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp y utilizarla con la sintaxis `delsrv mysql`.

2.3.15. Aumentar la versión de MySQL en Windows

Esta sección detalla algunos pasos a seguir cuando se actualiza MySQL para Windows.

1. Siempre debería hacerse una copia de respaldo de la instalación de MySQL en uso antes de llevar a cabo una actualización. Consulte Sección 5.8.1, “Copias de seguridad de bases de datos”.

2. Descargar la última distribución de MySQL para Windows desde http://dev.mysql.com/downloads.

3. Antes de actualizar MySQL, debe detenerse el servidor.

   Si el servidor está instalado como servicio de Windows, debe detenerse el servicio ingresando lo siguiente en la línea de comandos:

   ```
   C:\> NET STOP MYSQL
   ```

   Si no se está ejecutando el servidor MySQL como un servicio de Windows, debe detenerse el servidor con el siguiente comando:

   ```
   C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root shutdown
   ```

4. Cuando se actualiza a MySQL 5.0 desde una versión anterior de la 4.1.5 o bien cuando se actualiza desde una versión instalada desde un fichero zip a otra que utiliza el Asistente de Instalación MySQL, se debe quitar manualmente la instalación anterior, incluyendo el servicio MySQL (si el server se hubiera instalado como servicio de Windows).

   Para quitar el servicio MySQL, debe utilizarse el siguiente comando:

   ```
   C:\> C:\mysql\bin\mysqld --remove
   ```

   Si no se quita el servicio existente, el Asistente de Instalación MySQL puede fallar al instalar el nuevo servicio MySQL.

5. Si se está empleando el Asistente de Instalación MySQL, debe iniciarse el asistente como se indica en Sección 2.3.4, “Usar el asistente de instalación de MySQL”.

6. Si se está instalando MySQL desde un fichero Zip, debe descomprimirse el fichero. Durante la operación puede sobreescribirse la instalación actual de MySQL (generalmente localizada en `C:\mysql`), o instalar la nueva versión en una ubicación diferente, como `C:\mysql5`. Se recomienda sobreescribir la instalación existente.

7. Reiniciar el servidor. Por ejemplo, usando el comando `NET START MySQL` si se ejecutará MySQL como un servicio, o en otro caso, invocar directamente el comando `mysqld`.

8. Consulte Sección 2.10, “Aumentar la versión de MySQL” para obtener información adicional (no limitada a Windows) sobre la actualización de MySQL.

9. Si se producen errores, consulte Sección 2.3.14, “Resolución de problemas en la instalación de MySQL bajo Windows”.

2.3.16. Comparación entre MySQL en Windows y MySQL en Unix

MySQL para Windows ha demostrado por sí mismo ser muy estable. La versión para Windows de MySQL tiene las mismas características que su contraparte Unix, con las siguientes excepciones:

- **Windows 95 y los subprocesos**

  Windows 95 pierde cerca de 200 bytes de memoria principal por cada vez que crea un subproceso. Cada conexión en MySQL crea un nuevo subproceso, de modo que no se debería ejecutar `mysqld` por un período prolongado de tiempo, en Windows 95, si el servidor va a gestionar muchas conexiones. Otras versiones de Windows no presentan este inconveniente.

- **Cantidad limitada de puertos**

  Los sistemas Windows tienen alrededor de 4.000 puertos disponibles para conexiones de clientes, y luego de que una conexión se cierra, el puerto demora entre dos y cuatro minutos en liberarse. En situaciones donde los clientes se conecten y desconecten del servidor frecuentemente, es posible que todos los puertos disponibles se utilicen antes de que los puertos cerrados sean utilizables de nuevo. Si esto ocurre, el servidor MySQL no responderá aun cuando se esté ejecutando. Debe tenerse en cuenta que los puertos pueden ser usados por otras aplicaciones que se ejecuten en el mismo ordenador, en cuyo caso la cantidad de puertos disponibles para MySQL será menor que lo mencionado.

  Para más información, consulte http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **Lecturas simultáneas**

  MySQL depende de las llamadas del sistema `pread()` y `pwrite()` para ser capaz de mezclar `INSERT` y `SELECT`. Actualmente se usan mutexes para emular `pread()` y `pwrite()`. Se planea reemplazar en un futuro la interfaz a nivel de ficheros con una interfaz virtual, de modo que se pueda utilizar la interfaz `readfile()`/`writefile()` en Windows NT, 2000 y XP, para obtener más velocidad. La implementación actual limita a 2.048 el número de ficheros abiertos que MySQL 5.0 puede usar, lo cual significa que no se pueden abrir tantos procesos simultáneos en Windows NT, 2000, XP y 2003 como en Unix.

- **Bloqueo de lectura**

  MySQL utiliza un bloqueo de lectura por cada conexión, lo cual tiene los siguientes efectos cuando están habilitadas conexiones named pipe:

  - Una conexión no es desconectada automáticamente luego de ocho horas, como ocurre en la versión Unix de MySQL.

  - Si una conexión se congela, no es posible eliminarla sin interrumpir a MySQL.

- `mysqladmin kill` no funciona con una conexión congelada.
- `mysqladmin shutdown` no funciona en tanto haya conexiones congeladas.

Se planea resolver este problema en el futuro.

- **`ALTER TABLE`**

  Mientras se está ejecutando una sentencia `ALTER TABLE`, la tabla está bloqueada frente al uso por parte de otros subprocesos. Esto tiene que ver con el hecho de que en Windows no se puede eliminar un fichero que está en uso por otro subproceso. En el futuro se podría encontrar alguna solución a este problema.

- **`DROP TABLE`**

  Realizar `DROP TABLE` sobre una tabla que está en uso por una tabla `MERGE` no funcionará en Windows porque el manejador `MERGE` oculta el mapeo de la tabla a la capa superior de MySQL. Debido a que Windows no permite eliminar archivos que se encuentran abiertos, primero deberán guardarse los cambios en todas las tablas `MERGE` (con `FLUSH TABLES`) o eliminar la tabla `MERGE` antes de borrar la tabla en cuestión.

- **`DATA DIRECTORY` e `INDEX DIRECTORY`**

  Las opciones de `CREATE TABLE DATA DIRECTORY` e `INDEX DIRECTORY` se ignoran en Windows, ya que Windows no soporta vínculos simbólicos. Estas opciones también se ignoran en otros sistemas operativos que no tengan una llamada `realpath()` funcional.

- **`DROP DATABASE`**

  No se puede eliminar una base de datos que está siendo utilizada por algún subproceso.

- **Finalizar MySQL desde el Administrador de Tareas**

  No es posible finalizar MySQL desde el Administrador de Tareas o con la utilidad shutdown en Windows 95. Se lo debe detener usando `mysqladmin shutdown`.

- **Nombres case-insensitive**

  Los nombres de ficheros no son case sensitive en Windows, por lo tanto tampoco lo son los nombres de bases de datos y tablas. La única restricción es que los nombres de bases de datos y tablas deben ser especificados empleando el mismo esquema de mayúsculas y minúsculas a lo largo de la misma sentencia. Consulte Sección 9.2.2, “Sensibilidad a mayúsculas y minúsculas de identificadores”.

- **El separador de rutas '`\`'**

  En Windows, los componentes de las rutas de directorios están separados por '`\`', el cual es también el carácter de escape en MySQL. Si se está utilizando `LOAD DATA INFILE` o `SELECT ... INTO OUTFILE`, deben usarse nombres de ficheros al estilo Unix, separados con caracteres '`/`':

```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

  Una alternativa es duplicar el carácter '`\`':

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- **Problemas con pipes.**

  Los pipes no funcionan confiablemente desde la línea de comandos de Windows. Si el pipe incluye el carácter `^Z` / `CHAR(24)`, Windows lo reconocerá como fin de fichero y terminará el programa.

  Esto es un problema particularmente cuando se intenta aplicar un fichero de registro (log) binario:

```
C:\> mysqlbinlog binary-log-name | mysql --user=root
```

  Si ocurre un problema al aplicar el fichero de registro y se sospecha que es causado por un carácter `^Z` / `CHAR(24)`, puede intentarse la siguiente solución:

```
C:\> mysqlbinlog binary-log-file --result-file=/tmp/bin.sql
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

  El último comando tambien puede usarse para leer confiablemente cualquier fichero SQL que pueda contener datos binarios.

- **Error `Access denied for user` (Acceso denegado a usuario)**

  Si se intenta ejecutar un programa cliente MySQL para conectarse a un servidor que funciona en el mismo ordenador, pero se obtiene el mensaje de error `Access denied for user 'algún-usuario'@'unknown' to database 'mysql'`, significa que MySQL no puede resolver apropiadamente el nombre del ordenador anfitrión (host).

  Para resolver esto, se debe crear un fichero llamado `\windows\hosts` conteniendo la siguiente información:

```
127.0.0.1       localhost
```

La siguiente es una lista de temas pendientes para aquellos que deseen colaborar en el perfeccionamiento de MySQL para Windows:

- Agregar macros para utilizar los métodos de incremento/decremento provistos por Windows, más rápidos y seguros para el trabajo con subprocesos.

2.4. Instalar MySQL en Linux

La manera recomendada de instalar MySQL en Linux es utilizando paquetes RPM. Los RPMs de MySQL están generados en SuSE Linux 7.3, pero deberían funcionar con cualquier versión de Linux que soporte `rpm` y el uso de `glibc`. Para obtener los paquetes RPM, consulte Sección 2.1.3, “Cómo obtener MySQL”.

MySQL AB proporciona RPMs específicos para algunas plataformas; la diferencia entre un RPM específico para una plataforma y uno genérico es que el primero es generado sobre la misma plataforma a donde está destinado, y emplea enlazado dinámico, en tanto que el RPM genérico está enlazado estáticamente con LinuxThreads.

Nota: las distribuciones RPM de MySQL a menudo estan proporcionadas por otros proveedores. Hay que tener en cuenta que pueden diferir, en características y prestaciones, de aquellas generadas por MySQL AB, y que las instrucciones de instalación en este manual no se les aplican necesariamente. Se deberían consultar las instrucciones del proveedor.

Si ocurren problemas con un fichero RPM (por ejemplo, si se recibe el error “`Sorry, the host 'xxxx' could not be looked up`”), consulte Sección 2.12.1.2, “Notas sobre la distribución binaria de Linux”.

En la mayoría de los casos, sólo será necesario instalar los paquetes `MySQL-server` y `MySQL-client` para conseguir una instalación de MySQL en funcionamiento. Los otros paquetes no se necesitan para una instalación estándar. Si se deseara ejecutar un servidor MySQL-Max, el cual posee capacidades adicionales, se debería instalar también el RPM `MySQL-Max`. No obstante, ello debería hacerse solamente *después* de instalar el RPM de `MySQL-server`. Consulte Sección 5.1.2, “El servidor extendido de MySQL `mysqld-max`”.

Si se obtiene un mensaje de error de dependencias cuando se intentan instalar los paquetes MySQL (por ejemplo, “`error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...`”), se deberá instalar también el paquete `MySQL-shared-compat`, el cual incluye las bibliotecas para compatibilidad hacia atrás (`libmysqlclient.so.12` para MySQL 4.0 y `libmysqlclient.so.10` para MySQL 3.23).

Muchas distribuciones Linux aún incluyen MySQL 3.23 y usualmente enlazan las aplicaciones dinámicamente para economizar espacio de disco. Si estas bibliotecas compartidas están en un paquete separado (por ejemplo, `MySQL-shared`), es suficiente con dejar ese paquete instalado y solamente actualizar el servidor MySQL y los paquetes cliente (los cuales están enlazados estáticamente y no dependen de bibliotecas compartidas). Para aquellas distribuciones que incluyen las bibliotecas compartidas en el mismo paquete que el servidor MySQL (por ejemplo, Red Hat Linux), se puede instalar el RPM `MySQL-shared` 3.23 o utilizar en su lugar el paquete `MySQL-shared-compat`.

Están disponibles los siguientes paquetes RPM:

- `MySQL-server-`*`VERSION`*`.i386.rpm`

  El servidor MySQL. Será necesario, a menos que solamente se desee conectar a un servidor MySQL ejecutado en otro ordenador. Nota: los ficheros RPM del servidor se denominaban `MySQL-`*`VERSION`*`.i386.rpm` antes de la versión 4.0.10. Es decir, no incluían `-server` en su nombre.

- `MySQL-Max-`*`VERSION`*`.i386.rpm`

  El servidor MySQL-Max. Este servidor tiene capacidades adicionales que no posee el provisto en el RPM `MySQL-server`. Igualmente, debe instalarse primero el RPM `MySQL-server`, porque el RPM `MySQL-Max` depende de él.

- `MySQL-client-`*`VERSION`*`.i386.rpm`

  Los programas cliente MySQL estándar. Es probable que siempre se instale este paquete.

- `MySQL-bench-`*`VERSION`*`.i386.rpm`

  Pruebas al programa y pruebas de rendimiento. Requieren Perl y el módulo `DBD::mysql`.

- `MySQL-devel-`*`VERSION`*`.i386.rpm`

  Las bibliotecas y ficheros de cabecera que se necesitan para compilar otros clientes MySQL, como los módulos Perl.

- `MySQL-shared-`*`VERSION`*`.i386.rpm`

  Este paquete contiene las bibliotecas compartidas (`libmysqlclient.so*`) que ciertos lenguajes y aplicaciones necesitan para enlazar dinámicamente y usar MySQL.

- `MySQL-shared-compat-`*`VERSION`*`.i386.rpm`

  Este paquete incluye las bibliotecas compartidas para MySQL 3.23 y MySQL 4.0. Debe instalarse en lugar de `MySQL-shared` si hay instaladas aplicaciones enlazadas dinámicamente con MySQL 3.23 y se

desea actualizar a MySQL 4.0 sin afectar las dependencias de bibliotecas. Este paquete se encuentra disponible desde MySQL 4.0.13.

- `MySQL-embedded-VERSION.i386.rpm`

  La biblioteca del servidor MySQL incrustado (desde MySQL 4.0)

- `MySQL-VERSION.src.rpm`

  Contiene el código fuente de todos los paquetes anteriores. Puede usarse para regenerar los RPMs bajo otras arquitecturas (por ejemplo, Alpha o SPARC).

Para ver todos los ficheros contenidos en un paquete RPM (por ejemplo, un RPM `MySQL-server`), se debe ejecutar:

```
shell> rpm -qpl MySQL-server-VERSION.i386.rpm
```

Para llevar a cabo una instalación estándar mínima, debe ejecutarse:

```
shell> rpm -i MySQL-server-VERSION.i386.rpm
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

Para instalar solamente el paquete cliente, debe ejecutarse:

```
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

RPM ofrece una característica para verificar la integridad y autenticidad de los paquetes antes de instalarlos. Para más información consulte Sección 2.1.4, “Comprobar la integridad de paquetes con sumas de verificación MD5 o GnuPG”.

El servidor RPM ubica los datos bajo el directorio `/var/lib/mysql`. También crea una cuenta de acceso para el usuario `mysql` (si no existe anteriormente) a fin de ejecutar el servidor MySQL, y crea las correspondientes entradas en `/etc/init.d/` para iniciar el servidor automáticamente al arrancar el sistema. (Esto significa que si se había realizado una instalación previa y se hicieron cambios al script de inicio, posiblemente se desee hacer una copia de ese script para no perder los cambios al instalar un nuevo RPM). Consulte Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente” para más información sobre como MySQL puede iniciarse automáticamente junto con el sistema.

Si se va a instalar el RPM MySQL en una distribución antigua de Linux la cual no soporta scripts de inicio en `/etc/init.d` (directamente o por medio de un symlink), deberá crearse un vínculo simbólico que apunte a la ubicación donde realmente está instalado el script de inicialización. Por ejemplo, si la ubicación es `/etc/rc.d/init.d`, se deberán ejecutar los siguientes comandos antes de instalar el RPM para crear `/etc/init.d` como un vínculo simbólico que apunte allí:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

Sin embargo, todas las principales distribuciones Linux de la actualidad soportan la nueva disposición de directorios que utiliza `/etc/init.d`, porque es un requisito para cumplir con el LSB (Linux Standard Base, Base Estándar para Linux).

Si entre los ficheros RPM instalados se encuentra `MySQL-server`, el servidor `mysqld` debería estar ejecutándose luego de la instalación, y se debería estar en condiciones de comenzar a utilizar MySQL.

Si algo no va bien, se puede hallar más información en la sección dedicada a la instalación binaria. Consulte Sección 2.7, “Instalación de MySQL en otros sistemas similares a Unix”.

Nota: Las cuentas que se hallan en las tablas de permisos de MySQL, en principio no están protegidas con contraseñas. Después de iniciar el servidor se deben establecer contraseñas para esas cuentas siguiendo las instrucciones en Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

2.5. Instalar MySQL en Mac OS X

Se puede instalar MySQL en Mac OS X 10.2.x (“Jaguar”) y posteriores utilizando un paquete binario de Mac OS X en formato PKG en lugar de la distribución binaria tarball. Debe tenerse en cuenta que las versiones anteriores de Mac OS X (por ejemplo, 10.1.x) **no** no están soportadas por este paquete.

El paquete se encuentra dentro de un fichero de imagen de disco (`.dmg`) que deberá montarse haciendo doble click sobre su ícono en Finder. Una vez montado debería verse su contenido en la pantalla.

Para obtener MySQL, consulte Sección 2.1.3, “Cómo obtener MySQL”.

Nota: Antes de proceder con la instalación, deben haberse finalizado todas las instancias del servidor MySQL en ejecución, ya sea usando la Aplicación MySQL Manager (en Mac OS X Server) o a través de `mysqladmin shutdown` en la línea de comandos.

Para instalar el fichero PKG de MySQL, debe hacerse doble click en el ícono del paquete. Esto iniciará el Instalador de Paquetes de Mac OS X, el cual guiará el resto de la instalación.

Debido a un error en el instalador de paquetes de Mac OS X, puede llegar a verse este error en el cuadro de diálogo de selección de disco destino:

```
You cannot install this software on this disk. (null)
```

Si ocurre este error, simplemente debe hacerse click en el botón `Go Back` una vez para volver a la pantalla anterior. Luego hacer click en `Continue` para avanzar nuevamente a la selección de disco destinto, y entonces debería poderse elegir sin problemas la unidad de instalación. MySQL AB ha informado de este error a Apple, quien se encuentra investigando el problema.

El PKG para Mac OS X de MySQL se instala en `/usr/local/mysql-`*`VERSION`* y también instala un vínculo simbólico, `/usr/local/mysql`, apuntando a la nueva ubicación. Si existe un directorio llamado `/usr/local/mysql`, será renombrado a `/usr/local/mysql.bak` primero. Adicionalmente, el instalador creará las tablas de permisos en la base de datos `mysql` a través de la ejecución de `mysql_install_db` después de la instalación.

La disposición de la instalación es similar a la de la distribución binaria en fichero `tar`, todos los ficheros binarios de MySQL están ubicados en el directorio `/usr/local/mysql/bin`. El fichero de socket MySQL se crea por defecto en `/tmp/mysql.sock`. Consulte Sección 2.1.5, “Conformación de la instalación”.

La instalación de MySQL requiere una cuenta de usuario Mac OS X llamada `mysql`. En Mac OS X 10.2 y posteriores, debería existir por defecto una cuenta con este nombre.

Si se está ejecutando Mac OS X Server, entonces se tiene una versión de MySQL instalada. Las versiones de MySQL que acompañan a cada versión de Mac OS X Server se muestran en la siguiente tabla:

Versión de Mac OS X Server	Versión de MySQL
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

Esta sección del manual abarca solamente la instalación del PKG oficial para Mac OS X de MySQL. Se debe leer la ayuda de Apple relativa a la instalación de MySQL: Ejecutando la aplicación “Help View”, seleccionando la ayuda de “Mac OS X Server”, haciendo una búsqueda por “MySQL”, y leyendo el tema titulado “Installing MySQL.”

En versiones de MySQL preinstaladas en Mac OS X Server, hay que tener en cuenta especialmente que se debería dar inicio a `mysqld` con el comando `safe_mysqld` en lugar de `mysqld_safe` si MySQL es anterior a la versión 4.0.

Si anteriormente se estuvieron utilizando los paquetes para Mac OS X de Marc Liyanage, descargados de http://www.entropy.ch, es suficiente con seguir las instrucciones para actualizar paquetes que usan la disposición de la instalación binaria, como se ha presentado en estas páginas.

Si se está actualizando hacia el PKG MySQL oficial desde alguna de las versiones 3.23.xx de Marc, o desde la versión de MySQL que acompaña al Mac OS X Server, se pueden convertir al formato actual las tablas de privilegios MySQL existentes, ya que se añadieron algunos nuevos privilegios de seguridad. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Si se desea iniciar automáticamente el servidor MySQL junto con el arranque del sistema, será necesario instalar también el Componente MySQL Startup (Inicio de MySQL). En el caso de MySQL 5.0, viene como un paquete separado dentro de las imágenes de disco de instalación. Siplemente hay que hacer doble click en el ícono MySQLStartupItem.pkg y seguir las instrucciones para instalarlo.

El Componente de Inicio de MySQL sólo necesita ser instalado una vez: no hay necesidad de instalarlo cada vez que se hace una actualización de MySQL

El Componente de Inicio de MySQL se instala en `/Library/StartupItems/MySQLCOM`. (Antes de MySQL 4.1.2, la ubicación era `/Library/StartupItems/MySQL`, pero entraba en conflicto con el Componente de Inicio de MySQL instalado por Mac OS X Server). La instalación del Componente de Inicio agrega una variable `MYSQLCOM=-YES-` al fichero de configuración del sistema `/etc/hostconfig`. Si se deseara deshabilitar el inicio automático de MySQL, simplemente hay que cambiar esta variable a `MYSQLCOM=-NO-`.

En Mac OS X Server, la instalación por defecto de MySQL utiliza la variable `MYSQL` en el fichero `/etc/hostconfig`. El instalador del Componente de Inicio de MySQL provisto por MySQL AB deshabilita esta variable estableciéndola en `MYSQL=-NO-`. Esto evita conflictos al momento del arranque del sistema con la variable `MYSQLCOM` utilizada por el Componente de Inicio de MySQL AB. Sin embargo, ello no finaliza un server MySQL en ejecución. Eso debería ser hecho expresamente por el usuario.

Luego de la instalación, se puede iniciar MySQL ejecutando los siguientes comandos en una ventana de terminal. Se deben tener privilegios de administrador para llevar a cabo esta tarea.

Si se ha instalado el Componente de Inicio:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

Si no se ha instalado el Componente de Inicio, debe ingresarse la siguiente secuencia de comandos:

```
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)
```

Se debería estar en condiciones de conectar con el servidor MySQL, por ejemplo, ejecutando `/usr/local/mysql/bin/mysql`.

Nota: Las cuentas que se hallan en las tablas de permisos de MySQL, en principio no están protegidas con contraseñas. Después de iniciar el servidor se deben establecer contraseñas para esas cuentas siguiendo las instrucciones en Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

Se podría desear agregar alias al fichero de recursos del shell para facilitar el acceso a los programas más utilizados, como `mysql` y `mysqladmin`, desde la línea de comandos. La sintaxis para `tcsh` es:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

Para `bash`, debe usarse:

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

Aún mejor, es agregar `/usr/local/mysql/bin` a la variable de entorno `PATH`. Por ejemplo, si se emplea el shell `tcsh`, agregando la siguiente línea al fichero `$HOME/.tcshrc`:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

Si en el directorio home no existe el fichero `.tcshrc`, se lo deberá crear con un editor de textos.

Si se está actualizando una instalación existente, hay que notar que instalar un nuevo PKG MySQL no borra el directorio de la instalación anterior. Desafortunadamente, el instalador de Mac OS X aún no ofrece la funcionalidad necesaria para actualizar apropiadamente los paquetes instalados con anterioridad.

Para utilizar en la nueva instalación las bases de datos existentes, habrá que copiar el contenido del directorio de datos antiguo dentro del nuevo. Hay que asegurarse que ni el antiguo servidor ni el nuevo estén en funcionamiento cuando se haga esto. Luego de que se hayan copiado las bases de datos desde la antigua instalación hacia la nueva, y se haya iniciado exitosamente el nuevo servidor, debe considerarse la eliminación de la instalación anterior a fin de recuperar espacio en disco. Quizá también se desee borrar versiones antiguas de los directorios Receipt localizados en `/Library/Receipts/mysql-VERSION.pkg`.

2.6. Instalar MySQL sobre NetWare

MySQL fue portado a NetWare a través de un esfuerzo encabezado por Novell. Los clientes de Novell se sentirán gratificados al advertir que NetWare 6.5 incluye la distribución binaria de MySQL, con una licencia comercial para todos los servidores que ejecuten esa versión de NetWare.

MySQL para NetWare está compilado utilizando una combinación de Metrowerks CodeWarrior para NetWare y versiones especiales de compilación cruzada de las GNU autotools.

La última distribución binaria para NetWare puede obtenerse en http://dev.mysql.com/downloads/. Consulte Sección 2.1.3, “Cómo obtener MySQL”.

A fin de hospedar a MySQL, el servidor NetWare debe cumplir estos requisitos:

- Debe ser NetWare 6.5 con Support Pack 2 instalado y actualizado con la última LibC, o NetWare 6.0 con Support Pack 4 instalado y actualizado con la última LibC. El Support Pack 2 de NetWare

6.5 y otras actualizaciones pueden descargarse de: http://support.novell.com/filefinder/18197/index.html. El Support Pack 4 de NetWare 6.0 y otras actualizaciones pueden descargarse de: http://support.novell.com/filefinder/13659/index.html. La última biblioteca LibC puede descargarse de: http://developer.novell.com/ndk/libc.htm. Las instrucciones para actualizar LibC se encuentran en: http://developer.novell.com/ndk/doc/libc/index.html?page=/ndk/doc/libc/libc_enu/data/ajjl0r0.html.

- Para poder ejecutar la respectiva versión de NetWare, el sistema debe cumplir con los requisitos mínimos de Novell.
- Tanto los datos como los ficheros binarios de MySQL deben instalarse en un volumen NSS; los volúmenes tradicionales no están soportados.

Debe emplearse el siguiente procedimiento para instalar MySQL para NetWare:

1. Si se está actualizando desde una versión anterior, debe detenerse el servidor MySQL. Esto se hace desde la consola del servidor, utilizando el siguiente comando:

   ```
   SERVER:  mysqladmin -u root shutdown
   ```

2. Debe iniciarse sesión en el servidor de destino desde un ordenador cliente que tenga acceso a la ubicación donde se instalará MySQL.

3. Extraer en el servidor el paquete binario contenido en el fichero Zip. Hay que cerciorarse de habilitar las rutas en el fichero Zip para que sea usado. Lo más seguro es simplemente extraerlo en `SYS:\`.

   Si se está actualizando desde una instalación anterior, puede ser necesario copiar el directorio de datos (por ejemplo, `SYS:MYSQL\DATA`), así como `my.cnf`, si se lo había modificado. Luego puede borrarse la antigua copia de MySQL.

4. Posiblemente se desee renombrar el directorio de instalación con una denominación más consistente y simple de usar. Se recomienda emplear `SYS:MYSQL`; los ejemplos en este manual utilizan ese nombre para referirse al directorio de instalación en general.

5. Desde la consola del servidor, debe agregarse una ruta de búsqueda para el directorio conteniendo los NLMs de MySQL. Por ejemplo:

   ```
   SERVER:  SEARCH ADD SYS:MYSQL\BIN
   ```

6. Inicializar el directorio de datos y las tablas de permisos, de ser necesario, a través de la ejecución de `mysql_install_db` desde la consola del servidor.

7. Iniciar el servidor MySQL con el comando `mysqld_safe` desde la consola del servidor.

8. Para finalizar la instalación, se deberían agregar los siguientes comandos al `autoexec.ncf`. Por ejemplo, si la instalación de MySQL se encuentra en `SYS:MYSQL` y se desea iniciar MySQL automáticamente, habría que agregar las siguientes líneas:

   ```
   #Inicia el servidor de bases de datos MySQL 5.0.x
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE
   ```

   Si se ejecutará MySQL en NetWare 6.0, es altamente recomendable utilizar la opción `--skip-external-locking` en la línea de comandos:

   ```
   #Inicia el servidor de bases de datos MySQL 5.0.x
   ```

```
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --skip-external-locking
```

En tal caso también será necesario utilizar `CHECK TABLE` y `REPAIR TABLE` en lugar de `myisamchk`, puesto que `myisamchk` emplea bloqueo externo. Se sabe que el bloqueo externo cauas problemas en NetWare 6.0; el problema fue eliminado en NetWare 6.5.

`mysqld_safe` para NetWare despliega una pantalla. Cuando se descarga (finaliza) el NLM `mysqld_safe`, la pantalla no desaparece por defecto. En lugar de eso, espera por una entrada del usuario:

```
*<NLM has terminated; Press any key to close the screen>*
```

Si se desea que NetWare cierre la pantalla automáticamente, debe agregarse la opción `--autoclose` a `mysqld_safe`. Por ejemplo:

```
#Inicia el servidor de bases de datos MySQL 5.0.x
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

9. Al instalar MySQL 5.0, ya sea por primera vez o como actualización de una versión anterior, se debe descargar e instalar el módulo Perl para MySQL 5.0 desde http://forge.novell.com/modules/xfcontent/downloads.php/perl/Modules/MySQL-5.0.3a-Beta-LIBC-Based/. Al instalar MySQL 5.0, ya sea por primera vez o como actualización de una versión previa a la 4.1, se debe descargar e instalar la Extensión de PHP5 para MySQL 4.1 desde http://forge.novell.com/modules/xfcontent/downloads.php/php/Modules/MySQL%204.1/. (Este módulo también debería funcionar con MySQL 5.0)

El funcionamiento de `mysqld_safe` en NetWare se describe más adelante en Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.

Si ya había una instalación de MySQL en el servidor, hay que cerciorarse de verificar el `autoexec.ncf` en busca de comandos de inicio de MySQL, y editarlos o borrarlos según sea necesario.

Nota: Las cuentas que se hallan en las tablas de permisos de MySQL, en principio no están protegidas con contraseñas. Después de iniciar el servidor se deben establecer contraseñas para esas cuentas siguiendo las instrucciones en Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

2.7. Instalación de MySQL en otros sistemas similares a Unix

Esta sección abarca la instalación de aquellas distribuciones binarias que se proveen para varias plataformas en formato de ficheros comprimidos `tar` (con extensión `.tar.gz`). Consulte Sección 2.1.2.5, “Binarios de MySQL compilados por MySQL AB” para ver una lista detallada.

Para obtener MySQL, consulte Sección 2.1.3, “Cómo obtener MySQL”.

Las distribuciones binarias en ficheros `tar` tienen nombres con la forma `mysql-`*`VERSION`*`-`*`OS`*`.tar.gz`, donde *`VERSION`* es un número (por ejemplo, `5.0.9`), y *`OS`* indica el tipo de sistema operativo al cual está dirigida la distribución. (Por ejemplo, `pc-linux-i686`).

Adicionalmente a estos paquetes genéricos, MySQL AB también ofrece, para plataformas seleccionadas, distribuciones binarias en paquetes con el formato específico de la plataforma. Consulte Sección 2.2, “Instalación MySQL estándar con una distribución binaria” para obtener información sobre cómo instalarlas.

Para instalar una distribución binaria de MySQL en fichero `tar` se requieren las siguientes herramientas:

- GNU `gunzip` para descomprimir la distribución.
- Un `tar` para expandir la distribución. GNU `tar` funciona correctamente. Algunos sistemas operativos vienen con una versión preinstalada de `tar` que tiene algunos problemas. Por ejemplo, el `tar` incluido con Mac OS X y el de Sun presentan problemas con nombres de fichero largos. En Mac OS X puede utilizarse el también preinstalado programa `gnutar`. En otros sistemas que tengan un `tar` deficiente, se debería instalar antes GNU `tar`.

Si ocurren problemas, *siempre debe emplearse* *`mysqlbug`* para enviar consultas a la lista de correo MySQL. Aún si no se trata de un error, `mysqlbug` recoge información del sistema que será de utilidad para quienes intenten resolver el problema. Al no usar `mysqlbug` se reduce la probabilidad de obtener una solución. `mysqlbug` se puede hallar en el directorio `bin` luego de expandir la distribución. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

Los comandos básicos a ejecutarse para instalar y usar una distribución binaria de MySQL son:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

Nota: Este procedimiento no establece ninguna contraseña para las cuentas MySQL. Después de completar el procedimiento debe continuarse con Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

Esta es una versión más detallada del procedimiento para instalar una distribución binaria:

1. Crear un usuario y un grupo para `mysqld` a fin de que pueda ejecutarse:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   Estos comandos agregan el grupo `mysql` y el usuario `mysql`. La sintaxis para `useradd` y `groupadd` puede variar ligeramente entre las distintas versiones de Unix. También pueden llamarse `adduser` y `addgroup`.

   Si se quisiera llamar al usuario y al grupo con otro nombre en lugar de `mysql`, habría que substituir por el nombre apropiado en los siguientes pasos.

2. Posicionarse en el directorio en el cual se desea expandir la distribución. En el siguiente ejemplo se expandirá bajo `/usr/local`. (Las instrucciones, sin embargo, asumen que se tiene permisos suficientes para crear ficheros y directorios en `/usr/local`. Si tal directorio se encuentra protegido, se deberá llevar a cabo la instalación como usuario `root`.)

   ```
   shell> cd /usr/local
   ```

3. Obtener un fichero de distribución desde uno de los sitios listados en Sección 2.1.3, “Cómo obtener MySQL”. Dado un release, las distribuciones de todas las plataformas son generadas a partir del mismo código fuente.

4. Expandir la distribución, lo cual creará el directorio de instalación. Luego crear un vínculo simbólico a ese directorio:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

   El comando `tar` crea un directorio denominado `mysql-VERSION-OS`. El comando `ln` crea un vínculo simbólico a ese directorio. Esto permite referirse a ese directorio de una forma más sencilla: `/usr/local/mysql`.

   Con GNU `tar` no se necesita invocar separadamente a `gunzip`. Se puede reemplazar la primera línea con el siguiente comando alternativo, para descomprimir y extraer la distribución:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

5. Cambiar la ubicación dentro del directorio de instalación:

```
shell> cd mysql
```

   Se pueden encontrar varios ficheros y subdirectorios en el directorio `mysql`. Los más importantes a efectos de la instalación son los subdirectorios `bin` y `scripts`.

   - `bin`

     Este directorio contiene los programas cliente y el servidor. Se debería agregar la ruta completa de este directorio a la variable de entorno `PATH`, para que el shell encuentre los programas de MySQL apropiadamente. Consulte Apéndice E, *Variables de entorno*.

   - `scripts`

     Este directorio contiene el script `mysql_install_db` utilizado para inicializar la base de datos `mysql`, que contiene las tablas que almacenan los permisos de acceso al servidor.

6. Si no se ha instalado antes MySQL, se deben crear las tablas de permisos:

```
shell> scripts/mysql_install_db --user=mysql
```

   Si se ejecuta el comando como usuario `root`, se debe emplear la opción `--user` tal como se muestra. El valor de la opción debe ser el nombre de la cuenta de usuario creada en el primer paso, para permitir que el servidor se ejecute. Si se ejecuta el comando habiendo iniciado sesión como este último usuario, se puede omitir la opción `--user`.

   Despues de crear o actualizar la tabla de permisos, habrá que reiniciar el servidor manualmente.

7. Se debe cambiar el propietario de los programas binarios a `root` y el propietario del directorio de datos al que se creó para ejecutar `mysqld`. Asumiendo que se está en el directorio de instalación (`/usr/local/mysql`), el comando sería similar a este:

```
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

   El primer comando cambia el atributo de propietario de los ficheros y les asigna el usuario `root`. El segundo cambia el atributo de propietario del directorio de datos y le asigna el usuario `mysql`. El tercero cambia el atributo de grupo, asignándolo al grupo `mysql`.

8. Si se desea que MySQL se inicie automáticamente durante el arranque del ordenador, debe copiarse el fichero `support-files/mysql.server` a la ubicación donde se encuentran los ficheros de inicio del sistema. Puede hallarse más información dentro del mismo script `support-files/mysql.server` y en Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente”.

9. Pueden establecerse nuevas cuentas empleando el script `bin/mysql_setpermission` si se instalan los módulos de Perl `DBI` y `DBD::mysql`. Para más instrucciones consulte Sección 2.13, “Notas sobre la instalación de Perl”.

10. Si se desea utilizar `mysqlaccess` y la distribución MySQL se ha instalado en una ubicación no estándar, deberá cambiarse el valor de `$MYSQL`, la cual es la variable que `mysqlaccess` utiliza para saber dónde se encuentra el cliente `mysql`. Debe editarse el script `bin/mysqlaccess` aproximadamente en la línea 18, que tiene este aspecto:

    ```
    $MYSQL     = '/usr/local/bin/mysql';    # ruta al ejecutable mysql
    ```

    Debe modificarse la ruta para reflejar la ubicación del sistema donde `mysql` se encuentra realmente. Si no se hace así, se obtendrá un error `Broken pipe` cuando se ejecute `mysqlaccess`.

Después de que todo ha sido expandido e instalado, se debería probar la distribución.

El siguiente comando inicia al servidor MySQL:

```
shell> bin/mysqld_safe --user=mysql &
```

Hay más información acerca de `mysqld_safe` en Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.

Nota: Las cuentas que se hallan en las tablas de permisos de MySQL, en principio no están protegidas con contraseñas. Después de iniciar el servidor se deben establecer contraseñas para esas cuentas siguiendo las instrucciones en Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

2.8. Instalación de MySQL usando una distribución de código fuente

Antes de proceder a una instalación de código fuente, se debería verificar si hay una distribución binaria disponible para la plataforma que se desea utilizar y si esta sirve adecuadamente al propósito del usuario. MySQL AB ha hecho grandes esfuerzos para asegurarse que las distribuciones binarias están realizadas con las mejores opciones posibles.

Para obtener una distribucion de código fuente de MySQL, Sección 2.1.3, “Cómo obtener MySQL”.

Las distribuciones de código fuente MySQL se proveen como ficheros `tar` comprimidos y tienen nombres con la forma `mysql-`*`VERSION`*`.tar.gz`, donde *`VERSION`* es un número del tipo `5.0.9-beta`.

Se requieren las siguientes herramientas para generar e instalar MySQL a partir del código fuente:

- GNU `gunzip` para descomprimir la distribución.
- Un `tar` para expandir la distribución. GNU `tar` funciona correctamente. Algunos sistemas operativos vienen con una versión preinstalada de `tar` que tiene algunos problemas. Por ejemplo, el `tar` incluido con Mac OS X y el de Sun presentan problemas con nombres de fichero largos. En Mac OS X puede utilizarse el también preinstalado programa `gnutar`. En otros sistemas que tengan un `tar` deficiente, se debería instalar antes GNU `tar`.
- Un compilador ANSI C++. `gcc` 2.95.2 o posterior, `egcs` 1.0.2 o posterior o `egcs 2.91.66`, SGI C++, y SunPro ++ son algunos de los compiladores que funcionan correctamente. No se necesitará `libg++` si se emplea `gcc`. `gcc` 2.7.x tiene un error que imposibilita compilar algunos ficheros C++ a pesar de

que son correctos, como `sql/sql_base.cc`. Si solamente se dispone de `gcc` 2.7.x, será necesario actualizarlo para poder compilar MySQL. También se sabe que `gcc` 2.8.1 tiene problemas en algunas plataformas, de modo que debería evitarse su uso si hay un compilador más actual para la plataforma.

Se recomienda `gcc` 2.95.2 para compilar MySQL 3.23.x.

- Un buen programa `make`. GNU `make` siempre se recomienda y algunas veces es requerido. Si ocurriesen problemas, se aconseja intentar con GNU `make` 3.75 o posterior.

Si se dispone de una versión de `gcc` lo suficientemente actualizada como para soportar la opción `-fno-exceptions`, es *muy importante* que se utilice. De lo contrario, podría obtenerse un binario que presente errores fatales aleatorios. También se recomienda emplear `-felide-constructors` y `-fno-rtti` junto con `-fno-exceptions`. En caso de duda, debe procederse así:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

En la mayoría de los sistemas, esto producirá un binario rápido y estable.

Si ocurren problemas, *siempre debe emplearse* *`mysqlbug`* para enviar consultas a la lista de correo MySQL. Aún si no se trata de un error, `mysqlbug` recoge información del sistema que será de utilidad para quienes intenten resolver el problema. Al no usar `mysqlbug` se reduce la probabilidad de obtener una solución. `mysqlbug` se puede hallar en el directorio `bin` luego de expandir la distribución. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

2.8.1. Panorámica de la instalación de código fuente

Los comandos básicos para instalar una distribución MySQL de código fuente son:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> bin/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql var
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

Si se comienza desde un RPM fuente, debe procederse así:

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

Esto genera un binario RPM instalable. En versiones antiguas de RPM, podría ser necesario reemplazar el comando `rpmbuild` con `rpm`.

Nota: Este procedimiento no establece contraseñas para las cuentas de MySQL. Despues de concluirlo, hay que dirigirse a Sección 2.9, “Puesta en marcha y comprobación después de la instalación”, para la configuración y prueba posteriores a la instalación.

Esta es una versión más detallada del procedimiento para instalar una distribución de código fuente:

1. Crear un usuario y un grupo para `mysqld` a fin de que pueda ejecutarse:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   Estos comandos agregan el grupo `mysql` y el usuario `mysql`. La sintaxis para `useradd` y `groupadd` puede variar ligeramente entre las distintas versiones de Unix. También pueden llamarse `adduser` y `addgroup`.

   Si se quisiera llamar al usuario y al grupo con otro nombre en lugar de `mysql`, habría que substituir por el nombre apropiado en los siguientes pasos.

2. Posicionarse en el directorio en el cual se desea expandir la distribución.

3. Obtener un fichero de distribución desde uno de los sitios listados en Sección 2.1.3, “Cómo obtener MySQL”.

4. Expandir la distribución dentro del directorio actual:

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

   Este comando creará un directorio llamado `mysql-VERSION`.

   Con GNU `tar` no se necesita invocar separadamente a `gunzip`. Se puede usar el siguiente comando alternativo, para descomprimir y extraer la distribución:

   ```
   shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
   ```

5. Posicionarse en el directorio de más alto nivel de la distribución expandida:

   ```
   shell> cd mysql-VERSION
   ```

   Actualmente debe configurarse y compilarse MySQL desde este directorio. No se puede compilar en un directorio diferente.

6. Configurar el release y compilar todo:

   ```
   shell> ./configure --prefix=/usr/local/mysql
   shell> make
   ```

   Al ejecutar `configure`, es posible que se deseen especificar algunas opciones. Ejecutar `./configure --help` para obtener la lista de las opciones disponibles. Sección 2.8.2, “Opciones típicas de `configure`” trata acerca de alguans de las opciones más útiles.

   Si `configure` fallara y se enviase un mail a la lista de correo de MySQL para solicitar asistencia, deben incluirse algunas líneas del fichero `config.log` que en apariencia sirvan para resolver el problema. También, incluir el último par de líneas emitidas por `configure` en la pantalla. Para enviar el informe de error debe utilizarse el script `mysqlbug`. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

   Si el compilador falla, consulte Sección 2.8.4, “Problemas en la compilación de MySQL”.

7. Instalar la distribución:

   ```
   shell> make install
   ```

Si se desea crear un fichero de opciones, debe utilizarse como plantilla uno de los presentes en el directorio `support-files`. Por ejemplo:

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

Podría ser necesario iniciar sesión como `root` para ejecutar estos comandos.

Si se configurará el soporte para tablas `InnoDB`, habrá que editar el fichero `/etc/my.cnf`, quitar el carácter `#` al comienzo de las líneas de opciones que comienzan con `innodb_...`, y modificar los valores según lo deseado. Consulte Sección 4.3.2, “Usar ficheros de opciones” y Sección 15.3, “Configuración de `InnoDB`”.

8. Posicionarse en el directorio de instalación:

```
shell> cd /usr/local/mysql
```

9. Si no se ha instalado antes MySQL, se deben crear las tablas de permisos:

```
shell> bin/mysql_install_db --user=mysql
```

Si se ejecuta el comando como usuario `root`, se debe emplear la opción `--user` tal como se muestra. El valor de la opción debe ser el nombre de la cuenta de usuario creada en el primer paso, para permitir que el servidor se ejecute. Si se ejecuta el comando habiendo iniciado sesión como este último usuario, se puede omitir la opción `--user`.

Despues de crear o actualizar la tabla de permisos mediante `mysql_install_db`, habrá que reiniciar el servidor manualmente.

10. Se debe cambiar el propietario de los programas binarios a `root` y el propietario del directorio de datos al que se creó para ejecutar `mysqld`. Asumiendo que se está en el directorio de instalación (`/usr/local/mysql`), el comando sería similar a este:

```
shell> chown -R root  .
shell> chown -R mysql var
shell> chgrp -R mysql .
```

El primer comando cambia el atributo de propietario de los ficheros y les asigna el usuario `root`. El segundo cambia el atributo de propietario del directorio de datos y le asigna el usuario `mysql`. El tercero cambia el atributo de grupo, asignándolo al grupo `mysql`.

11. Si se desea que MySQL se inicie automáticamente durante el arranque del ordenador, debe copiarse el fichero `support-files/mysql.server` a la ubicación donde se encuentran los ficheros de inicio del sistema. Puede hallarse más información dentro del mismo script `support-files/mysql.server` y en Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente”.

12. Pueden establecerse nuevas cuentas empleando el script `bin/mysql_setpermission` si se instalan los módulos de Perl `DBI` y `DBD::mysql`. Para más instrucciones consulte Sección 2.13, “Notas sobre la instalación de Perl”.

Luego de que todo se ha instalado, se debería inicializar y probar la distribución por medio del siguiente comando:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

Si este comando fallara inmediatamente y emitiera el mensaje `mysqld ended`, se podrá encontrar más información en el fichero *`host_name`*`.err`, localizado en el directorio de datos.

Hay más información acerca de `mysqld_safe` en Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.

Nota: Las cuentas que se hallan en las tablas de permisos de MySQL, en principio no están protegidas con contraseñas. Después de iniciar el servidor se deben establecer contraseñas para esas cuentas siguiendo las instrucciones en Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

2.8.2. Opciones típicas de `configure`

El script `configure` brinda un gran control sobre la configuración de una distribución de código fuente. Generalmente esto se hace por medio de las opciones que siguen a `configure` en la línea de comandos, aunque `configure` también se ve afectado por ciertas variables de entorno. Consulte Apéndice E, *Variables de entorno*. Para obtener una lista de las opciones aceptadas por `configure`, debe ejecutarse este comando:

```
shell> ./configure --help
```

A continuación se describen algunas de las opciones de `configure` más comunes:

- Para compilar solamente las bibliotecas cliente y los programas cliente de MySQL, sin el servidor, debe emplearse la opción `--without-server`.

  ```
  shell> ./configure --without-server
  ```

  Si no se dispone de un compilador de C++, no se podrá compilar `mysql` (es el único programa cliente que necesita de C++). En este caso, se puede quitar de `configure` el código que comprueba la existencia del compilador C++ y luego ejecutar `./configure` con la opción `--without-server`. El proceso de compilación igualmente intentará generar `mysql`, pero se puede ignorar cualquier advertencia acerca de `mysql.cc`. (Si `make` se detiene, debe intentarse con `make -k`, que continúa con el resto de la generación aún si ocurren errores).

- Si se deseara generar la biblioteca MySQL incrustada (`libmysqld.a`), debe utilizarse la opción `--with-embedded-server`.

- Si no se desea que los archivos de registro (logs) y los directorios de bases de datos se ubiquen dentro de `/usr/local/var`, se debe emplear un comando de `configure` similar a este:

  ```
  shell> ./configure --prefix=/usr/local/mysql
  shell> ./configure --prefix=/usr/local \
             --localstatedir=/usr/local/mysql/data
  ```

  El primer comando cambia el prefijo de instalación de modo que todo sea instalado en `/usr/local/mysql` en lugar de `/usr/local`. Els egundo comando conserva el prefijo de instalación por defecto, pero reemplaza la ubicación predeterminada de los directorios de datos (normalmente, `/usr/local/var`) y lo establece en `/usr/local/mysql/data`. Después de haber compilado MySQL, se pueden cambiar estas opciones con ficheros de opciones. consulte Sección 4.3.2, “Usar ficheros de opciones”.

- Si se está utilizando Unix y se desea que el socket MySQL se ubique en otro directorio que el preeterminado (normalmente en `/tmp` o `/var/run`), debe emplearse un comando `configure` similar a este:

  ```
  shell> ./configure \
  ```

```
          --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

El nombre de fichero para el socket debe ser una ruta absoluta. También se puede modificar la ubicación de `mysql.sock` posteriormente, con un fichero de opciones MySQL. Consulte Sección A.4.5, “Cómo proteger o cambiar el fichero socket de MySQL `/tmp/mysql.sock`”.

- Se se desea compilar programas con enlazado estático (por ejemplo, para hacer una distribución binaria, incrementar la velocidad, o solucionar problemas que ocurren con algunas distribuciones de Red Hat Linux) debe ejecutarse `configure` de esta manera:

```
shell> ./configure --with-client-ldflags=-all-static \
          --with-mysqld-ldflags=-all-static
```

- Si se está usando `gcc` y no se han instalado `libg++` o `libstdc++`, se le puede indicar a `configure` que utilice `gcc` como compilador de C++:

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

Cuando se emplea `gcc` como compilador de C++, este no intenta enlazar con `libg++` o `libstdc++`. Esta puede ser buena idea incluso si se dispone de estas bibliotecas, ya que algunas versiones de ellas causaron, en el pasado, problemas extraños a usuarios de MySQL.

La siguiente lista presenta algunos compiladores y la configuración de variables de entorno comúnmente utilizada con cada uno.

- `gcc` 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

- `egcs` 1.0.3a:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors \
-fno-exceptions -fno-rtti"
```

- `gcc` 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

- `pgcc` 2.90.29 o posterior:

```
CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \
CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \
-felide-constructors -fno-exceptions -fno-rtti"
```

En la mayoría de los casos, podrá obtenerse un binario MySQL razonablemente optimizado usando las opciones de la lista anterior, con el añadido de las siguientes opciones en la línea de `configure`:

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

En otras palabras, la línea completa de `configure` tendría el siguiente aspecto para todas las versiones recientes de `gcc`:

```
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
```

Los binarios provistos en el sitio Web de MySQL en http://www.mysql.com/ están, en su totalidad, compilados con la máxima optimización y deberían ajustarse perfectamente a la mayoría de los usuarios. Consulte Sección 2.1.2.5, “Binarios de MySQL compilados por MySQL AB”. Existen algunas configuraciones que se pueden manipular para hacer un binario más rápido aún, pero se reservan para los usuarios avanzados. Consulte Sección 7.5.4, “Efectos de la compilación y del enlace en la velocidad de MySQL”.

Si la generación falla y emite mensajes de error relativos a que el compilador o el enlazador no son capaces de crear la biblioteca compartida `libmysqlclient.so.#` (donde '#' es un número de versión), se puede resolver el problema pasándole la opción `--disable-shared` a `configure`. En este caso, `configure` no generará una biblioteca compartida `libmysqlclient.so.#`.

- Por defecto, MySQL emplea el conjunto de caracteres `latin1` (ISO-8859-1). Para cambiar el conjunto de caracteres por defecto, se utiliza la opción `--with-charset`:

```
shell> ./configure --with-charset=CHARSET
```

*`CHARSET`*puede ser cualquiera entre los siguientes: `big5`, `cp1251`, `cp1257`, `czech`, `danish`, `dec8`, `dos`, `euc_kr`, `gb2312`, `gbk`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, or `win1251ukr`. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

En MySQL 5.0 también puede especificarse el tipo de comparación (collation) estándar. MySQL utiliza la comparación `latin1_swedish_ci` por defecto. Para cambiar esto se emplea la opción `--with-collation`:

```
shell> ./configure --with-collation=COLLATION
```

Para cambiar tanto el conjunto de caracteres como la forma de comparación, se usan ambas opciones `--with-charset` y `--with-collation`. La forma de comparación debe ser apropiada para el conjunto de caracteres. (Para determinar qué tipos están disponibles para cada conjunto de carcateres se usa la sentencia `SHOW COLLATION`).

Si se desean convertir caracteres entre el servidor y el cliente, se debería revisar la sentencia `SET CHARACTER SET`. Consulte Sección 13.5.3, “Sintaxis de `SET`”.

Advertencia: si se cambia el conjunto de caracteres luego de haber creado tablas, se debe ejecutar `myisamchk -r -q --set-character-set=`*`charset`* *en cada tabla*. De otro modo, los índices podrían quedar incorrectamente ordenados. (Esto puede ocurrir si se instala MySQL, se crean algunas tablas, se reconfigura MySQL para usar un conjunto de caracteres diferente, y se lo reinstala).

Con la opción de `configure --with-extra-charsets=`*`LIST`*, pueden definirse conjuntos de caracteres adicionales para ser compilados dentro del servidor. *`LIST`* debe ser uno de los siguientes:

- una lista de nombres de conjuntos de caracteres separados por espacios
- `complex` - para incluir todos los conjuntos de caracteres que no pueden ser cargados dinámicamente
- `all` - para incluir en los binarios todos los conjuntos de caracteres.

- Para configurar MySQL con código de depuración, debe usarse la opción `--with-debug`:

```
shell> ./configure --with-debug
```

  Esto causa la inclusión de un asignador de memoria seguro que puede detectar algunos errores y brinda información de lo que está ocurriendo. Consulte Sección D.1, “Depurar un servidor MySQL”.

- Si los programas cliente utilizan subprocesos (threads), también se deberá compilar una versión a prueba de subprocesos (thread-safe) de la biblioteca cliente de MySQL, con la opción de configure `--enable-thread-safe-client`. Esto crea una biblioteca `libmysqlclient_r` con la cual se podrán enlazar las aplicaciones que emplean subprocesos. Consulte Sección 24.2.15, “Cómo hacer un cliente multihilo”.

- Ahora, a partir de MySQL 5.0.4, es posible generar a MySQL 5.0 con soporte para tablas grandes, utilizando la opción `--with-big-tables`.

  Esta opción tiene por efecto que las variables utilizadas para llevar la cuenta de las filas de tablas se almacenen empleando un tipo `unsigned long long` en lugar de `unsigned long`. Esto permite mantener tablas con hasta aproximadamente 1.844E+19 ($(2^{32})^2$) registros en vez de 2^{32} (~4.295E+09). Antes se hacía necesario pasar al compilador el argumento `-DBIG_TABLES` en forma manual con el fin de habilitar la misma característica.

- Las opciones que pertenezcan a un sistema en particular se hallan en la sección de este manual dedicada específicamente a ese sistema. Consulte Sección 2.12, “Notas específicas sobre sistemas operativos”.

2.8.3. Instalar desde el árbol de código fuente de desarrollo

Atención: Se debe leer esta sección únicamente si se está interesado en colaborar con MySQL AB en la prueba de código nuevo. Si solamente se desea obtener MySQL para ejecutarlo en un sistema, se debe emplear una distribución estándar (ya sea binaria o de código fuente).

Para obtener el directorio de desarrollo más reciente, se deben seguir estas instrucciones:

1. Descargar el cliente gratuito BitKeeper desde http://www.bitmover.com/bk-client.shar.

2. En Unix, instalar el cliente gratuito de esta manera:

```
shell> sh bk-client.shar
shell> cd bk_client-1.1
shell> make all
shell> PATH=$PWD:$PATH
```

   En Windows, instalarlo de esta manera:

   - Descargar e instalar Cygwin desde http://cygwin.com.

   - Asegurarse de que `gcc` fue instalado bajo Cygwin. Esto se comprueba emitiendo `which gcc`. Si no está instalado, ejecutar el gestor de paquetes (package manager) de Cygwin, seleccionar `gcc`, e instalarlo.

   - Bajo Cygwin, llevar a cabo estos pasos:

```
shell> sh bk-client.shar
shell> cd bk_client-1.1
```

Luego, editar el fichero `Makefile` y modificar la línea que lee `$(CC) $(CFLAGS) -o sfio -lz sfio.c` para que quede así:

```
$(CC) $(CFLAGS) -o sfio sfio.c -lz
```

Ejecutar el comando `make` y establecer la ruta:

```
shell> make all
shell> PATH=$PWD:$PATH
```

3. Después que el cliente gratuito BitKeeper se haya instalado, dirigirse en primer lugar al directorio desde donde se desea trabajar y utilizar el siguiente comando para hacer una copia local de la rama de MySQL 5.0:

```
shell> sfioball -r+ bk://mysql.bkbits.net/mysql-5.0 mysql-5.0
```

Normalmente, no es necesario generar la documentación, dado que ya es suministrada en una variedad de formatos en http://dev.mysql.com/doc/. Los formatos que pueden descargarse allí (HTML, PDF, etc.) están generados diariamente, de modo que no se gana gran cosa generándola en base a los documentos en formato XML DocBook que hay en el directorio `mysqldoc`. Si de todos modos se deseara copiar el repositorio de documentación, debe emplearse el siguiente comando:

```
shell> sfioball -r+ bk://mysql.bkbits.net/mysqldoc mysqldoc
```

En el ejemplo anterior, el árbol de código fuente se establece dentro del subdirectorio `mysql-5.0/` dentro del

Si se está detrás de un firewall y solamente pueden iniciarse conexiones HTTP, BitKeeper también puede usarse sobre HTTP.

Si se necesita usar un servidor proxy, establecer la variable de entorno `http_proxy` para apuntar al mismo:

```
shell> export http_proxy="http://your.proxy.server:8080/"
```

Reemplazar `bk://` con `http://` cuando se copie un repositorio. Ejemplo:

```
shell> sfioball -r+ http://mysql.bkbits.net/mysql-5.0 mysql-5.0
```

La descarga inicial del árbol de código fuente puede llevar un tiempo, dependiendo de la velocidad de la conexión. Si es así, habrá que tener paciencia hasta completar la descarga.

4. Para actualizar la copia local del repositorio MySQL 5.0, debe emplearse este comando:

```
shell> update bk://mysql.bkbits.net/mysql-5.0
```

5. Son necesarios: GNU `make`, `autoconf` 2.58 (o posterior), `automake` 1.8, `libtool` 1.5, y `m4` para ejecutar la siguiente serie de comandos. Aún cuando muchos sistemas operativos vienen con su propia implementación de `make`, hay muchas posibilidades de que la compilación falle con extraños mensajes de error. Por consiguiente, es muy recomendable utilizar GNU `make` (a veces llamado `gmake`) en lugar de otros.

Afortunadamente, un gran número de sistemas operativos se distribuyen con las herramientas GNU preinstaladas o proveen paquetes instalables de las mismas. En cualquier caso, pueden descargarse de las siguientes direcciones:

- http://www.gnu.org/software/autoconf/
- http://www.gnu.org/software/automake/
- http://www.gnu.org/software/libtool/
- http://www.gnu.org/software/m4/
- http://www.gnu.org/software/make/

Para configurar MySQL 5.0, también se necesitará GNU `bison` 1.75 o posterior. Las versiones antiguas de `bison` pueden producir este error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

Nota: El mensaje es: "Error fatal: se ha excedido el tamaño máximo para la tabla (32767)". El tamaño máximo de la tabla no está realmente excedido, el mensaje es consecuencia de errores en versiones antiguas de `bison`.

El siguiente ejemplo muestra los comandos típicamente requeridos para configurar un árbol de código fuente. El primer comando `cd` es para posicionarse en el directorio de más alto nivel en el árbol, debe reemplazarse `mysql-5.0` con el nombre de directorio apropiado.

```
shell> cd mysql-5.0
shell> bk -r edit
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd innobase; aclocal; autoheader; autoconf; automake)
shell> (cd bdb/dist; sh s_all)
shell> ./configure  # Add your favorite options here
shell> make
```

O bien puede usarse `BUILD/autorun.sh` como un atajo para la siguiente secuencia de comandos:

```
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd innobase; aclocal; autoheader; autoconf; automake)
shell> (cd bdb/dist; sh s_all)
```

La línea de comandos que cambia el directorio dentro de `innobase` y `bdb/dist` se usa para configurar los motores de almacenamiento `InnoDB` y Berkeley DB (`BDB`). Pueden omitirse si no se necesita soporte para `InnoDB` o `BDB`.

Si se obtienen algunos mensajes de error extraños durante este paso, debe verificarse si verdaderamente se encuentra instalado `libtool`.

En el subdirectorio `BUILD/` se encuentra una colección de scripts de configuración estándar utilizados por MySQL AB. Es posible que resulte más conveniente utilizar el script `BUILD/compile-pentium-debug` que el conjunto de comandos antes mencionado. Para compilar en una arquitectura diferente, debe modificarse el script eliminando los flags que son específicos de Pentium.

6. Cuando la generación esté concluida, debe ejcutarse `make install`. Debe tenerse cuidado con esto al aplicarlo en un ordenador en producción; los comandos pueden sobreeescribir la instalación en uso. Si hay otra instalación de MySQL, se recomienda ejecutar `./configure` con valores diferentes para las opciones `--prefix`, `--with-tcp-port`, y `--unix-socket-path` que aquellos empleados en el servidor en producción.

7. Ahora se debe "jugar" con la nueva instalación e intentar que las nuevas características colapsen con algún error. Debe comenzarse por ejecutar `make test`. Consulte Sección 27.1.2, “El paquete de pruebas MySQL Test”.

8. Si se ha llegado a la etapa de `make` y la distribución no se compila, debe informarse en la base de datos de errores en http://bugs.mysql.com/. Si se han instalado las últimas versiones de las herramientas GNU necesarias, y estas caen al intentar procesar los ficheros de configuración provistos, también debe informarse. Sin embargo, si se ejecuta `aclocal` y se obtiene un error `command not found` o similar, no debe informarse. En su lugar, hay que asegurarse de que todas las herramientas necesarias estén instaladas y que la variable `PATH` está correctamente establecida para que el shell las pueda localizar.

9. Luego de la copia inicial del repositorio (`sfioball`) para obtener el árbol de código fuente, se debe actualizar el repositorio (`update`) periódicamente para obtener novedades.

10. Para examinar la historia de cambios del árbol, con todas las diferencias, puede verse el fichero `BK/ChangeLog` localizado en el árbol de código fuente y observar las descripciones `ChangeSet` listadas. Para examinar una descripción changeset en particular, se utiliza el comando `sfioball` para extraer dos revisiones del árbol de código fuente, y luego se utiliza un comando externo `diff` para compararlas. En caso de hallar diferencias poco claras o tener preguntas acerca del código, no debe dudarse en enviar un correo electrónico a la lista de correo `internals` de MySQL. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”. Asimismo, si se considera que se tiene una mejor idea sobre cómo realizar algo, debe enviarse un mensaje de correo a la misma lista con la corrección.

11. El cliente gratuito BitKeeper es ditribuido con su código fuente. La única documentación disponible para el cliente gratuito es el propio código fuente.

También pueden verse cambios, comentarios y código fuente a través de la Web. Para ver esta información en el caso de MySQL 5.0, dirigirse a http://mysql.bkbits.net:8080/mysql-5.0.

2.8.4. Problemas en la compilación de MySQL

Todos los programas de MySQL se compilan en Solaris o Linux limpiamente, sin advertencias (warnings), utilizando `gcc`. En otros sistemas podrían emitirse advertencias debido a diferencias en los ficheros de cabecera del sistema. Consulte Sección 2.8.5, “Notas sobre MIT-pthreads” para advertencias que pueden emitirse al usar MIT-pthreads. Para otros problemas, debe verificarse la siguiente lista.

La solución a muchos problemas incluye la reconfiguración. Si se necesita reconfigurar, hay que tomar nota de lo siguiente:

- Si se ejecuta `configure` después de habérselo ejecutado anteriormente, éste puede emplear información recogida durante la primera vez. Esta información se almacena en `config.cache`. Cuando se inicia `configure`, busca aquel fichero y lee su contenido si existe, suponiendo que la información continúa siendo válida. Esa suposición es incorrecta cuando se ha reconfigurado.

- Cada vez que se ejecuta `configure`, se debe ejecutar `make` nuevamente para recompilar. Sin embargo, primero se podría desear eliminar ficheros objeto antiguos, pertenecientes a procesos de generación anteriores, ya que fueron compilados utilizando diferentes opciones de configuración.

Para evitar que se utilicen información de configuración o ficheros objeto antiguos, deben ejecutarse estos comandos antes de volver a ejecutar `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternativamente, puede utilizarse `make distclean`.

La siguiente lista enumera los problemas más comunes al compilar MySQL:

- Si se obtienen errores como los que se muestran cuando se compila `sql_yacc.cc`, probablemente se trate de memoria insuficiente o espacio de intercambio insuficiente:

  ```
  Internal compiler error: program cc1plus got fatal signal 11
  Out of virtual memory
  Virtual memory exhausted
  ```

  El problema es que `gcc` necesita enormes cantidades de memoria para compilar `sql_yacc.cc` con funciones "inline". Inténtese ejecutar `configure` con la opción `--with-low-memory`:

  ```
  shell> ./configure --with-low-memory
  ```

  Esta opción causa que `-fno-inline` se agregue a los argumentos del compilador si se está utilizando `gcc`, y `-O0` si se está utilizando cualquier otro. Se debería probar la opción `--with-low-memory` aún si se tiene tanta cantidad de memoria de espacio de intercambio que no parece posible que sean insuficientes. Este problema se ha observado inclusive en sistemas con configuraciones de hardware generosas. y la opción `--with-low-memory` usualmente lo repara.

- Por defecto, `configure` asume `c++` como nombre del compilador y GNU `c++` enlaza con `-lg++`. Si se está empleando `gcc`, tal comportamiento puede causar problemas como este durante la configuración:

  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  También se podrían observar problemas durante la compilacion relacionados con `g++`, `libg++`, o `libstdc++`.

  Una causa para estos problemas es que no se tenga `g++`, o quizá se tenga `g++` pero no `libg++`, o `libstdc++`. Se debe inspeccionar el fichero `config.log`. Este debería contener el motivo exacto por el cual el compilador de C++ no está funcionando. Para eludir estos problemas, se puede utilizar `gcc` como compilador de C++. Debe ponerse en la variable de entorno `CXX` el valor `"gcc -O3"`. Por ejemplo:

  ```
  shell> CXX="gcc -O3" ./configure
  ```

  Esto funciona porque `gcc` compila código C++ tan bien como `g++`, pero no enlaza por defecto con las bibliotecas `libg++` o `libstdc++`.

  Otra forma de solucionar el problema es instalar `g++`, `libg++`, y `libstdc++`. Sin embargo, se recomiendo no emplear `libg++` o `libstdc++` con MySQL porque sólo incrementará el tamaño del ejecutable sin obtener ningún beneficio. Algunas versiones de estas bibliotecas también han causado en el pasado problemas extraños a los usuarios de MySQL.

- Si la compilación fallara por errores como cualquiera de los siguientes, se debe actualizar la versión de `make` a GNU `make`:

  ```
  ```

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

O bien:

```
make: file `Makefile' line 18: Must be a separator (:
```

O bien:

```
pthread.h: No such file or directory
```

Se sabe que los programas `make` de Solaris y FreeBSD son problemáticos.

La versión 3.75 de GNU `make` funciona correctamente.

- Si se desean definir flags para ser usados por los compiladores de C o C++, se debe hacer agregando los flags a las variables de entorno `CFLAGS` y `CXXFLAGS`. De este mismo modo pueden especificarse los nombres del compilador utilizando `CC` y `CXX`. Por ejemplo:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

Consulte Sección 2.1.2.5, “Binarios de MySQL compilados por MySQL AB” para una lista de definiciones de flags que han resultado útiles en varios sistemas.

- Si se obtiene un mensaje de error como este, se necesitará actualizar el compilador `gcc`:

```
client/libmysql.c:273: parse error before `__attribute__'
```

`gcc` 2.8.1 funciona correctamente, pero se recomienda utilizar `gcc` 2.95.2 o `egcs` 1.0.3a en su lugar.

- Si se obtienen errores como los siguientes al compilar `mysqld`, `configure` no está detectando correctamente el tipo del último argumento pasado a `accept()`, `getsockname()`, o `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);
```

Para corregir esto, debe modificarse el fichero `config.h` (el cual es generado por `configure`). Deben buscarse estas líneas:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Hay que cambiar `XXX` por `size_t` o `int`, dependiendo del sistema operativo. (Tener en cuenta que esto debe hacerse cada vez que se ejecuta `configure`, porque este regenera el fichero `config.h`.)

- El fichero `sql_yacc.cc` se genera a partir de `sql_yacc.yy`. Normalmente el proceso de compilación no necesita crear `sql_yacc.cc`, porque MySQL viene con una copia ya generada. Sin embargo, si se necesita crearlo de nuevo, se podría hallar este error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

Este es un síntoma de que se posee una versión deficiente de `yacc`. Posiblemente se necesite instalar en su lugar `bison` (la versión GNU de `yacc`).

- En Linux Debian 3.0, se necesita instalar `gawk` en lugar del predeterminado `mawk` si se desea compilar MySQL 5.0 con soporte para Bases de Datos Berkeley.
- Si se necesita depurar `mysqld` o un cliente MySQL, ejecutar `configure` con la opción `--with-debug`, luego recompilar y enlazar los clientes con la nueva biblioteca. Consulte Sección D.2, “Depuración de un cliente MySQL”.
- Si se obtiene un error de compilación en Linux (por ejemplo, en Linux SuSE 8.1 o Linux RedHat 7.2) similar a los siguientes:

```
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
```

Por defecto, el script `configure` intenta determinar el número correcto de argumentos empleando `g++`, el compilador C++ GNU. Este proceso arroja resultados incorrectos si no está instalado `g++`. Hay dos formas de solucionar esto:

- Asegurarse de que GNU C++ `g++` está instalado. En algunas distribuciones de Linux, el paquete que se necesita se llama `gpp`; en otras, se llama `gcc-c++`.
- Para utilizar `gcc` como compilador de C++ se debe poner en la variable de entorno `CXX` el valor `gcc`:

```
export CXX="gcc"
```

Después debe volver a ejecutarse `configure`.

2.8.5. Notas sobre MIT-pthreads

Esta sección describe alguno de los probleas que pueden ocurrir al utilizar MIT-pthreads.

En Linux, *no se deberían* utilizar MIT-pthreads. En su lugar, utilizar la implementación de LinuxThreads instalada. Consulte Sección 2.12.1, “Notas sobre Linux”.

Si el sistema no provee soporte nativo para subprocesos, se necesita compilar MySQL utilizando el paquete MIT-pthreads. Esto incluye a antiguos sistemas FreeBSD, SunOS 4.x, Solaris 2.4 y anteriores, y algunos otros. Consulte Sección 2.1.1, “Sistemas operativos que MySQL soporta”.

El paquete MIT-pthreads no es parte de la distribución de código fuente de MySQL 5.0. Si se lo necesita, habrá que descargarlo desde http://www.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz.

Luego de la descarga, debe expandirse dentro del más alto nivel del directorio de código fuente de MySQL. Creará un nuevo subdirectorio llamado `mit-pthreads`.

- En la mayoría de los sistemas, para forzar el uso de MIT-pthreads se debe ejecutar `configure` con la opción `--with-mit-threads`.

```
shell> ./configure --with-mit-threads
```

La generación (building) en un directorio que no sea de código fuente no está soportada cuando se usan MIT-pthreads porque se desea minimizar los cambios al código.

- La comprobación que determina si se utilizará MIT-pthreads sucede únicamente durante la parte del proceso de configuración que tiene que ver con el código del servidor. Si se ha configurado la distribución empleando `--without-server` para generar solamente el código del cliente, los clientes no saben si se está utilizando MIT-pthreads y emplean conexiones socket Unix por defecto. Dado que los ficheros socket de Unix no funcionan bajo MIT-pthreads en algunas plataformas, esto significa que se deberá utilizar `-h` o `--host` cuando se ejecuten programas cliente.

- Cuando se compila MySQL empleado MIT-pthreads, el bloqueo de sistema se deshabilita por defecto por razones de rendimiento. Se le puede indicar al servidor que emplee bloqueo de sistema con la opción `--external-locking`. Esto es necesario solamente si se desea poder ejecutar dos servidores MySQL sobre los mismos ficheros de datos, lo cual no es recomendable.

- Algunas veces, el comando `bind()` de pthread falla al adosarse a un socket sin emitir mensajes de error (al menos en Solaris). El resultado es que todas las conexiones al servidor fallan. Por ejemplo:

```
shell> mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

La solución a esto es finalizar el servidor `mysqld` y reiniciarlo. Esto ha ocurrido solamente cuando se ha detenido el servidor por la fuerza y se lo ha reiniciado inmediatamente.

- Con MIT-pthreads, la llamada de sistema `sleep()` no es interrumpida con `SIGINT` (interrupción). Esto solamente es perceptible cuando se ejecuta `mysqladmin --sleep`. Se debe esperar a que la llamada `sleep()` termine antes de que el pedido de interrupción sea atendido y el proceso se detenga.

- Durante el enlazado, se pueden recibir mensajes de advertencia como los siguientes (al menos en Solaris); deben ser ignorados.

```
ld: warning: symbol `_iob' has differing sizes:
    (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
ld: warning: symbol `__iob' has differing sizes:
    (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- Algunas otras advertencias también deben ignorarse:

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- No se ha podido hacer funcionar `readline` con MIT-pthreads. (Esto no es necesario, pero podría ser de interés para alguien).

2.8.6. Instalar MySQL desde el código fuente en Windows

Estas instrucciones describen como generar los binarios de MySQL a partir del código fuente de la versión 5.0 en Windows. Las instrucciones que se proveen sirven para generar los binarios a partir de

una distribución estándar de código fuente o a partir del árbol de BitKeeper que contiene el último código fuente en desarrollo.

Nota: Las instrucciones en este documento son estrictamente para usuarios que deseen probar MySQL en Windows a partir de la última distribución en código fuente o proveniente del árbol BitKeeper. Para usos en producción, MySQL AB no aconseja utilizar un servidor MySQL generado por el usuario a partir del código fuente. Normalmente, es mejor emplear distribuciones binarias precompiladas de MySQL las cuales MySQL AB genera específicamente para un rendimiento óptimo en Windows. Las instrucciones para instalar distribuciones binarias se enecuentran en Sección 2.3, “Instalar MySQL en Windows”.

Para generar MySQL en Windows a partir del código fuente, se necesita que el siguiente compilador y recursos estén disponibles en el sistema:

- Compilador VC++ 6.0 (actualizado con SP4 o SP5 y paquete pre-procesador). El paquete pre-procesador se necesita para el macro assembler. Para más detalles consulte http://msdn.microsoft.com/vstudio/downloads/updates/sp/vs6/sp5/faq.aspx.
- Aproximadamente 45MB de espacio en disco.
- 64MB de RAM.

También se necesitará una distribución MySQL de código fuente para Windows. Hay dos formas de obtener una distribución de código fuente de MySQL 5.0:

1. Obtener una distribución de código fuente preparada por MySQL AB desde http://dev.mysql.com/downloads/.
2. Preparar una distribución de código fuente a partir del último arbol BitKeeper de código en desarrollo. Si se planea hacer esto, habrá que crear el paquete en un sistema Unix y luego transferirlo al sistema Windows. (La razón para esto es que algunas de las configuraciones y etapas de la generación requieren herramientas que funcionan solamente en Unix.) El uso de BitKeeper requiere:
   - Un sistema ejecutando Unix o un sistema operativo similar, como Linux.
   - Bitkeeper 3.0 instalado en ese sistema. Consulte Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo” para instrucciones sobre la descarga e instalación de BitKeeper.

si se está empleando una distribución de código fuente Windows, se puede ir directamente a Sección 2.8.6.1, “Generar MySQL usando VC++”. Para generar a partir del árbol de BitKeeper, proceder según Sección 2.8.6.2, “Crear un paquete de código fuente Windows a partir de la última fuente de desarrollo”.

Si algo no funciona como se esperaba, o se tienen sugerencias para hacer respecto a formas de mejorar el actual proceso de compilación en Windows, debe enviarse un mensaje a la lista de correo `win32`. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”.

2.8.6.1. Generar MySQL usando VC++

Note: Los ficheros de espacio de trabajo (workspace) de VC++ para MySQL 5.0 son compatibles con Microsoft Visual Studio 6.0 y ediciones posteriores (7.0 / .NET); estos ficheros son probados por el personal de MySQL AB antes de cada release.

Debe seguirse este procedimiento para generar MySQL:

1. Crear un directorio de trabajo (por ejemplo `C:\workdir`).
2. Expandir la distribución de código fuente en el mencionado directorio, utilizando `WinZip` u otra herramienta para Windows que pueda leer ficheros `.zip`.

3. Ejecutar Visual Studio
4. En el menú File, seleccionar Open Workspace .
5. Abrir el espacio de trabajo `mysql.dsw` que se hallará en el directorio de trabajo.
6. En el menú Build, seleccionar el menú Set Active Configuration.
7. Seleccionar mysqld - Win32 Debug y hacer click en OK.
8. Presionar **F7** para comenzar la compilación del servidor de depuración, bibliotecas, y algunas aplicaciones cliente.
9. Del mismo modo se compila la versión release.
10. Las versiones de depuración de los programas y las bibliotecas se encuentran en los directorios `client_debug` y `lib_debug`. Las versiones release de los programas y las bibliotecas se encuentran en los directorios `client_release` y `lib_release`. Si se desean generar tanto la versión de depuración como la de release, se puede seleccionar la opción Build All del menú Build.
11. Probar el servidor. El servidor generado mediante las instrucciones anteriores espera que el directorio de MySQL y el directorio de datos sean `C:\mysql` y `C:\mysql\data` por defecto. Si se desea probar el servidor empleado el directorio raíz del árbol de código fuente como directorio base y de datos, se le deben indicar sus rutas al servidor. Esto puede hacerse desde la línea de ocmandos con las opciones `--basedir` y `--datadir`, o colocando las opciones apropiadas en un fichero de opciones (el fichero `my.ini` dentro del directorio Windows o bien `C:\my.cnf`). También podría especificarse la ruta a un directorio de datos existente con anterioridad.
12. Iniciar el servidor desde el directorio `client_release` o `client_debug` dependiendo de cuál se quiera utilizar. Las instrucciones generales para el inicio del servidor se encuentran en Sección 2.3, “Instalar MySQL en Windows”. Las instrucciones deberán adaptarse si se desea utilizar un directorio base o directorio de datos diferente.
13. Cuando el servidor se esté ejecutando en modo autónomo o como un servicio basado en la configuración establecida, intentar conectarse a él desde la utilidad interactiva de línea de comandos `mysql`, que se encuentra en el directorio `client_release` o `client_debug`.

Cuando se esté satisfecho con el funcionamiento de los programas compilados, detener el servidor. Entonces, instalar MySQL de la siguiente manera:

1. Crear los directorios donde se desea instalar MySQL. Por ejemplo, para instalarlo en `C:\mysql`, emplear estos comandos:

```
C:\> mkdir C:\mysql
C:\> mkdir C:\mysql\bin
C:\> mkdir C:\mysql\data
C:\> mkdir C:\mysql\share
C:\> mkdir C:\mysql\scripts
```

   Si se desean compilar otros clientes y enlazarlos con MySQL, se deberían también crear varios directorios adicionales:

```
C:\> mkdir C:\mysql\include
C:\> mkdir C:\mysql\lib
C:\> mkdir C:\mysql\lib\debug
C:\> mkdir C:\mysql\lib\opt
```

   Si se desean efectuar pruebas de rendimiento de MySQL, crear este directorio:

```
C:\> mkdir C:\mysql\sql-bench
```

Las pruebas de rendimiento requieren soporte para Perl. Consulte Sección 2.13, “Notas sobre la instalación de Perl”.

2. Desde el directorio `workdir`, deben copiarse dentro de `C:\mysql` los siguientes directorios:

```
C:\> 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
```

Si se desean compilar otros clientes y enlazarlos a MySQL, también se deberían copiar varias bibliotecas y ficheros de cabecera:

```
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
```

Si se desean efectuar pruebas de rendimiento de MySQL, también debe hacerse lo siguiente:

```
C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
```

Configurar e iniciar el servidor del mismo modo que para la distribución binaria para Windows. Consulte Sección 2.3, “Instalar MySQL en Windows”.

2.8.6.2. Crear un paquete de código fuente Windows a partir de la última fuente de desarrollo

Para crear un paquete de código fuente Windows a partir del árbol BitKeeper actual, deben seguirse las siguientes instrucciones. Este procedimiento debe llevarse a cabo en un sistema con Unix o un sistema operativo estilo Unix. Por ejemplo, este procedimiento funciona bien en Linux.

1. Copiar el árbol de código fuente BitKeeper para MySQL 5.0. Para más información sobre cómo copiar el árbol de código fuente, consulte las instrucciones en Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”.

2. Configurar y generar la distribución de modo que se tenga un servidor ejecutable para trabajar con él. Una forma de lograr esto es ejecutar el siguiente comando en el directorio de más alto nivel del árbol de código fuente:

```
shell> ./BUILD/compile-pentium-max
```

3. Luego de asegurarse que el proceso de generación se completó con éxito, ejecutar el siguiente script en el directorio de más alto nivel del árbol de código fuente:

```
shell> ./scripts/make_win_src_distribution
```

Este script crea un paquete de código fuente Windows para ser usado en un sistema Windows. Se le pueden pasar diferentes opciones al script según las necesidades que se tengan. El script acepta las siguientes opciones:

- `--help`

  Muestra un mensaje de ayuda.

- `--debug`

  Imprime información acerca de las operaciones que realiza el script, sin crear el paquete.

- `--tmp`

  Especifica una ubicación temporal.

- `--suffix`

  Sufijo para el nombre del paquete.

- `--dirname`

  Nombre del directorio (intermedio) para copiar ficheros.

- `--silent`

  No imprime la lista de ficheros procesados.

- `--tar`

  Crea un paquete `tar.gz` en lugar de uno `.zip`.

Por defecto, `make_win_src_distribution` crea un fichero en formato Zip con el nombre `mysql-`*`VERSION`*`-win-src.zip`, donde *`VERSION`* es la versión del árbol de código fuente.

4. Copiar o subir al ordenador Windows el paquete de código fuente Windows que se acaba de crear. Para compilarlo, utilizar las instrucciones en Sección 2.8.6.1, “Generar MySQL usando VC++”.

2.8.7. Compilar los clientes de MySQL en Windows

En los ficheros de código fuente, se debería incluir `my_global.h` antes de `mysql.h`:

```
#include <my_global.h>
#include <mysql.h>
```

`my_global.h` incluye cualquier otro fichero necesario para compatibilidad con Windows (como `windows.h`) si el programa se compilará en Windows.

Se puede enlazar el código con la biblioteca dinámica `libmysql.lib`, la cual es solamente un wrapper para cargar `libmysql.dll` sobre demanda, o con la biblioteca estática `mysqlclient.lib`.

Las bibliotecas cliente de MySQL están compiladas como bibliotecas con uso de subprocesos, por lo tanto el código del usuario también debería ser multi-hilo.

2.9. Puesta en marcha y comprobación después de la instalación

Luego de instalar MySQL, hay algunos temas a los que hay que dirigir la atención. Por ejemplo, en Unix, se debería inicializar el directorio de datos y crear las tablas de permisos MySQL. En todas las plataformas, una cuestión importante de seguridad es que, inicialmente, las cuentas en las tablas de permisos no tienen contraseñas. Se deberían asignar contraseñas para evitar accesos no autorizados al servidor MySQL. Se pueden crear tablas de zonas horarias (time zone) para habilitar el reconocimiento de zonas horarias con nombre. (Actualmente, estas tablas solamente pueden llenarse en Unix. Este problema será revisado pronto en Windows).

Las siguientes secciones incluyen procedimientos de pos-instalación que son específicos de sistemas Windows y Unix. Otra sección, Sección 2.9.2.3, “Arrancar y resolver problemas del servidor MySQL”, se aplica a todas las plataformas; describe que hacer si ocurren problemas al tratar de iniciar el servidor. Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL” también se aplica a todas las plataformas. Se deberían seguir estas instrucciones para asegurarse de que las cuentas MySQL están correctamente protegidas mediante contraseñas.

Cuando se esté listo para crear cuentas de usuario adicioniales, se podrá encontrar información sobre el sistema de control de acceso de MySQL y administración de cuentas en Sección 5.6, “El sistema de privilegios de acceso de MySQL” y Sección 5.7, “Gestión de la cuenta de usuario MySQL”.

2.9.1. Pasos a seguir después de la instalación en Windows

En Windows, el directorio de datos y la tabla de permisos no necesitan ser creados. MySQL para Windows incluye la tabla de permisos con un conjunto de cuentas ya inicializadas en la base de datos `mysql`, dentro del directorio de datos. Al contrario que en Unix, no se debe ejecutar el script `mysql_install_db`. Sin embargo, si no se instala MySQL utilizando el ASistente de Instalación, se deberían establecer contraseñas para proteger las cuentas. Consulte Sección 2.3.4.1, “Introducción”. El procedimiento para hacerlo está en Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”.

Antes de establecer passwords, se podría desear ejecutar algunos programas cliente para asegurarse de que hay conexión con el servidor y que éste se encuentra funcionando correctamente. Tras cerciorarse de que el servidor está ejecutándose (consultar Sección 2.3.10, “Arrancar el servidor la primera vez”), se utilizarían los siguientes comandos para verificar que se puede traer información desde el mismo. La salida por pantalla debe ser similar a esto:

```
C:\> C:\mysql\bin\mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+

C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+

C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
+------+-------+------+
| host | db    | user |
+------+-------+------+
| %    | test% |      |
```

```
+------+-------+------+
```

Si se está empleando una versión de Windows que soporta servicios, y se desea que el servidor se ejecute automáticamente al iniciarse Windows, consulte Sección 2.3.12, “Arrancar MySQL como un servicio de Windows”.

2.9.2. Pasos a seguir después de la instalación en Unix

Luego de instalar MySQL en Unix, se necesita inicializar las tablas de permisos, ejecutar el servidor, y asegurarse de que éste funciona correctamente. También se podría desear que el servidor se inicie y detenga automáticamente cuando lo haga el sistema. Se deben asignar contraseñas a las cuentas en las tablas de permisos.

En Unix, las tablas de permisos se configuran mediante el programa `mysql_install_db`. En algunos métodos de instalación, este programa se ejecuta automáticamente.

- Si se instala MySQL en Linux a partir de una distribución RPM, el servidor RPM ejecuta `mysql_install_db`.
- Si se instala MySQL en Mac OS X a partir de una distribución PKG, el instalador ejecuta `mysql_install_db`.

En otro caso, será necesario ejecutar manualmente `mysql_install_db`.

El siguiente procedimiento describe cómo inicializar las tablas de permisos (si no se hizo anteriormente) y luego iniciar el servidor. También sugiere posibles comandos a utilizar para verificar que el servidor está accesible y en correcto funcionamiento. Para información relativa a iniciar y detener el servidor automáticamente, consulte Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente”.

Luego de completar el procedimiento y tener el servidor en funcionamiento, se deberían asignar contraseñas a las cuentas creadas por `mysql_install_db`. Las instrucciones para hacerlo se hallan en Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”.

En los ejemplos mostrados, el servidor se ejecuta bajo el identificador de usuario de la cuenta de inicio de sesión `mysql`. Se asume que dicha cuenta existe. La cuenta `mysql` puede haber sido creda especialmente o bien originarse al cambiar el nombre de una cuenta existente.

1. Posicionarse en el nivel más alto del directorio de instalación de MySQL, representado por *`BASEDIR`*:

   ```
   shell> cd BASEDIR
   ```

   `BASEDIR` muy probablemente se reemplace por algo similar a `/usr/local/mysql` o `/usr/local`. Los siguientes pasos asumen que se está en este directorio.

2. De ser necesario, ejecutar el programa `mysql_install_db` para establecer las tablas de permisos MySQL iniciales, las que contienen los privilegios que determinan qué usuarios están autorizados a conectarse al servidor. Habrá que ejecutarlo si se utilizó una distribución que no lo hace automáticamente.

   Por lo general, `mysql_install_db` sólo requiere ser ejecutado la primera vez que se instala MySQL, de modo que este paso puede obviarse si se está actualizando una instalación existente. No obstante, `mysql_install_db` no sobreescribe ninguna tabla, por lo tanto, es seguro utilizarlo en cualquier circunstancia.

   Para inicializar las tablas de permisos se utilizará uno de los siguientes comandos, dependiendo de si `mysql_install_db` se encuentra en el directorio `bin` o `scripts`:

   ```
   ```

```
shell> bin/mysql_install_db --user=mysql
shell> scripts/mysql_install_db --user=mysql
```

El script `mysql_install_db` crea el directorio de datos, la base de datos `mysql` que almacena todos los privilegios para el servidor, y la base de datos `test` que puede emplearse para probar MySQL. El script también crea entradas en la tabla de permisos para la cuenta `root` y la cuenta de usuario anónimo. Las cuentas no están protegidas por contraseña en un principio. Una descripción de los permisos que tienen se encuentra en Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”. Brevemente, estos privilegios le permiten al usuario `root` de MySQL hacer cualquier cosa, y le permiten a cualquier usuario de MySQL crear o utilizar bases de datos cuyo nombre sea `test` o comience con `test_`.

Es importante asegurarse de que los directorios y ficheros de bases de datos tienen como propietario a la cuenta de inicio de sesión `mysql`, para que el servidor tenga acceso de lectura y escritura a los mismos. Para cerciorarse de esto, si `mysql_install_db` se ejecuta mientras se está como `root` del sistema operativo, hay que usar la opción `--user` como se muestra. En otro caso, el script se deberá ejecutar mientras se está como usuario `mysql` del sistema operativo, en cuyo caso se puede omitir la opción `--user`.

`mysql_install_db` crea varias tablas en la base de datos `mysql`, incluyendo `user`, `db`, `host`, `tables_priv`, `columns_priv`, y `func`, entre otras. Consulte Sección 5.6, “El sistema de privilegios de acceso de MySQL” para una descripción completa.

Si no se desea tener la base de datos `test`, se la puede eliminar con `mysqladmin -u root drop test` luego de iniciar el servidor.

Si ocurren problemas con `mysql_install_db`, consulte Sección 2.9.2.1, “Problemas en la ejecución de `mysql_install_db`”.

Hay algunas alternativas para ejecutar el script `mysql_install_db` que se provee con la distribución de MySQL:

- Si se desea que los privilegios iniciales sean diferentes de los establecidos por defecto, se puede modificar `mysql_install_db` antes de ejecutarlo. Sin embargo, es preferible utilizar `GRANT` y `REVOKE` para modificar los privilegios *después* que las tablas de permisos se hayan creado. En otras palabras, se puede ejecutar `mysql_install_db`, y posteriormente emplear `mysql -u root mysql` para conectarse al servidor como usuario `root` de MySQL y aplicar las sentencias `GRANT` y `REVOKE` que sean necesarias.

  Si se desea instalar MySQL en varios ordenadores con los mismos privilegios, se pueden colocar las sentencias `GRANT` y `REVOKE` en un fichero y ejecutarlo como un script utilizando `mysql` después de ejecutar `mysql_install_db`. Por ejemplo:

  ```
  shell> bin/mysql_install_db --user=mysql
  shell> bin/mysql -u root < your_script_file
  ```

  De este modo se evita teclear las sentencias en cada ordenador.

- Es posible regenerar las tablas de permisos completamente aunque ya estén creadas. Se puede necesitar esto si se está aprendiendo a usar `GRANT` y `REVOKE` y se han hecho tantas modificaciones tras la ejecución de `mysql_install_db` que se desea vaciar las tablas de permisos y comenzar de nuevo.

  Para regenerar las tablas de permisos, eliminar todos los ficheros `.frm`, `.MYI`, y `.MYD` en el directorio que contiene la base de datos `mysql`. (Este es el directorio llamado `mysql` dentro del

directorio de datos, el cual aparece listado como el valor `datadir` cuando se ejecuta `mysqld --help`.) Luego debe ejecutarse nuevamente el script `mysql_install_db`

- Se puede iniciar `mysqld` manualmente utilizando la opción `--skip-grant-tables` e ingresar los permisos manualmente utilizando `mysql`:

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

Desde `mysql`, ejecutar manualmente los comandos contenidos en `mysql_install_db`. Al finalizar hay que asegurarse de ejecutar `mysqladmin flush-privileges` o `mysqladmin reload` para indicarle al servidor que lea nuevamente las tablas de permisos.

Nótese que al no utilizar `mysql_install_db`, no solamente hay que cargar manualmente el contenido de las tablas de permisos, sino que también hay que crearlas primero.

3. Iniciar el servidor MySQL:

```
shell> bin/mysqld_safe --user=mysql &
```

Es importante que el servidor MySQL sea ejecutado utilizando una cuenta de sistema operativo sin privilegios (distinta a `root`). Para cerciorarse de esto, se debe usar la opción `--user` si se ejecuta `mysql_safe` habiendo iniciado sesión del sistema operativo como `root`. En otro caso, se debería ejecutar el script mientras se ha iniciado sesión como `mysql`, en cuyo caso se puede omitir la opción `--user`.

Mayores instrucciones para ejecutar MySQL como un usuario sin privilegios se encuentran en Sección A.3.2, “Cómo ejecutar MySQL como usuario normal”.

Si se omite la creación de las tablas de permisos antes de llegar a este paso, aparecerá el siguiente mensaje en el fichero de registro de error cuando se inicie el servidor:

```
mysqld: Can't find file: 'host.frm'
```

Si ocurren otros problemas al iniciar el servidor, consulte Sección 2.9.2.3, “Arrancar y resolver problemas del servidor MySQL”.

4. Utilizar `mysqladmin` para verificar que el servidor se encuentra en ejecución. Los siguientes comandos proporcionan formas simples de saber si el servidor está activo y responde a conexiones:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

La salida producida por `mysqladmin version` varía dependiendo de la plataforma y la versión de MySQL, pero debería ser similar a esto:

```
shell> bin/mysqladmin version
mysqladmin  Ver 8.41 Distrib 5.0.9-beta, for pc-linux-gnu on i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version          5.0.9-beta-Max
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
```

Para ver qué otras tareas pueden hacerse con `mysqladmin`, se lo debe invocar con la opción `--help`.

5. Verificar que se pueda detener el servidor:

```
shell> bin/mysqladmin -u root shutdown
```

6. Verificar que se pueda reiniciar el servidor. Hacerlo mediante `mysqld_safe` o invocando directamente a `mysqld`. Por ejemplo:

```
shell> bin/mysqld_safe --user=mysql --log &
```

Si `mysqld_safe` fallara, consultar Sección 2.9.2.3, “Arrancar y resolver problemas del servidor MySQL”.

7. Ejecutar algunas pruebas sencillas para comprobar que se puede traer información desde el servidor. La salida debería ser similar a lo que se muestra aquí:

```
shell> bin/mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+

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                      |
+---------------------------+

shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+------+--------+------+
| host | db     | user |
+------+--------+------+
| %    | test   |      |
| %    | test_% |      |
```

```
+------+--------+------+
```

8. Hay un conjunto de pruebas de rendimiento en el directorio `sql-bench` (dentro del directorio de instalación de MySQL) que puede utilizarse para comparar el desempeño de MySQL en distintas plataformas. Este conjunto de pruebas está escrito en Perl. Utiliza el módulo Perl DBI para proporcionar una interface independiente de la base de datos a varias bases de datos, y algunos otros módulos adicionales de Perl también son requeridos para ejecutar las pruebas. Se deben tener los siguientes módulos instalados:

   ```
   DBI
   DBD::mysql
   Data::Dumper
   Data::ShowTable
   ```

   Estos módulos pueden ser obtenidos desde CPAN (http://www.cpan.org/). También, consulte Sección 2.13.1, “Instalación de Perl en Unix”.

   El directorio `sql-bench/Results` contiene los resultados de muchas ejecuciones a través de diferentes bases de datos y plataformas. Para ejecutar todos los tests, deben introducirse estos comandos:

   ```
   shell> cd sql-bench
   shell> perl run-all-tests
   ```

   Si no se encuentra el directorio `sql-bench`, probablemente se ha instalado MySQL empleando otros ficheros RPM que no son el RPM de código fuente (El RPM de código fuente incluye el directorio de pruebas de rendimiento `sql-bench`) En este caso, se deberá primero instalar el conjunto de pruebas de rendimiento antes de poder utilizarlo. Para MySQL 5.0, hay un RPM separado con pruebas de rendimiento llamado `mysql-bench-`*`VERSION`*`-i386.rpm` que contiene el código de prueba y los datos.

   Si se posee una distribución de código fuente, también hay pruebas en su subdirectorio `tests`. Por ejemplo, para ejecutar `auto_increment.tst`, debe ejecutarse este comando desde el directorio de más alto nivel en la distribución de código fuente:

   ```
   shell> mysql -vvf test < ./tests/auto_increment.tst
   ```

   El resultado esperado del test se encuentra en el fichero `./tests/auto_increment.res`.

9. En este punto, se debería tener el servidor en funcionamiento. Sin embargo, ninguna de las cuentas iniciales de MySQL tiene una contraseña, así que se les debe asignar empleando las instrucciones halladas en Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”.

En MySQL 5.0, el procedimiento de instalación crea zonas horarias en la base de datos `mysql`. No obstante, se deben llenar las tablas en forma manual. Las instrucciones para hacerlo se encuentran en Sección 5.9.8, “Soporte de zonas horarias en el servidor MySQL”.

2.9.2.1. Problemas en la ejecución de `mysql_install_db`

El propósito del script `mysql_install_db` es generar tablas de permisos MySQL nuevas. No sobreescribe las tablas de permisos existentes ni afecta a otros datos.

Para crear nuevamente las tablas de privilegios, primero debe detenerse el servidor `mysqld` si está ejecutándose. Luego se renombra -para preservarlo- el directorio `mysql` que está dentro del directorio de datos, y se ejecuta `mysql_install_db`. Por ejemplo:

```
shell> mv mysql-data-directory/mysql mysql-data-directory/mysql-old
shell> mysql_install_db --user=mysql
```

Esta sección detalla problemas que podrían hallarse cuando se ejecute `mysql_install_db`:

- **`mysql_install_db` falla al instalar las tablas de permisos**

  `mysql_install_db` puede fallar al instalar las tablas de permisos y finalizar despues de mostrar los siguientes mensajes:

  ```
  Starting mysqld daemon with databases from XXXXXX
  mysqld ended
  ```

  En tal caso, se debe examinar el fichero de registro de errores muy cuidadosamente. El registro podría hallarse en el directorio `XXXXXX` con un nombre similar al mensaje de error, y debería indicar el motivo por el que `mysqld` no se inició. Si no fuera de ayuda, habrá que enviar un informe de error incluyendo el texto del registro. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

- **Ya hay un proceso `mysqld` en ejecución**

  Esto indica que el servidor se está ejecutando, en cuyo caso las tablas de permisos probablemente ya se crearon. De ser así, no es necesario ejecutar `mysql_install_db` en absoluto, porque sólo se hace una vez (cuando se instala MySQL la primera vez).

- **No es posible instalar un segundo servidor `mysqld` cuando hay uno en ejecución.**

  Esto puede ocurrir cuando se tiene una instalación de MySQL y se desea realizar una nueva instalación en una ubicación diferente. Por ejemplo, se podría tener una instalación en producción y crear una segunda instalación con fines de prueba. Generalmente, el problema es que el segundo servidor intenta utilizar una interfaz de red que está siendo usada por el primero. En este caso se debería ver uno de los siguientes mensajes de error:

  ```
  Can't start server: Bind on TCP/IP port:
  Address already in use
  Can't start server: Bind on unix socket...
  ```

  Consulte Sección 5.11, “Ejecutar más de un servidor MySQL en la misma máquina” para ver instrucciones sobre la instalación de múltiples servidores.

- **No se tiene acceso de escritura a `/tmp`**

  Si no se tiene acceso de escritura para crear ficheros temporales o un fichero de socket Unix en la ubicación por defecto (el directorio `/tmp`), ocurrirá un error al ejecutar `mysql_install_db` o el servidor `mysqld`.

  Se pueden especificar distintos directorios temporales y ubicaciones para ficheros socket de Unix ejecutando los siguientes comandos antes de iniciar `mysql_install_db` o `mysqld`:

  ```
  shell> TMPDIR=/some_tmp_dir/
  shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
  shell> export TMPDIR MYSQL_UNIX_PORT
  ```

some_tmp_dir debería ser la ruta completa a algún directorio para el cual se tenga permiso de escritura.

Luego de hacer esto se debería estar en condiciones de ejecutar `mysql_install_db` e iniciar el servidor con estos comandos:

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

Si `mysql_install_db` está ubicado en el directorio `scripts`, debe modificarse el primer comando para que diga `scripts/mysql_install_db`.

Consulte Sección A.4.5, “Cómo proteger o cambiar el fichero socket de MySQL `/tmp/mysql.sock`”. Consulte Apéndice E, *Variables de entorno*.

2.9.2.2. Arrancar y parar MySQL automáticamente

Generalmente, el servidor `mysqld` se inicia de alguna de estas formas:

- Invocando `mysqld` directamente. Esto funciona en cualquier plataforma.
- Ejecutando el servidor MySQL como un servicio de Windows. Esto puede hacerse en versiones de Windows que soporten servicios (como NT, 2000, XP, y 2003). El servicio se puede configurar para que inicie el servidor automáticamente cuando arranca Windows, o como un servicio manual que se inicia a pedido. Para instrucciones, consulte: Sección 2.3.12, “Arrancar MySQL como un servicio de Windows”.
- Invocando `mysqld_safe`, el cual intenta determinar las opciones apropiadas para `mysqld` y entonces lo ejecuta con dichas opciones. Este script se usa en sistemas basados en BSD Unix. Consulte Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.
- Invocando `mysql.server`. Este script se usa principalmente al iniciar y detener el sistema en sistemas que emplean directorios de ejecución al estilo System V, donde usualmente se instala bajo el nombre `mysql`. El script `mysql.server` inicia el servidor mediante la ejecución de Sección 5.1.4, “El script `mysql.server` para el arranque del servidor”.
- En Mac OS X, se instala paquete separado llamado MySQL Startup Item para habilitar el inicio automático de MySQL junto con el sistema. El Startup Item inicia el servidor invocando a `mysql.server`. Consulte Sección 2.5, “Instalar MySQL en Mac OS X” para más detalles.

Los scripts `mysql.server` y `mysqld_safe` y el Mac OS X Startup Item se pueden utilizar para iniciar el servidor manualmente, o en forma automática al inicio del sistema. `mysql.server` y el Startup Item también se emplean para detener el servidor.

Para iniciar o detener el servidor manualmente empleando el script `mysql.server`, se lo debe invocar con los argumentos `start` o `stop`:

```
shell> mysql.server start
shell> mysql.server stop
```

Antes de que `mysql.server` inicie el servidor, se posiciona en el directorio de instalación de MySQL, y luego ejecuta `mysqld_safe`. Si se desea que el servidor se ejecute como un usuario específico, debe agregarse la correspondiente opción `user` al grupo `[mysqld]` del fichero de opciones `/etc/my.cnf`, como se muestra más adelante en esta sección. (Es posible que haya que editar `mysql.server` si se instaló una distribución binaria de MySQL en una ubicación no estándar. La modificación consiste en hacer `cd` al directorio apropiado antes de ejecutar `mysqld_safe`. En caso de hacer esto, tener en cuenta

que la versión modificada de `mysql.server` puede ser sobreescrita si se actualiza MySQL en el futuro; se debería hacer una copia de la versión modificada para restaurarla.)

`mysql.server stop` detiene el servidor mediante el envío de una señal. También se lo puede detener manualmente ejecutando `mysqladmin shutdown`.

Para iniciar y detener MySQL automáticamente, se necesita agregar los comandos de inicio y detención en los sitios apropiados de los ficheros `/etc/rc*`.

Si se emplea el paquete RPM de servidor para Linux (`MySQL-server-VERSION.rpm`), el script `mysql.server` se instala en el directorio `/etc/init.d` con el nombre `mysql`. No se necesita instalarlo manualmente. Consulte Sección 2.4, “Instalar MySQL en Linux” para más información sobre los paquetes Linux RPM.

Algunos vendedores proveen paquetes RPM que instalan un script de inicio con un nombre distinto, como `mysqld`.

Si se instala MySQL desde una distribución de código fuente o mediante una distribución binaria que no instala `mysql.server` automáticamente, se lo puede instalar manualmente. El script se encuentra en el directorio `support-files` dentro del directorio de instalación de MySQL o en el árbol de código fuente de MySQL.

Para instalar `mysql.server` manualmente, se los debe copiar en el directorio `/etc/init.d` con el nombre `mysql`, y luego hacerlo ejecutable. Esto se hace posicionándose dentro del directorio donde está `mysql.server` y ejecutando estos comandos:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Los sistemas Red Hat antiguos utilizan el directorio `/etc/rc.d/init.d` en lugar de `/etc/init.d`. Los comandos anteriores deben modificarse de acuerdo a esto. Alternativamente, puede crearse primero `/etc/init.d` como un vínculo simbólico que apunte a `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

Luego de instalar el script, los comandos necesarios para activarlo en el arranque del sistema dependen del sistema operativo. En Linux, puede utilizarse `chkconfig`:

```
shell> chkconfig --add mysql
```

En algunos sistemas Linux, el siguiente comando tmabién parece ser necesario para habilitar completamente al script `mysql`:

```
shell> chkconfig --level 345 mysql on
```

En FreeBSD, los scripts de inicio generalmente se encuentran en `/usr/local/etc/rc.d/`. La página de manual `rc(8)` establece que los scripts en dicho directorio se ejecutan solamente si su nombre base concuerda con el patrón de nombre de fichero shell `*.sh`. Cualquier otro fichero o directorio presente dentro del directorio es ignorado sin advertencias. En otras palabras, en FreeBSD, se debería instalar el script `mysql.server` como `/usr/local/etc/rc.d/mysql.server.sh` para habilitar el inicio automático.

Como una alternativa a la configuración anterior, algunos sistemas operativos también emplean `/etc/rc.local` o `/etc/init.d/boot.local` para iniciar servicios adicionales en el arranque del sistema.

Para iniciar MySQL utilizando este método, se puede agregar un comando como el siguiente al fichero de inicio apropiado:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

En otros sistemas operativos, consultar la documentación para ver cómo instalar scripts de inicio.

Se pueden agregar opciones a `mysql.server` en un fichero global `/etc/my.cnf`. Un típico fichero `/etc/my.cnf` podría verse como este:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

El script `mysql.server` acepta las siguientes opciones: `basedir`, `datadir`, y `pid-file`. Si se utilizan, *deben* estar en un fichero de opciones, no en la línea de comandos. `mysql.server` sólo acepta en la línea de comandos las opciones `start` y `stop`.

La siguiente tabla indica los grupos del fichero de opciones que son leidos por el servidor y por cada script de inicio.

Script	Grupos de opciones
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-versión principal]`
`mysql.server`	`[mysqld]`, `[mysql.server]`, `[server]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`

`[mysqld-versión principal]` significa que los grupos con nombres como `[mysqld-4.0]`, `[mysqld-4.1]`, y `[mysqld-5.0]` son leídos por servidores con números de versión 4.0.x, 4.1.x, 5.0.x y así sucesivamente. Esta característica sirve para especificar opciones que serán leídas solamente por servidores pertenecientes a releases de una determinada serie.

Por razones de compatibilidad hacia atrás, `mysql.server` también lee el grupo `[mysql_server]` y `mysqld_safe` también lee el grupo `[safe_mysqld]`. No obstante, cuando se emplee MySQL 5.0, se debería actualizar el fichero de opciones para que contenga los grupos `[mysql.server]` y `[mysqld_safe]`.

Consulte Sección 4.3.2, “Usar ficheros de opciones”.

2.9.2.3. Arrancar y resolver problemas del servidor MySQL

Si ocurren problemas durante el inicio del servidor, inténtese lo siguiente:

- Especificar cualquier opción especial necesaria para el motor de almacenamiento en uso.
- Asegurarse de que el servidor conoce dónde se encuentra el directorio de datos.
- Cerciorarse de que el servidor pueda utilizar el directorio de datos. El propietario y los permisos sobre el directorio de datos y su contenido deben establecerse de forma que el servidor sea capaz de acceder a ellos y modificarlos.
- Inspeccionar el registro de errores para ver porqué el servidor no se inicia.

- Verificar que están disponibles las interfaces de red que el servidor intenta utilizar.

Algunos motores de almacenamiento tienen opciones que controlan su comportamiento. Se puede crear un fichero `my.cnf` y establecer opciones de inicio para el motor que se planea utilizar. Si se van a usar motores de almacenamiento con soporte para tablas transaccionales (`InnoDB`, `BDB`), hay que asegurarse de que se han configurado según lo deseado antes de iniciar el servidor:

- Si se están empleando tablas `InnoDB`, hay que remitirse a las opciones de inicio específicas. En MySQL 5.0, `InnoDB` utiliza valores por defecto para sus opciones de configuración si no se indica ninguna. Consulte Sección 15.3, “Configuración de `InnoDB`”.

- Si se están usando tablas `BDB` (Berkeley DB), será necesario familiarizarse con las diferentes opciones de inicio específicas de `BDB`. Consulte Sección 14.4.3, “Opciones de arranque de `BDB`”.

Cuando el servidor `mysqld` se inicia, se posiciona en el directorio de datos. Aquí es donde espera encontrar bases de datos y donde grabará sus ficheros de registro. En Unix, también grabará aquí el fichero pid (process ID, o identificador de proceso).

La ubicación del directorio de datos se establece en forma fija cuando se compila el servidor. Aquí es donde, por defecto, buscará el directorio de datos. Si el mismo se encuentra en otra parte del sistema, el servidor no funcionará correctamente. Se puede conocer la ubicación por defecto ejecutando `mysqld` con las opciones `--verbose` y `--help`.

Si los valores por defecto no coinciden con la instalación realizada en el sistema, se los puede sustituir especificando opciones para `mysqld` o `mysqld_safe` en la linea de comandos. También se pueden colocar en un fichero de opciones.

Para especificar explícitamente la ubicación del directorio de datos, se emplea la opción `--datadir`. Sin embargo, normalmente se le puede indicar a `mysqld` la ubicación del directorio base donde está instalado MySQL, y el servidor buscará allí el directorio de datos. Esto se hace con la opción `--basedir` option.

Para verificar los efectos de especificar opciones de ruta, hay que invocar `mysqld` con dichas opciones seguidas de `--verbose` y `--help`. Por ejemplo, posicionándose donde `mysqld` está instalado, y ejecutando los siguientes comandos, se verán los efectos de iniciar el servidor en el directorio base `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

Se pueden suministrar otras opciones, como `--datadir`, pero hay que tener en cuenta que `--verbose` y `--help` deben aparecer en último lugar.

Una vez que se haya logrado determinar la configuración de ruta deseada, iniciar el servidor sin `--verbose` y `--help`.

Si `mysqld` ya está ejecutándose, se puede conocer la configuración de rutas que está usando mediante la ejecución de este comando:

```
shell> mysqladmin variables
```

O bien:

```
shell> mysqladmin -h host_name variables
```

`host_name` es el nombre del host del servidor MySQL.

Si se obtuviera el `Errcode 13` (que significa `Permission denied (permiso denegado)`) al iniciar `mysqld`, indica que los permisos de acceso al directorio de datos o a su contenido no permiten el acceso del servidor. En este caso, hay que cambiar los permisos sobre los directorios y ficheros involucrados para que el servidor tenga derecho a usarlos. También se puede iniciar el servidor bajo el usuario de sistema operativo `root`, pero esto puede traer aparejados problemas de seguridad y debería ser evitado.

En Unix, hay que posicionarse en el directorio de datos y verificar el propietario del directorio y su contenido para asegurarse de que el servidor tiene acceso. Por ejemplo, si el directorio de datos es `/usr/local/mysql/var`, usar este comando:

```
shell> ls -la /usr/local/mysql/var
```

Si el directorio de datos o sus ficheros o subdirectorios no tienen como propietario a la cuenta empleada para ejecutar el servidor, cambiar el propietario para que sea esa cuenta:

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

Si el servidor falla en iniciarse correctamente, verificar el fichero de registro de errores para ver si se puede encontrar la causa. Los ficheros de registro se localizan en el directorio de datos (generalmente, `C:\Program Files\MySQL\MySQL Server 5.0\data` en Windows, `/usr/local/mysql/data` en una distribución binaria de Linux, y `/usr/local/var` en una distribución de código fuente de Linux). Se buscan en el directorio de datos los ficheros con un nombre con la forma *`host_name`*`.err` y *`host_name`*`.log`, donde *`host_name`* es el nombre del host del servidor. Luego, examinar las últimas líneas de estos ficheros. En Unix, puede utilizarse `tail` para mostrarlas:

```
shell> tail host_name.err
shell> tail host_name.log
```

El registro de errores contiene información que indica el motivo por el cual el servidor no ha podido iniciarse. Por ejemplo, es posible ver algo como esto al examinarlo:

```
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
```

Esto significa que no se inició `mysqld` con la opción `--bdb-no-recover` y Berkeley DB halló algo incorrecto con sus propios ficheros de registro cuando intentó recuperar las bases de datos. Para que sea posible continuar, habría que mover los ficheros de registro Berkeley DB antiguos desde el directorio de bases de datos a alguna otra ubicación, donde puedan examinarse posteriormente. Los ficheros de registro `BDB` reciben nombres en secuencia comenzando en `log.0000000001`, donde el número se incrementa cada vez.

Si se está ejecutando `mysqld` con soporte para tablas `BDB` y `mysqld` realiza un volcado del núcleo al inicio, podría deberse a problemas con el registro de recuperación de `BDB`. En este caso, se puede intentar el inicio de `mysqld` con `--bdb-no-recover`. Si esto ayuda, entonces se deberían eliminar todos los ficheros de registro `BDB` del directorio de datos e intentar el inicio de `mysqld` nuevamente, sin la opción `--bdb-no-recover`.

Si ocurriese cualquiera de los siguientes errores, significa que algún otro programa (quizá otro servidor `mysqld`) está utilizando el puerto TCP/IP o socket Unix que `mysqld` intenta emplear:

```
Can't start server: Bind on TCP/IP port: Address already in use
```

```
Can't start server: Bind on unix socket...
```

Utilizar `ps` para determinar si se tiene otro servidor `mysqld` en ejecución. Si es así, detener el servidor antes de iniciar `mysqld` de nuevo. (si hay otro servidor ejecutándose, y realmente se desea tener múltiples servidores, se puede hallar información sobre cómo hacerlo en Sección 5.11, “Ejecutar más de un servidor MySQL en la misma máquina”.)

Si no hay otro servidor ejecutándose, inténtese ejecutar el comando `telnet nombre-de-host número-puerto-TCP-IP`. (El número de puerto MySQL por defecto es 3306). Luego presionar Enter un par de veces. Si no se obtiene un mensaje de error como `telnet: Unable to connect to remote host: Connection refused`, algún otro programa está ocupando el puerto TCP/IP que `mysqld` está intentando utilizar. Se necesitará determinar qué programa es y desactivarlo, o bien indicar a `mysqld` que escuche en un puerto diferente mediante la opción `--port`. En este caso, también se necesitará especificar el número de puerto en los programas cliente cuando se conecten al servidor a través de TCP/IP.

Otra razón por la que el puerto podría ser inaccesible es que se tenga un firewall que bloquee las conexiones a él. Si es así, modificar la configuración del firewall para permitir el acceso a ese puerto.

Si el servidor se inicia pero no es posible conectarse a él, habría que cerciorarse de que se tiene una entrada en `/etc/hosts` que se vea así:

```
127.0.0.1       localhost
```

Estre problema ocurre solamente en sistemas que no tienen una biblioteca para trabajo con subprocesos y para los cuales MySQL debe configurarse para usar MIT-pthreads.

Si no es posible iniciar `mysqld`, se puede generar un fichero de seguimiento para hallar el problema utilizando la opción `--debug`. Consulte Sección D.1.2, “Crear ficheros de traza”.

Consulte Sección 2.3.14, “Resolución de problemas en la instalación de MySQL bajo Windows” para obtener mayor información sobre la resolución de problemas en instalaciones Windows.

2.9.3. Hacer seguras las cuentas iniciales de MySQL

Una parte del proceso de instalación de MySQL es configurar la base de datos `mysql`, que contiene las tablas de permisos:

- Las distribuciones para Windows contiene tablas de permisos preinicializadas que se instalan automáticamente.
- En Unix, las tablas de permisos se llenan mediante el programa `mysql_install_db`. Algunos métodos de instalación ejecutan este programa en forma automática. Otros necesitan que sea ejecutado manualmente. Para más detalles, consulte Sección 2.9.2, “Pasos a seguir después de la instalación en Unix”.

Las tablas de permisos definen las cuentas de usuario MySQL iniciales y sus permisos de acceso. Estas cuentas tienen la siguiente configuración:

- Se crean dos cuentas con el nombre de usuario `root`. Son cuentas de superusuario que pueden realizar cualquier tarea. Inicialmente las cuentas `root` no tienen contraseñas, de forma que cualquier persona puede conectarse al servidor MySQL como `root` *sin una contraseña* y recibirá todos los privilegios.
  - En Windows, una cuenta `root` sirve para conectarse desde el ordenador local (localhost) y la otra permite conectarse desde cualquier ordenador.

  - En Unix, ambas cuentas `root` son para conexiones desde el ordenador local (localhost). Las conexiones deben establecerse desde el ordenador local especificando el nombre de host `localhost` para una de las cuentas, o el nombre propiamente dicho del host o número de IP para la otra.

- Se crean dos cuentas de usuario anónimo, cada una con un nombre de usuario vacío. Los usuarios anónimos no tienen contraseña, de modo que cualquier persona puede usarlos para conectarse al servidor MySQL.

  - En Windows, una cuenta anónima es para conexiones desde el ordenador local. Tiene todos los privilegios, exactamente como la cuenta `root`. La otra sirve para conectarse desde cualquier ordenador y tiene todos los permisos sobre la base de datos `test` u otras cuyo nombre comience con `test`.

  - En Unix, ambas cuentas anónimas son para conexiones desde el ordenador local (localhost). Las conexiones deben establecerse desde el ordenador local especificando el nombre de host `localhost` para una de las cuentas, o el nombre propiamente dicho del host o número de IP para la otra. Estas cuentas tienen todos los permisos sobre la base de datos `test` u otras cuyo nombre comience con `test`.

Como se advierte, ninguna de las cuentas iniciales tiene contraseña. Esto significa que la instalación de MySQL estará desprotegida hasta que:

- Si se desea evitar que los clientes se conecten como anónimos sin una contraseña, se les puede establecer contraseñas o bien eliminar las cuentas anónimas.

- Se deberían establecer contraseñas para las cuentas `root` de MySQL.

Las siguientes instrucciones describen cómo establecer contraseñas para las cuentas iniciales de MySQL, primero para las cuentas anónimas y luego para las cuentas `root`. En los ejemplos se debe reemplazar "*`newpwd`*" con el password que realmente se utilizará. También se instruye cómo eliminar las cuentas anónimas, si se prefiriera impedir completamente el acceso de usuarios anónimos.

Podría desearse posponer la aplicación de contraseñas hasta más tarde, para que no sea necesario ingresarlas mientras se desarrollan tareas adicionales de configuración o prueba. Sin embargo, hay que asegurarse de establecerlas antes de poner la instalación en trabajo de producción real.

Para 1800 proteger con contraseña las cuentas anónimas, puede emplearse tanto `SET PASSWORD` como `UPDATE`. En ambos casos, hay que asegurarse de cifrar el password utilizando la función `PASSWORD()`.

Para emplear `SET PASSWORD` en Windows, hacer lo siguiente:

```
shell> mysql -u root
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'%' = PASSWORD('newpwd');
```

Para emplear `SET PASSWORD` en Unix, hacer lo siguiente:

```
shell> mysql -u root
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

En la segunda sentencia `SET PASSWORD`, debe reemplazarse *`host_name`* con el nombre del host del servidor. Este es el nombre que aparece en la columna `Host` del registro correspondiente a `root` que no es `localhost` en la tabla `user`. Si no se puede determinar el nombre de este host, utilizar la siguiente sentencia antes que `SET PASSWORD`:

```
mysql> SELECT Host, User FROM mysql.user;
```

Localizar el registro que tiene a `root` en la columna `User` y cualquier otro excepto `localhost` en la columna `Host`. Entonces, utilizar ese valor de `Host` en la segunda sentencia `SET PASSWORD`.

La otra forma de asignar contraseñas a las cuentas anónimas es utilizando `UPDATE` para modificar directamente la tabla `user`. Hay que conectarse al servidor como `root` y emitir una sentencia `UPDATE` que asigne un valor a la columna `Password` en los registros apropiados de la tabla `user`. El procedimiento es igual en Windows y en Unix. La siguiente sentencia `UPDATE` asigna una contraseña a las dos cuentas anónimas de una sola vez:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

Luego de actualizar las contraseñas directamente en la tabla `user` empleando `UPDATE`, se le debe indicar al servidor que relea las tablas de privilegios con `FLUSH PRIVILEGES`. De otro modo, los cambios no tendrán efecto hasta que se reinicie el servidor.

Si en lugar de lo anterior se prefiere eliminar las cuentas anónimas, hay que hacer lo siguiente:

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

La sentencia `DELETE` se aplica tanto en Windows como en Unix. En Windows, si solamente se desean remover las cuentas anónimas que tengan los mismos privilegios que `root`, debe hacerse lo siguiente:

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

Esta cuenta permite el acceso anónimo con todos los privilegios, por lo tanto, al removerla se refuerza la seguridad.

A la cuenta `root` se le pueden asignar contraseñas en varias formas. En el tratamiento del tema que se hace a continuación se muestran tres métodos:

- Usar la sentencia `SET PASSWORD`
- Usar el programa cliente de línea de comandos `mysqladmin`
- Usar la sentencia `UPDATE`

Para asignar contraseñas empleando `SET PASSWORD`, hay que conectarse al servidor como `root` y emitir dos sentencias `SET PASSWORD`, asegurándose de cifrar la contraseña con la función `PASSWORD()`.

En Windows se hace así:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

En Unix, así:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

En la segunda sentencia `SET PASSWORD`, se debe reemplazar *`host_name`* con el nombre del host del servidor. Es el mismo nombre de host que se utilizó al asignar contraseñas a las cuentas anónimas.

Para establecer contraseñas en las cuentas `root` empleando `mysqladmin`, ejecutar los siguientes comandos:

```
  shell> mysqladmin -u root password "newpwd"
  shell> mysqladmin -u root -h host_name password "newpwd"
```

Estos comandos se aplican tanto a Windows como a Unix. En el segundo comando, *`host_name`* debe reemplazarse con el nombre del host del servidor. Las comillas dobles que encierran la contraseña no son siempre necesarias, pero se las debe usar si la contraseña contiene espacios u otros caracteres que sean especiales para el intérprete de comandos.

También puede usarse `UPDATE` para modificar directamente la tabla `user`. La siguiente sentencia `UPDATE` establece una contraseña para ambas cuentas `root` de una sola vez:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

La sentencia `UPDATE` se aplica tanto a Windows como a Unix.

Luego de establecer las contraseñas, se las deberá suministrar en cada conexión al servidor. Por ejemplo, si se desea emplear `mysqladmin` para detener el servidor, se debería hacer mediante este comando:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Nota: En caso de olvidar la contraseña de `root` despues de establecerla, el procedimiento para reinicializarla se cubre en Sección A.4.1, “Cómo reiniciar la contraseña de root”.

Para configurar nuevas cuentas, se debe usar la sentencia `GRANT`. Para instrucciones consulte Sección 5.7.2, “Añadir nuevas cuentas de usuario a MySQL”.

2.10. Aumentar la versión de MySQL

Como regla general, se recomienda que al actualizar de una serie a otra se pase a la serie inmediatamente superior sin saltar ninguna. Por ejemplo, si actualmente se está ejecutando MySQL 3.23 y se desea actualizar a una serie más moderna, debe pasarse a MySQL 4.0 y no 4.1 o 5.0.

Los siguientes puntos conforman una lista de lo que se debería hacer al llevar a cabo una actualización:

- Antes de actualizar de MySQL 4.1 a 5.0, debe leerse Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0” y Apéndice C, *Historial de cambios de MySQL*. Estos proveen información acerca de características que son nuevas o diferentes respecto a las halladas en MySQL 4.1. Si se deseara actualizar desde una serie anterior a MySQL 4.1, se debería actualizar a la serie inmediatamente superior cada vez hasta llegar a MySQL 4.1, entonces se procedería con la actualización a MySQL 5.0. Para más información sobre actualizaciones desde series anteriores a MySQL 4.1, consulte Manual de referencia de MySQL 4.1.

- Antes de llevar a cabo una actualización, hay que hacer copia de respaldo de las bases de datos.
- Si se está ejecutando MySQL Server en Windows, consulte Sección 2.3.15, “Aumentar la versión de MySQL en Windows”.
- Una actualización a MySQL 5.0 desde la versión 4.1 implica cambios en las tablas de permisos almacenadas en la base de datos `mysql`; donde se agregaron columnas y tablas para soportar las nuevas características. Para sacar partido de estas características, hay que cerciorarse de que las tablas de permisos están actualizadas. El procedimiento para actualizar las tablas de permisos se describe en Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”. Antes de empezar, las tablas se pueden respaldar con `mysqldump`; y luego pueden volver a cargarse los datos utilizando `mysql` o `mysqlimport` para volver a crear y llenar las tablas.
- Si se está empleando replicación, consulte Sección 6.6, “Aumentar la versión de la replicación” para información sobre la actualización de la configuración de replicación.
- Si está instalada una distribución MySQL-Max, la cual incluye un servidor llamado `mysqld-max`, y luego se actualiza a una versión no Max de MySQL, `mysqld_safe` continuará intentando ejecutar el antiguo servidor `mysqld-max`. En ese caso se debe remover manualmente el antiguo servidor `mysqld-max` a fin de asegurarse que `mysqld_safe` ejecute el nuevo servidor `mysqld`.

Los ficheros de formato y datos pueden moverse entre diferentes versiones pertenecientes a la misma arquitectura en la medida que correspondan a la misma serie de MySQL. La serie actualmente en producción es la 5.0. Si se cambia el conjunto de caracteres al ejecutar MySQL, se debe emplear `myisamchk -r -q --set-character-set=` *`charset`* en todas las tablas `MyISAM`. De otro modo, los índices podrían estar incorrectamente ordenados, porque al cambiar el conjunto de caracteres también cambia la forma de ordenarlos.

Si se desea tomar precauciones al utilizar una nueva versión, siempre se puede renombrar el antiguo `mysqld` antes de instalar uno nuevo. Por ejemplo, si se está empleado MySQL 4.1.13 y se desea actualizar a la 5.0.10, se debe renombrar el servidor actual de `mysqld` a `mysqld-4.1.13`. Si el nuevo `mysqld` hace algo inesperado, simplemente se lo detiene y se reinicia con el viejo `mysqld`.

Si luego de una actualización se experimentan problemas con programas cliente recompilados, tal como `Commands out of sync` o volcados de núcleo inesperados, probablemente al compilarlos se hayan empleado ficheros de cabecera o bibliotecas antiguos. En tal caso se debería chequear la fecha de los ficheros `mysql.h` y `libmysqlclient.a` para verificar que pertenecen a la nueva distribución de MySQL. Si no es así, habrá que recompilar los programas con los nuevos ficheros de cabecera y bibliotecas.

Si ocurriesen problemas como que el nuevo servidor `mysqld` no se iniciara o que no fuera posible conectarse sin usar una contraseña, hay que cerciorarse de que no exista un fichero `my.cnf` perteneciente a la instalación anterior. Se puede verificar con la opción `--print-defaults` (por ejemplo, `mysqld --print-defaults`). Si éste imprimiera algo más que el nombre del programa, significa que se tiene un fichero `my.cnf` aún activo, que está afectando la operación del cliente o del servidor.

Es una buena idea recompilar y reinstalar el módulo Perl `DBD::mysql` cada vez que se instale un nuevo release de MySQL. Lo mismo se aplica a otras interfaces con MySQL, como la extensión `mysql` de PHP y el módulo `MySQLdb` de Python.

2.10.1. Aumentar la versión de 4.1 a 5.0

Nota: es una buena práctica hacer copia de respaldo de los datos antes de instalar una nueva versión del software. Si bien MySQL ha puesto lo mejor de sí para asegurar un alto nivel de calidad, se deberían proteger los datos mediante una copia de respaldo.

En general, al actualizar de MySQL 4.1 a 5.0, se debería hacer lo siguiente:

- Verificar la lista de cambios que se encuentra más adelante en esta sección, para ver si cualquiera de ellos podría afectar las aplicaciones en uso. En especial, aquellos marcados como **Cambio incompatible**; estos resultan en incompatibilidades con versiones anteriores de MySQL, y podrían requerir atención *antes de actualizar*.

- Se debe leer el historial de cambios de MySQL 5.0 para ver qué características significativas nuevas se pueden utilizar. Consulte Sección C.1, “Cambios en la entrega 5.0.x (Desarrollo)”.

- Si se está ejecutando el Servidor MySQL para Windows, consulte Sección 2.3.15, “Aumentar la versión de MySQL en Windows”. Hay que tener en cuenta también que dos de los servidores MySQL para Windows cambiaron su nombre. Consulte Sección 2.3.9, “Seleccionar un tipo de servidor MySQL”.

- MySQL 5.0 incorporó soporte para procedimientos almacenados. Esto requiere que la tabla `proc` se encuentre en la base de datos `mysql`. Para crear este fichero, se debe ejecutar el script `mysql_fix_privilege_tables` tal como se describe en Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

- MySQL 5.0 también incorporó soporte para vistas. Esto requiere columnas adicionales de privilegios en las tablas `user` y `db` en la base de datos `mysql`. Para crear estas columnas, se debe ejecutar el script `mysql_fix_privilege_tables` como se describe en Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

- Si se está usando replicación, consulte Sección 6.6, “Aumentar la versión de la replicación” para información sobre cómo actualizar la configuración de replicación.

Entre MySQL 4.1 y 5.0 se introdujeron varios cambios notorios de comportamiento, para hacer a MySQL más compatible con el estándar SQL. Estos cambios pueden afectar a las aplicaciones en uso.

La siguiente lista describe los cambios que pueden afectar a las aplicaciones, a los que se debería prestar atención cuando se actualice a la versión 5.0.

Server Changes:

- **Cambio incompatible**: Cambió el orden de indexación para los espacios sobrantes en columnas `TEXT` en tablas `InnoDB` y `MyISAM`. A partir de MySQL 5.0.3, los índices son comparados incluyendo los espacios hasta el final (exactamente como MySQL ordena los campos `CHAR`, `VARCHAR` y `TEXT`). Si se tiene un índice sobre una columna `TEXT`, se debería ejecutar `CHECK TABLE` sobre ella. Si la verificación informa de errores, habrá que reconstruir los índices: volcar (dump) y volver a generar la tabla si es `InnoDB`, o bien ejecutar `OPTIMIZE TABLE` o `REPAIR TABLE` si es una tabla `MyISAM`.

- **Cambio incompatible**: Las tablas `MyISAM` e `InnoDB` que tengan columnas `DECIMAL` y se crearon con MySQL 5.0.3 a 5.0.5 aparecerán corruptas tras una actualización a MySQL 5.0.6. Debe hacerse un volcado de dichas tablas con `mysqldump` antes de actualizar, y volver a generarlas luego de la actualización. (La misma incompatibilidad ocurriría con estas tablas si fueran creadas en MySQL 5.0.6 y se hiciera un regreso a las versiones de MySQL entre 5.0.3 y 5.0.5).

- **Cambio incompatible**: a partir de MySQL 5.0.3, el servidor ya no carga por defecto las funciones definidas por el usuario, a menos que tengan por lo menos un símbolo auxiliar (por ejemplo, un símbolo `xxx_init` o `xxx_deinit`) además del símbolo principal de la función. Este comportamiento puede omitirse con la opción `--allow-suspicious-udfs`. Consulte Sección 27.2.3.6, “Precauciones de seguridad en funciones definidas por usuarios”.

- **Incompatible change**: El registro (log) de actualización fue eliminado en MySQL 5.0. Si anteriormente se lo tenía habilitado, se debería habilitar el registro binario (binary log) en su lugar.

- **Cambio incompatible:** Fue quitado el soporte para el motor de almacenamiento `ISAM`. Si se tenían tablas `ISAM`, se deberán convertir antes de actualizar. Por ejemplo, para convertir una tabla `ISAM` a fin de utilizar el motor de almacenamiento `MyISAM`, se emplea esta sentencia:

  ```
  ALTER TABLE tbl_name ENGINE = MyISAM;
  ```

  Debe emplearse una sentencia similar para cada tabla `ISAM` existente en las bases de datos.

- **Cambio incompatible**: Se ha quitado de MySQL 5.0 el soporte para opciones `RAID` en tablas `MyISAM`. Si se tienen tablas que utilicen estas opciones, se deberían convertir antes de actualizar. Una forma de hacerlo es realizar un volcado de la tabla con `mysqldump`, editar el fichero creado para eliminar todas las opciones `RAID` en las sentencias `CREATE TABLE`, y luego volver a generar las tablas a partir del fichero. Otra posibilidad es utilizar `CREATE TABLE new_tbl ... SELECT raid_tbl` para crear una nueva tabla a partir de la tabla `RAID`. Sin embargo, la parte `CREATE TABLE` de la sentencia debe contener suficiente información para regenerar los atributos de columna así como los índices, o los atributos de columna podrían perderse y los índices no aparecer en la nueva tabla. Consulte Sección 13.1.5, “Sintaxis de `CREATE TABLE`”.

  Los ficheros `.MYD` para tablas `RAID` en una determinada base de datos, son almacenados bajo el directorio de la base de datos, en subdirectorios que tienen nombres consistentes en dos dígitos hexadecimales en el rango de `00` a `ff`. Luego de convertir todas las tablas que emplean opciones `RAID`, estos subdirectorios seguirán existiendo pero pueden eliminarse. Hay que cerciorarse de que están vacíos, y borrarlos manualmente. (Si no están vacíos, es señal de que alguna tabla `RAID` ha quedado sin convertir).

- En MySQL 5.0.6, fue modificado el registro binario de procedimientos almacenados y triggers. Este cambio incide sobre la seguridad, la replicación, y la recuperación de datos, como se trata en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

SQL Changes:

- Las columnas `DECIMAL` ahora se almacenan en un formato más eficiente. Para convertir una tabla a fin de utilizar el nuevo tipo `DECIMAL`, se le debería aplicar una sentencia `ALTER TABLE`. Esta sentencia también cambiará las columnas `VARCHAR` de la tabla para que utilicen el nuevo tipo de columna `VARCHAR`. Para información acerca de posibles incompatibilidades con aplicaciones preexistentes, consulte Capítulo 23, *Matemáticas de precisión*.

- MySQL 5.0.3 y posteriores emplean matemática de precisión cuando realizan cálculos con valores `DECIMAL` (64 dígitos decimales) y para redondear números de valor exacto. Consulte Capítulo 23, *Matemáticas de precisión*.

- A partir de MySQL 5.0.3, los espacios sobrantes no se quitan de los valores almacenados en columnas `VARCHAR` y `VARBINARY`. Las longitudes máximas para columnas `VARCHAR` y `VARBINARY` en MySQL 5.0.3 son de 65.535 caracteres y 65.535 bytes, respectivamente.

  Nota: Si se crea una tabla con los nuevos tipos de columnas `VARCHAR` o `VARBINARY` en MySQL 5.0.3 o posterior, la tabla será inutilizable si se regresa a una versión anterior a la 5.0.3. Debe hacerse un volcado de la tabla antes de instalar la versión anterior y volver a generarla luego.

- A partir de MySQL 5.0.3, `BIT` es un tipo de dato separado, no un sinónimo para `TINYINT(1)`. Consulte Sección 11.1.1, “Panorámica de tipos numéricos”.

- MySQL 5.0.2 incorpora varios modos SQL que permiten un control más estricto sobre el rechazo de registros que tengan valores inválidos o perdidos. Consulte Sección 5.3.2, “El modo

SQL del servidor" y Sección 1.7.6.2, "Restricciones (constraints) sobre datos inválidos". Si se desea habilitar este control pero continuar usando la capacidad de MySQL para almacenar fechas incorrectas, como `'2004-02-31'`, se debería iniciar el servidor con la opción `--sql_mode=TRADITIONAL,ALLOW_INVALID_DATES`.

- A partir de MySQL 5.0.2, las palabras clave `SCHEMA` y `SCHEMAS` son aceptadas como sinónimos de `DATABASE` y `DATABASES` respectivamente. (Mientras que "schemata" es gramaticalmente correcta e incluso aparece en algunos nombres de tablas y bases de datos de sistema de MySQL 5.0, no puede ser utilizada como palabra clave en la entrada de sentencias).

- Las variables de usuario no son case sensitive en MySQL 5.0. En MySQL 4.1, `SET @x = 0; SET @X = 1; SELECT @x;` creaba dos variables y retornaba `0`. En MySQL 5.0, crea una sola variable y devuelve `1`.

Cambios en la API de C:

- El flag `reconnect` en la estructura `MYSQL` es establecido en 0 por `mysql_real_connect()`. Solamente experimentan un cambio aquellos programas cliente que no establecen explícitamente este flag a 0 o 1 luego de `mysql_real_connect()`. Tener habilitada la reconexión automática por defecto se considera muy riesgoso (debido a que los bloqueos de tablas, las tablas temporales, las variables de usuario y las variables de sesión se pierden luego de la reconexión).

2.10.2. Aumentar la versión de las tablas de privilegios

MySQL 5.0 introduce una serie de cambios en la estructura de las tablas de permisos (las tablas en la base de datos `mysql`) a fin de agregar nuevos privilegios y características. Las tablas de permisos también deben actualizrse cuando se efectúa la actualización a MySQL 5.0. En primer lugar debe hacerse una copia de respaldo de la base de datos `mysql`, y luego emplear el siguiente procedimiento.

En Unix o sistemas similares, se deben actualizar las tablas de permisos mediante la ejecución del script `mysql_fix_privilege_tables`:

```
shell> mysql_fix_privilege_tables
```

Se debe ejecutar este script mientras el servidor está en ejecución. Intenta conectarse como `root` al servidor en localhost. Si la cuenta `root` requiere una contraseña, la misma debe indicarse en la línea de comandos. Para MySQL 5.0 la contraseña se indica de este modo:

```
shell> mysql_fix_privilege_tables --password=root_password
```

El script `mysql_fix_privilege_tables` ejecuta todas las acciones necesarias para convertir las tablas de permisos hacia el formato 5.0. Durante su ejecución podrían verse algunas alertas del tipo `Duplicate column name`, pero deben ignorarse.

Después de ejecutar el script, el servidor debe ser detenido y reiniciado.

MySQL 5.0 para Windows incluye un script SQL llamado `mysql_fix_privilege_tables.sql` que puede ejecutarse empleando el cliente `mysql`. Si la instalación de MySQL está ubicada en `C:\Program Files\MySQL\MySQL Server 5.0`, el comando se vería así:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql -u root -p mysql
mysql> SOURCE C:/Program Files/MySQL/MySQL Server 5.0/scripts/mysql_fix_privilege_tables.sql
```

Si la instalación se localizara en cualquier otro directorio, habrá que ajustar la ruta apropiadamente.

El comando `mysql` solicitará la contraseña para el usuario `root`; hay que ingresarla.

Al igual que en el procedimiento para Unix, se podrían observar algunas alertas `Duplicate column name` a medida que `mysql` procesa las sentencias en el script `mysql_fix_privilege_tables.sql`, pero pueden ignorarse.

Luego de ejecutar el script, hay que detener y reiniciar el servidor.

Si se está actualizando a MySQL 5.0.1 o posterior, el procedimiento de actualización de las tablas de permisos que se acaba de describir agrega columnas relacionadas con las vistas, para los privilegios `CREATE VIEW` y `SHOW VIEW`. Estos privilegios existen a nivel global y a nivel de base de datos. Sus valores iniciales se establecen de esta forma:

- En MySQL 5.0.2 o posterior, `mysql_fix_privilege_tables` copia el valor de `Create_priv` de la tabla `user` dentro de las columnas `Create_view_priv` y `Show_view_priv`.
- En 5.0.1, los permisos relacionados con vistas no están habilitados para ninguna cuenta, por lo que no se puede utilizar `GRANT` para otorgar estos permisos a las cuentas que deban tenerlos. Para solventar esto, hay que conectarse al servidor como `root` y utilizar las siguientes sentencias para otorgarle estos privilegios a las cuentas `root` en forma manual, a través de `UPDATE`:

  ```
  mysql> UPDATE mysql.user SET Show_view_priv = 'Y', Create_view_priv = 'Y'
      -> WHERE User = 'root';
  mysql> FLUSH PRIVILEGES;
  ```

  Luego de esto, `root` se podrá usar `GRANT` para otorgar privilegios de vistas a otras cuentas. Nota: Se deben emplear las sentencias tal como se indican; `GRANT ALL` no tiene efecto en los niveles global y de base de datos, porque `GRANT` requiere que realmente se posean los privilegios que se otorgan.

2.10.3. Copiar bases de datos MySQL a otra máquina

Se pueden copiar los ficheros `.frm`, `.MYI`, y `.MYD` para tablas `MyISAM` entre diferentes arquitecturas siempre que soporten el mismo formato de punto flotante. (MySQL se encarga del problema de intercambio de bytes -byte-swapping-). Consulte Sección 14.1, “El motor de almacenamiento `MyISAM`”.

En casos en que se necesite transferir bases de datos entre diferentes arquitecturas, se puede emplear `mysqldump` para crear un fichero conteniendo sentencias SQL. Luego puede transferirse al otro ordenador y suministrarlo al cliente `mysql`.

`mysqldump --help` permite ver las opciones disponibles. Si se están transportando los datos hacia una versión de MySQL más nueva, se debería usar `mysqldump --opt`, para aprovechar las optimizaciones que resultan en un fichero de volcado más pequeño y más rápido de procesar.

La forma más fácil (aunque no la más rápida) de mover una base de datos entre dos ordenadores es ejecutar los siguientes comandos en el ordenador donde se encuentra la base de datos:

```
shell> mysqladmin -h 'otro_ordenador' create nombre_bd
shell> mysqldump --opt nombre_bd | mysql -h 'otro_ordenador' nombre_bd
```

Si se desea copiar una base de datos desde un ordenador remoto a través de una red lenta, se puede utilizar:

```
shell> mysqladmin create nombre_bd
shell> mysqldump -h 'otro_ordenador' --opt --compress nombre_bd | mysql nombre_bd
```

También se puede almacenar el resultado en un fichero, luego transferirlo al ordenador de destino, y cargar el fichero en la base de datos. Por ejemplo, para volcar una base de datos hacia un fichero en el ordenador de origen:

```
shell> mysqldump --quick nombre_bd | gzip > nombre_bd.contenidos.gz
```

(El fichero creado en este ejemplo está comprimido). Se debe transferir hacia el ordenador de destino el fichero con el contenido de la base de datos y ejecutar estos comandos allí:

```
shell> mysqladmin create nombre_bd
shell> gunzip < nombre_bd.contenidos.gz | mysql nombre_bd
```

También se puede emplear `mysqldump` y `mysqlimport` para transferir la base de datos. Para tablas grandes, esto será mucho más rápido que simplemente utilizar `mysqldump`. En los siguientes comandos, `DUMPDIR` representa la ruta completa al directorio donde se depositará la salida de `mysqldump`.

En primer lugar, crear el directorio para los ficheros de salida y volcar la base de datos:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR nombre_bd
```

Luego, transferir los ficheros desde el directorio `DUMPDIR` hacia el directorio correspondiente en el ordenador de destino, y allí cargar los ficheros en MySQL:

```
shell> mysqladmin create nombre_bd          #
crea la base de datos
shell> cat DUMPDIR/*.sql | mysql nombre_bd  #
crea las tablas en la base de datos
shell> mysqlimport nombre_bd DUMPDIR/*.txt  #
carga los datos en las tablas
```

Además, no hay que olvidar copiar la base de datos `mysql`, porque es la que contiene las tablas de permisos. Posiblemente, los comandos en el ordenador de destino se deban ejecutar como usuario `root` de MySQL hasta que la base de datos `mysql` esté en su lugar.

Luego de importar la base de datos `mysql` en el ordenador de destino, ejecutar `mysqladmin flush-privileges` para que el servidor vuelva a cargar la información de la tabla de permisos.

2.11. Bajar la versión de MySQL

Esta sección describe los pasos a seguir si se está regresando a una versión previa de MySQL (downgrading), en el improbable caso de que la versión anterior funcione mejor que la nueva.

Si el downgrading se produce dentro de la misma serie de releases (por ejemplo, de 4.1.13 a 4.1.12) la regla general es que simplemente hay que instalar los nuevos binarios sobre los anteriores. No es necesario hacer nada con las bases de datos. Sin embargo, como siempre, es mejor hacer una copia de respaldo.

La siguiente lista enumera los pasos que se deberían seguir cada vez que se lleva a cabo un downgrading:

- Leer la sección de actualización de la versión desde la que se hará el downgrading, para asegurarse de que no tenga ninguna característica realmente necesaria. Sección 2.10, “Aumentar la versión de MySQL”.

- También debería leerse, si existe, la sección que explique el downgrading desde esa versión.

En la mayoría de los casos se pueden mover los ficheros de formato y de datos entre diferentes versiones de la misma arquitectura y en la medida que no cambie la serie de releases de MySQL, que actualmente es 5.0.

Si se regresa de una serie a otro, pueden surgir incompatibilidades en los formatos de almacenamiento de las tablas. En este caso, se utiliza `mysqldump` para obtener un volcado de las tablas antes de hacer el downgrading. Despues de hacerlo, se carga el fichero de volcado empleando `mysql` o `mysqlimport` para volver a crear las tablas. Consulte Sección 2.10.3, “Copiar bases de datos MySQL a otra máquina” para ver ejemplos.

Normalmente, el síntoma de que un cambio en el formato de las tablas era incompatible con el downgrading es que no se pueden abrir las tablas. En tal caso, utilizar el siguiente procedimiento:

1. Detener el antiguo servidor MySQL, hacia el que se está tratando de hacer el downgrading.
2. Reiniciar el nuevo servidor MySQL, desde el cual se está tratando de hacer el downgrading.
3. Volcar las tablas que aparecen inaccesibles para el antiguo servidor, empleando `mysqldump` para crear un fichero de volcado.
4. Detener el nuevo servidor MySQL y reiniciar el antiguo.
5. Cargar el fichero de volcado en el viejo servidor. Las tablas deberían ser accesibles.

2.11.1. Volver a la versión 4.1

Luego de hacer un downgrading desde MySQL 5.0, es posible encontrar la siguiente información en el fichero `mysql.err`:

```
Incorrect information in file: './mysql/user.frm'
```

En tal caso, hacer lo siguiente:

1. Iniciar MySQL 5.0.4 (o posterior)
2. Ejecutar `mysql_fix_privilege_tables`, el cual cambiará la tabla `mysql.user` a un formato que puedan entender tanto MySQL 4.1 como 5.0.
3. Detener el servidor MySQL.
4. Iniciar MySQL 4.1

Si el procedimiento anterior falla, el siguiente debería funcionar:

1. Iniciar MySQL 5.0.4 (o posterior).
2. Ejecutar `mysqldump --opt --add-drop-table mysql > /tmp/mysql.dump`.
3. Detener el servidor MySQL.
4. Iniciar MySQL 4.1 con la opción `--skip-grant`.
5. Ejecutar `mysql mysql < /tmp/mysql.dump`.
6. Ejecutar `mysqladmin flush-privileges`.

2.12. Notas específicas sobre sistemas operativos

2.12.1. Notas sobre Linux

Esta sección se ocupa de problemas que han ocurrido bajo Linux. Las primeras subsecciones describen dificultades relacionadas con el sistema operativo en general, problemas que pueden ocurrir al emplear distribuciones binarias o de código fuente, y problemas posteriores a la instalación. Las restantes subsecciones se ocupan de problemas que se dan en plataformas Linux específicas.

Nótese que la mayoría de estos problemas ocurren en versiones antiguas de Linux. Si se está ejecutando una versión reciente, es probable que no se vea ninguno de ellos.

2.12.1.1. Notas sobre el sistema operativo Linux

MySQL requiere por lo menos Linux Versión 2.0.

Advertencia: Se detectaron algunos problemas extraños con Linux 2.2.14 y MySQL sobre sistemas SMP (multiprocesamiento simétrico). También se tiene información de algunos usuarios que encontraron serios problemas de estabilidad al ejecutar MySQL con el kernel 2.2.14. Si se está empleando este kernel, se debería actualizar a la versión 2.2.19 (o posterior) o 2.4. Si se cuenta con un ordenador con múltiples CPUs, habría que considerar seriamente el uso del kernel 2.4, ya que representa un notable incremento de velocidad. También es más estable.

Al emplear LinuxThreads, se debería ver un mínimo de tres procesos `mysqld` en ejecución. De hecho, son hebras (threads). Uno corresponde al gestor de LinuxThreads, otro es para manejar conexiones, y uno más para manejar advertencias y señales.

2.12.1.2. Notas sobre la distribución binaria de Linux

El binario para Linux-Intel y los releases RPM de MySQL están configurados para funcionar a la mayor velocidad posible. Quienes desarrollan MySQL siempre tratan de emplear el compilador estable más rápido disponible.

El release binario se enlaza con `-static`, lo cual significa que normalmente no habrá que preocuparse por la versión de las bibliotecas del sistema que se tenga. Un programa enlazado con `-static` es ligeramente más rápido (3-5%). Sin embargo, un problema con los programas enlazados estáticamente es que no se pueden emplear funciones definidas por el usuario (FDU o UDF, por sus siglas en inglés). Si se van a escribir o emplear FDUs (esto es sólo para programadores de C o C++), habría que recompilar MySQL empleando enlazado dinámico.

Un problema con las distribuciones binarias es que en sistemas Linux antiguos que utilizan `libc` (tal como Red Hat 4.x o Slackware), se tendrán algunos problemas (no fatales) con la resolución del nombre de host. Si el sistema emplea `libc` en lugar de `glibc2`, probablemente se encontrarán algunas dificultades con la resolución de nombres de host y `getpwnam()`. Esto ocurre porque `glibc` (desafortunadamente) depende de algunas bibliotecas externas para implementar la resolución de nombres de host y `getpwent()`, incluso cuando se compila con `-static`. Estos problemas se manifiestan de dos maneras:

- Ver el siguiente mensaje de error al ejecutar `mysql_install_db`:

```
Sorry, the host 'xxxx' could not be looked up
```

  Esto puede solventarse mediante la ejecución de `mysql_install_db --force`, lo cual evita que se ejecute la prueba de `resolveip` en `mysql_install_db`. La contraparte de esto es que no se pueden utilizar nombres de host en las tablas de permisos: excepto `localhost`, se deben usar números d eIP

en su lugar. Si se está utilizando una versión de MySQL que no soporta la opción `--force`, se debe quitar manualmente la prueba `resolveip` de `mysql_install` empleando un editor de textos.

- También se puede hallar el siguiente error cuando se ejecuta `mysqld` con la opción `--user`:

```
getpwnam: No such file or directory
```

  Para resolver esto, iniciar `mysqld` mediante el comando `su` en lugar de especificar la opción `--user`. Esto provoca que el sistema cambie el ID de usuario del proceso `mysqld`, así no debe hacerlo `mysqld`.

Otra solución, que resuelve ambos problemas, es no usar una distribución binaria. En su lugar se debe obtener una distribución de código fuente de MySQL (en formatos RPM o `tar.gz`).

En algunas versiones 2.2 de Linux puede producirse el error `Resource temporarily unavailable` cuando los clientes establezcan un número elevado de nuevas conexiones TCP/IP al servidor `mysqld`. La razón es que Linux tiene un retraso entre el momento en que se cierra un socket TCP/IP y el momento en que el sistema lo libera realmente. Sólo hay capacidad para una cantidad limitada de conexiones, por eso se produce el error de recursos no disponibles (resource-unavailable) si los clientes intentan realizar demasiadas conexiones TCP/IP en un período corto de tiempo. Por ejemplo, este error puede verse cuando se ejecuta la prueba de rendimiento `test-connect` sobre TCP/IP.

Se ha indagado sobre este problema en diferentes listas de correo de Linux pero nunca se ha obtenido una solución convincente. El único “fix” conocido es para clientes que emplean conexiones persistentes, o, si se están ejecutando el servidor y los clientes en el mismo ordenador, emplear conexiones a través de ficheros socket de Unix en lugar de conexiones TCP/IP.

2.12.1.3. Notas sobre la distribución de código fuente para Linux

Las siguientes notas relativas a `glibc` son aplicables únicamente en el caso que se desee compilar el código de MySQL. Si se está ejecutando Linux en un ordenador x86, en la mayoría de los casos será mucho mejor utilizar el binario. Quienes hacen MySQL enlazan los binarios utilizando la mejor y más actual versión de `glibc` que tienen disponible y con las opciones de compilación más apropiadas a fin de hacerlo apto para un servidor de uso intensivo o high-load. Para un usuario típico, o incluso en configuraciones con muchas conexiones concurrentes o tablas excediendo el límite de 2Gb, el binario provisto por MySQL AB es en la mayoría de los casos la mejor elección. Luego de leer el siguiente texto, si aún persiste la duda, hay que probar el binario para determinar si cubre las necesidades del usuario. Si no es suficiente, entonces puede intentarse una compilación propia. En tal caso MySQL AB apreciará que se le comuniquen los detalles para crear mejores binarios en el futuro.

MySQL emplea LinuxThreads en Linux. Si se utiliza una versión antigua de Linux que no tiene `glibc2`, se deberá instalar LinuxThreads antes de compilar MySQL. LinuxThreads puede descargarse de http://dev.mysql.com/downloads/os-linux.html.

Notar que las versiones de `glibc` hasta la 2.1.1 inclusive tienen un error fatal en el manejo de `pthread_mutex_timedwait()`, el cual es utilizado al emitir sentencias `INSERT DELAYED`. Se recomienda que no se use `INSERT DELAYED` sin antes haber actualizado `glibc`.

El kernel de Linux y la biblioteca LinuxThreads pueden manejar por defecto un máximo de 1024 subprocesos. Si se planea gestionar más de 1000 conexiones simultáneas, se necesitarán algunos cambios en LinuxThreads:

- Incrementar `PTHREAD_THREADS_MAX` en el fichero `sysdeps/unix/sysv/linux/bits/local_lim.h` a un valor de 4096 y reducir `STACK_SIZE` en el fichero `linuxthreads/internals.h` a un valor de 256KB. Las rutas son relativas a la raíz de `glibc`. (Tener en cuenta que MySQL no es estable con 600 a 1000 conexiones si `STACK_SIZE` se deja en el valor por defecto de 2MB).

- Recompilar LinuxThreads para producir una nueva biblioteca `libpthread.a`, y volver a enlazarla con MySQL.

Hay otro problema que afecta enormemente el rendimiento de MySQL, especialmente en sistemas SMP (multiprocesamiento simétrico, por sus siglas en inglés). La implementación de mutex en LinuxThreads en `glibc` 2.1 es muy pobre para programas con muchos subprocesos que mantengan el mutex sólo por un corto tiempo. Esto produce un resultado paradójico: si se enlaza MySQL con LinuxThreads sin modificar, el quitar procesadores de un sistema SMP realmente mejora el rendimiento de MySQL en muchos casos. MySQL AB ha creado un parche disponible para `glibc` 2.1.3 para corregir este comportamiento (http://www.mysql.com/Downloads/Linux/linuxthreads-2.1-patch).

Con `glibc` 2.2.2, MySQL utiliza el mutex adaptable, lo cual es mucho mejor inclusive que `glibc` 2.1.3 con parche y todo. Hay que estar al tanto, sin embargo, de que bajo ciertas condiciones, el código de exclusión mutua (mutex) actual emplea con demasiada frecuencia los spinlocks (bucles de programa que ciclan constantemente esperando por una condición), lo cual repercute en las prestaciones de MySQL. La probabilidad de que esto ocurra se puede reducir dando al proceso `mysqld` la máxima prioridad. MySQL AB también ha logrado corregir el comportamiento relativo a los spinlocks con un parche, que puede descargarse desde http://www.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch. Este combina la corrección del uso excesivo de spinlocks, máximo número de procesos, y la capacidad de la pila, todo en uno. Se lo deberá aplicar en el directorio `linuxthreads` con `patch -p0 </tmp/linuxthreads-2.2.2.patch`. Es de esperar que sea incluido de alguna forma en futuras versiones de `glibc` 2.2. De cualquier modo, si enlaza con `glibc` 2.2.2, aún será necesario corregir `STACK_SIZE` y `PTHREAD_THREADS_MAX`. Es de esperar que los valores por defecto de éstos sean llevados en el futuro a un número más aceptable para configuraciones MySQL de altas prestaciones, de modo que los comandos necesarios para recompilarlo se reduzcan a `./configure; make; make install`.

Se recomienda que se usen estos parches para producir una versión estática, especial, de `libpthread.a` y emplearla solamente para enlazado dinámico con MySQL. Se sabe que los mencionados parches son seguros para MySQL y mejoran significativamente su rendimiento, pero no se puede decir nada acerca de sus efectos sobre otras aplicaciones. Enlazar otras aplicaciones que requieran LinuxThreads con la versión estática parcheada o hacer una versión mixta e instalarla en el sistema, es por cuenta del usuario y a su riesgo.

Si se experimenta cualquier problema extraño durante la instalación de MySQL, o algunas utilidades comunes se congelan, es muy probable que esté relacionado con las bibliotecas o el compilador. Si este es el caso, utilizar el binario de MySQL AB resolverá el problema.

Si el usuario compila sus propios programas cliente, es posible que vea los siguientes errores en tiempo de ejecución:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

Este problema puede evitarse de alguna de estas maneras:

- Enlazando clientes con el flag `-Wl,r/full/path/to/libmysqlclient.so` en lugar de `-Lpath`).
- Copiando `libmysqclient.so` a la carpeta `/usr/lib`.
- Agregando la ruta del directorio donde se ubica `libmysqlclient.so` a la variable de entorno `LD_RUN_PATH` antes de ejecutar el cliente.

Si se emplea el compilador Fujitsu (`fcc/FCC`), se puede tener algún problema al compilar MySQL porque los ficheros de cabecera de Linux están muy orientados a `gcc`. La siguiente línea de `configure` debería funcionar con `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
```

2.12.1.4. Notas para después de la instalación en Linux

`mysql.server` es un fichero que puede hallarse en el directorio `support-files` bajo el directorio de instalación de MySQL o en un árbol de código fuente MySQL. Se instala como `/etc/init.d/mysql` para conseguir que MySQL inicie y se detenga automáticamente. Consulte Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente”.

Si MySQL no puede abrir suficiente ficheros o conexiones, es posible que Linux no se haya configurado para gestionar suficientes ficheros.

En Linux 2.2 y posteriores, se puede verificar la cantidad de manejadores de ficheros asignados de esta forma:

```
shell> cat /proc/sys/fs/file-max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

Si se poseen más de 16 MB de ram, se debería agregar a los scripts de inicio algo como lo siguiente (por ejemplo, `mysql.server` en SuSE Linux):

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

Los comandos `echo` también pueden ejecutarse como `root`,desde la línea de comandos, pero los valores establecidos se perderán la próxima vez que se reinicie el ordenador.

De manera alternativa, estos parámetros pueden configurarse al inicio usando la herramienta `sysctl`, la cual es usada por muchas distribuciones de Linux (incluyendo SuSE Linux 8.0 y posteriores). Colocar los siguientes valores en un fichero llamado `/etc/sysctl.conf`:

```
# Incrementa algunos valores para MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

También se debería agregar lo siguiente a `/etc/my.cnf`:

```
[mysqld_safe]
open-files-limit=8192
```

Esto elevará a 8192 el límite del servidor para el número total de conexiones y ficheros abiertos.

La constante `STACK_SIZE` de Linuxthreads controla el espacio de la pila de subprocesos en el espacio de direcciones. Necesita ser lo suficientemente grande como para brindar bastante lugar para cada pila individual de subprocesos, pero lo suficientemente pequeña para mantener la pila de algunos subprocesos

ejecutándose fuera de los datos globales de `mysqld`. Desafortunadamente, la experiencia demostró que la implementación Linux de `mmap()` desasigna una región previamente asignada (mapped), si se solicita asignar (map) una dirección actualmente en uso, poniendo a cero los datos de toda la página en lugar de retornar un error. De modo que la seguridad de `mysqld` o cualquier otra aplicación hebrada depende de la "caballerosidad" del código que crea los subprocesos. El usuario debe tomar medidas para cerciorarse que el número de procesos en ejecución en cualquier momento dado es lo suficientemente bajo como para que las pilas de procesos se mantengan lejos del montón (heap) global. Con `mysqld` esto debería hacerse, estableciendo un valor razonable para la variable `max_connections`.

Si el usuario compila por sí mismo a MySQL, puede aplicar un parche a LinuxThreads para un mejor uso de la pila. Consulte Sección 2.12.1.3, “Notas sobre la distribución de código fuente para Linux”. Si no se desea aplicar un parche a LinuxThreads, se deberá establecer `max_connections` a un valor no mayor de 500, o incluso menos si se tienen un gran buffer de claves, grandes tablas de montón (heap tables) o alguna otra cosa que obligue a `mysqld` a reservar gran cantidad de memoria,o si se está ejecutando un kernel 2.2 con un parche de 2Gb. Si se está empleando la versión binaria en RPM, se puede establecer en forma segura `max_connections` a un valor de 1500, asumiendo que no hay grandes buffers de claves, o grandes tablas de montón (heap tables) con muchos datos. Mientras más se reduzca `STACK_SIZE` en LinuxThreads, más subprocesos podrán crearse sin problemas. Se recomiendan valores ente 128KB y 256KB.

Si se utilizan muchas conexiones simultáneas, puede sufrirse una “característica” del kernel 2.2, que intenta prevenir ataques de bomba de bifurcación (fork bomb) penalizando los procesos que bifurcan o clonan un proceso hijo. Esto provoca que MySQL no escale bien a medida que se incrementa el número de clientes simultáneos. En sistemas con una única CPU, esto se manifiesta a través de lentitud en la creación de procesos; puede llevar largo tiempo conectarse a MySQL (hasta un minuto) y puede llevar lo mismo para detenerlo. En sistemas con múltiples CPUs, se observó una caída gradual en la velocidad de las consultas a medida que aumenta el número de clientes. En la búsqueda de una solución, se recibió un parche para el kernel de parte de un usuario que lo necesitó para su sitio web. Este parche puede descargarse de http://www.mysql.com/Downloads/Patches/linux-fork.patch. MySQL AB probó ampliamente este parche tanto en sistemas en desarrollo como en producción. Funcionó incrementando notablemente el rendimiento de MySQL sin causas problemas, por lo tanto se lo recomienda a los usuario que aún ejecuten servidores de altas prestaciones en kernels 2.2.

Este problema se resolvió en el kernel 2.4, de modo que si no se está satisfecho con el rendimiento actual del sistema, en lugar de aplicar un parche al kernel 2.2, podría ser más sencillo actualizarlo a 2.4. En sistemas SMP (multiprocesamiento simétrico), la actualización también favorecerá el desempeño de SMP además de corregir el error.

Al probar MySQL con un kernel 2.4 en un ordenador de dos procesadores, se halló que MySQL escala *mucho* mejor. Hasta 1,000 clientes prácticamente no se producen retrasos en el rendimiento de las consultas, y el factor de escalado de MySQL (calculado como el máximo rendimiento respecto al rendimiento de un cliente) fue 180%. Se observaron resultados similares en un ordenador de cuatro procesadores: hasta 1,000 clientes prácticamente no se producen retrasos en el rendimiento de las consultas, y el factor de escalado de MySQL fue de 300%. Al basarse en estos resultados, definitivamente se recomienda que los servidores de alta prestación ejecutando un kernel 2.2 se actualicen a 2.4.

A fin de obtener el máximo rendimiento, es esencial ejecutar el proceso `mysqld` con la prioridad más alta posible en kernel 2.4. Esto puede lograrse agregando un comando `renice -20 $$` a `mysqld_safe`. En las pruebas sobre un ordenador de 4 procesadores, el aumento de la prioridad produjo un 60% de incremento en el rendimiento con 400 clientes.

En la actualidad, MySQL AB se halla recolectando información sobre el desempeño de MySQL sobre un kernel 2.4 en sistemas de cuatro y ocho procesadores. Si se tiene acceso a tales sistemas, y se han hecho algunas pruebas de rendimiento, por favor envíese un mensaje de correo electrónico a `<benchmarks@mysql.com>` con los resultados. Estos serán revisados para su inclusión en este manual.

Si con `ps` se advierte un proceso `mysqld` muerto, generalmente significa un error en MySQL o una tabla corrupta. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”.

Para que se genere un volcado de núcleo en Linux cuando `mysqld` finalice imprevistamente con una señal `SIGSEGV`, debe iniciarse `mysqld` con la opción `--core-file`. También es probable que se necesite aumentar el espacio para el fichero de volcado de núcleo mediante el agregado de `ulimit -c 1000000` a `mysqld_safe` o iniciando `mysqld_safe` con `--core-file-size=1000000`. Consulte Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.

2.12.1.5. Notas sobre Linux x86

MySQL necesita la versión 5.4.12 o posterior de `libc`. Se sabe que trabaja correctamente con `libc` 5.4.46. `glibc` versión 2.0.6 y posterior también debería funcionar. Han ocurrido algunos problemas con los RPMs de `glibc` para Red Hat, de modo que si sucede, se deberá buscar cualquier actualización disponible. Los RPMs de las versiones `glibc` 2.0.7-19 y 2.0.7-29 funcionan adecuadamente.

Si se está empleando Red Hat 8.0 o una nueva biblioteca `glibc` 2.2.x, se puede ver una finalización imprevista de `mysqld` en `gethostbyaddr()`. Esto sucede porque la nueva biblioteca `glibc` requiere un tamaño de pila mayor a 128Kb para esta llamada. Para solucionar el problema, se debe iniciar `mysqld` con la opción `--thread-stack=192K`. (Use `-O thread_stack=192K` si está utilizando una versión de MySQL anterior a MySQL 4). Este tamaño de pila es el predeterminado en MySQL 4.0.10 y posteriores, de modo que el problema mencionado no existirá.

Si se está empleando `gcc` 3.0 o posterior para compilar MySQL, se deberá instalar la biblioteca `libstdc++v3` antes de compilar MySQL; si no se hace de esta forma, se obtendrá un error relativo a un símbolo inexistente `__cxa_pure_virtual` durante el enlazado.

En algunas distribuciones de Linux antiguas, `configure` puede producir un error como este:

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Simplemente hay que hacer lo que el mensaje dice (Error de sintaxis en sched.h. Cambie _P a __P en el fichero /usr/include/sched.h) Agregar un carácter de subrayado adicional al nombre de macro `_P` que tiene solamente uno, y reintentar.

Pueden aparecer algunas advertencias durante la compilación. Las que se listan a continuación, pueden ignorarse:

```
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'
```

Si `mysqld` realiza siempre un volcado de núcleo al iniciarse, el problema puede estar en una versión antigua de `/lib/libc.a`. Debe intentarse renombrando el fichero, luego borrar `sql/mysqld` y ejecutar nuevamente el comando `make install`. Luego reintentar. Este problema se informó en algunas instalaciones de Slackware.

Si se obtiene el siguiente error al enlazar `mysqld`, significa que la biblioteca `libg++.a` no se instaló correctamente:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

Se puede evitar el uso de `libg++.a` ejecutando `configure` de esta manera:

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

2.12.1.6. Notas sobre Linux SPARC

En algunas implementaciones, `readdir_r()` no funciona correctamente. El síntoma es que la sentencia `SHOW DATABASES` devuelve una respuesta vacía. Esto puede solucionarse eliminando `HAVE_READDIR_R` del fichero `config.h` después de configurar y antes de compilar.

2.12.1.7. Notas sobre Linux Alpha

Se hicieron pruebas de rendimiento y funcionamiento con MySQL 5.0 sobre Alpha, y parece funcionar correctamente.

Actualmente, los paquetes binarios de MysQL se crean sobre SuSE Linux 7.0 para AXP, kernel 2.4.4-SMP, Compiladores Compaq C (V6.2-505) y Compaq C++ (V6.3-006) sobre un ordenador Compaq DS20 con procesador Alpha EV6.

Los compiladores mencionados pueden descargarse de http://www.support.compaq.com/alpha-tools/. Usando éstos en lugar de `gcc`, se ha obtenido una mejora del 9% al 14% en el rendimiento de MySQL.

Para MySQL 5.0 sobre Alpha, se utiliza el flag `-arch generic` en las opciones del compilador, lo cual garantiza que el binario se ejecute en todos los procesadores Alpha. También se compila estáticamente para evitar problemas con bibliotecas. El comando `configure` se ve así:

```
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
```

Si se desea emplear `egcs`, la siguiente linea en `configure` ha funcionado bien:

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --disable-shared
```

Algunos problemas conocidos al ejecutar MySQL en Linux-Alpha:

- La depuración de aplicaciones multihilo como MySQL no funciona en `gdb 4.18`. Debe emplearse `gdb` 5.1 en su lugar.
- Si se intenta enlazar estáticamente `mysqld` cuando se utiliza `gcc`, la imagen resultante realiza un volcado de núcleo al iniciarse. Esto significa que *no se debe* emplear `--with-mysqld-ldflags=-all-static` con `gcc`.

2.12.1.8. Notas sobre Linux PowerPC

MySQL debería funcionar en MkLinux con el paquete más nuevo de `glibc` (se lo probó con `glibc` 2.0.7).

2.12.1.9. Notas sobre Linux MIPS

Para lograr que MySQL funcione en Qube2 (Linux Mips), se necesita la verseón más nueva de las bibliotecas `glibc`. `glibc-2.0.7-29C2` funciona correctamente. También es necesario emplear el compilador de C++ `egcs` (`egcs` 1.0.2-9, `gcc` 2.95.2 o posterior).

2.12.1.10. Notas sobre Linux IA-64

Para lograr que MySQL se compile en Linux IA-64, se los desarrolladores de MySQL utilizaron el siguiente comando `configure` para compilar con `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
```

En IA-64, los clientes binarios de MySQL utilizan bibliotecas compartidas. Esto significa que si se instala la distribución binaria provista por MySQL AB en otra ubicación que no sea `/usr/local/mysql`, se deberá agregar la ruta donde está instalado `libmysqlclient.so`, ya sea en el fichero `/etc/ld.so.conf` o en la variable de entorno `LD_LIBRARY_PATH`.

Consulte Sección A.3.1, “Problemas al enlazar a la biblioteca de clientes MySQL”.

2.12.2. Notas sobre Mac OS X

En Mac OS X, `tar` no puede manejar nombres largos de fichero. Si se necesita descomprimir una distribución `.tar.gz`, se deberá emplear `gnutar`.

2.12.2.1. Mac OS X 10.x (Darwin)

MySQL debería funcionar sin mayores problemas en Mac OS X 10.x (Darwin).

Los problemas conocidos son:

- Los valores de tiempo de conexión (`wait_timeout`, `interactive_timeout` y `net_read_timeout`) no se respetan.

  Probablemente esto sea indicio de un problema de manejo en la biblioteca de subprocesos, donde la señal no interrumpe una lectura pendiente. Es de esperar que una futura actualización de la biblioteca de subprocesos lo corrija.

El binario provisto por MySQL para Mac OS X está compilado sobre Darwin 6.3 con la siguiente línea en `configure`:

```
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
```

Consulte Sección 2.5, “Instalar MySQL en Mac OS X”.

2.12.2.2. Servidor Mac OS X 1.2 (Rhapsody)

En versiones actuales de Mac OS X Server no se requieren cambios al sistema operativo antes de compilar MySQL. La compilación para la plataforma Server es lo mismo que para la versión cliente de MySQL.

En versiones antiguas (Mac OS X Server 1.2, también conocido como Rhapsody), se debe instalar un paquete pthread antes de intentar configurar MySQL.

Consulte Sección 2.5, “Instalar MySQL en Mac OS X”.

2.12.3. Notas sobre Solaris

En Solaris pueden experimentarse problemas aún antes de lograr descomprimir la distribución de MySQL, ya que `tar` no puede manejar nombres de fichero largos en Solaris. Esto significa que pueden verse errores cuando se intente expandir MySQL.

Si esto ocurre, habrá que emplear el GNU `tar` (`gtar`) para expandir la distribución. Se puede hallar una copia precompilada para Solaris en http://dev.mysql.com/downloads/os-solaris.html.

Los procesos nativos de Sun solamente funcionan en Solaris 2.5 y posteriores. Para la versión 2.4 y anteriores, MySQL utilizará MIT-pthreads automáticamente. Consulte Sección 2.8.5, “Notas sobre MIT-pthreads”.

Si se obtienen el siguiente error en `configure`, significa que algo falló en la instalación del compilador:

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

En este caso se debería actualizar a una versión más reciente del compilador. También podría resolverse este problema insertando la siguiente línea en el fichero `config.cache`:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

Si se esta empleando Solaris en una SPARC, el compilador recomendado es `gcc` 2.95.2 o 3.2. Se lo puede descargar de http://gcc.gnu.org/. Hay que notar que `egcs` 1.1.1 y `gcc` 2.8.1 no funcionan confiablemente en SPARC.

La línea recomendada en `configure` al utilizar `gcc` 2.95.2 es:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
    --enable-assembler
```

Si se tiene un sistema UltraSPARC, se puede mejorar el rendimiento en un 4% agregando `-mcpu=v8 -Wa,-xarch=v8plusa` a las variables de entorno `CFLAGS` y `CXXFLAGS`.

Si se tiene el compilador Forte 5.0 (o posterior) de Sun, se puede ejecutar `configure` de esta manera:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

Para crear un binario de 64 bits con el compilador Forte de Sun, deben utilizarse las siguientes opciones de configuración:

```
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
```

Para crear un binario de 64 bits para Solarios utilizando `gcc`, debe agregarse `-m64` a `CFLAGS` y `CXXFLAGS` y quitar `--enable-assembler` de la linea de `configure`.

En las pruebas de rendimiento MySQL, se obtuvo un incremento del 4% en la velocidad en una UltraSPARC cuando se utilizó Forte 5.0 en modo de 32 bits, comparado con `gcc` 3.2 con el flag `-mcpu`.

Si se crea un binario `mysqld` de 64 bits, es un 4% más lento que el binario de 32 bits, pero puede manejar más subprocesos y memoria.

Al utilizar Solaris 10 para x86_64, cualquier sistema de ficheros (filesystem) donde se deseen almacenar ficheros InnoDB debería ser montado con la opción `forcedirectio`. (Por defecto, el montaje se realiza sin esta opción) Si no se hace de este modo, el rendimiento caerá significativamente al usar el motor de almacenamiento InnoDB en dicha plataforma.

Si se tienen problemas con `fdatasync` o `sched_yield`, se podrán solucionar agregando `LIBS=-lrt` en la línea de `configure`.

Para compiladores anteriores a WorkShop 5.3, se podría tener que editar el script `configure`, cambiando esta línea:

```
#if !defined(__STDC__) || __STDC__ != 1
```

Poniendo esta en su lugar:

```
#if !defined(__STDC__)
```

Si se inicia `__STDC__` con la opción `-Xc`, el compilador de Sun no podrá compilar con el fichero de cabecera `pthread.h` de Solaris. Esto es un error de Sun (en el compilador o en el fichero).

Si `mysqld` emite el siguiente mensaje de error cuando se lo ejecuta, es porque se lo ha compilado con el compilador de Sun sin habilitar la opción de multihilo `-mt`:

```
libc internal error: _rmutex_unlock: rmutex not held
```

Agregar `-mt` a `CFLAGS` y `CXXFLAGS` y recompilar.

Si se está utilizando la versión SFW de `gcc` (que viene con Solaris 8), se debe agregar `/opt/sfw/lib` a la variable de entorno `LD_LIBRARY_PATH` antes de ejecutar `configure`.

Si se está empleando el `gcc` disponible en `sunfreeware.com`, pueden tenerse muchos problemas. Para evitarlo, se debería recompilar `gcc` y GNU `binutils` en el ordenador donde se los ejecutará.

Si se obtiene el siguiente mensaje de error al compilar MySQL con `gcc`, significa que `gcc` no está configurado para la versión en uso de 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'
```

Lo más apropiado para hacer en este caso es conseguir la versión más nueva de `gcc` y compilarlo con el `gcc` que se tiene. Al menos en Solaris 2.5, casi todas las versiones binarias de `gcc` tienen ficheros

de cabecera antiguos e inutilizables que hacen caer a todos los programas que usan subprocesos y posiblemente también a otros programas.

Solaris no provee versiones estáticas de todas las bibliotecas del sistema (`libpthreads` y `libdl`), de modo que no se puede compilar MySQL con `--static`. Si se intenta hacer tal cosa, se obtendrá uno de los siguientes errores:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

Si se enlaza con los programas cliente MySQL del usuario, se puede ver el siguiente error en tiempo de ejecución:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

Este problema puede evitarse por medio de alguno de estos métodos:

- Enlazar clientes con el flag `-Wl,r/full/path/to/libmysqlclient.so` en lugar de `-Lpath`).
- Copiar `libmysqclient.so` al directorio `/usr/lib`.
- Antes de ejecutar el cliente, agregar a la variable de entorno `LD_RUN_PATH` la ruta del directorio donde se localiza `libmysqlclient.so`.

Si ocurren problemas con `configure` al intentar enlazar con `-lz` cuando no se tiene instalado `zlib`, hay dos opciones:

- Si se desea tener la capacidad de usar el protocolo de comunicación comprimido, se deberá conseguir `zlib` desde `ftp.gnu.org` e instalarlo.
- Ejecutar `configure` con la opción `--with-named-z-libs=no` cuando se compile MySQL.

Si se está utilizando `gcc` y se tienen problemas con la carga de funciones definidas por el usuario (UDFs) en MySQL, hay que intentar agregar `-lgcc` a la línea donde se enlaza la UDF.

Si se desea que MySQL inicie automáticamente, se debe copiar `support-files/mysql.server` a `/etc/init.d` y crear un vínculo simbólico hacia él, llamado `/etc/rc3.d/S99mysql.server`.

Si demasiados procesos intentan conectarse muy rápidamente a `mysqld`, se verá este error en el log de MySQL:

```
Error in accept: Protocol error
```

Se puede intentar iniciar el servidor con la opción `--back_log=50` como una forma de solución. (Utilizar `-O back_log=50` en versiones anteriores a MySQL 4).

Solaris no soporta ficheros de núcleo para aplicaciones `setuid()`, de forma que no se logrará un fichero de núcleo a partir de `mysqld` si se está empleando la opción `--user`.

2.12.3.1. Notas sobre Solaris 2.7/2.8

Generalmente se puede utilizar un binario de Solaris 2.6 en Solaris 2.7 y 2.8. La mayoría de los problemas mencionados bajo Solaris 2.6 también se aplican a Solaris 2.7 y 2.8.

MySQL debería detectar automáticamente nuevas vesiones de Solaris y habilitar soluciones específicas para los siguientes problemas.

Solaris 2.7/2.8 tiene algunos errores en los ficheros de inclusión. Se obtiene el siguiente error al usar `gcc`:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

Si ocurre eso, puede solucionarse copiando `/usr/include/widec.h` a `.../lib/gcc-lib/os/gcc-version/include` y cambiando la línea 41:

```
#if     !defined(lint) && !defined(__lint)
```

Colocando esta:

```
#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

Como alternativa, puede editarse directamente el fichero `/usr/include/widec.h`. En cualquiera de las dos formas, se debe eliminar `config.cache` y ejecutar `configure` nuevamente.

Si se obtienen los siguientes errores al ejecutar `make`, es debido a que `configure` no detectó correctamente el fichero `curses.h` (probablemente a causa del error en `/usr/include/widec.h`):

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

La solución es hacer algo de lo siguiente:

- Configure con `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure`.
- Editar `/usr/include/widec.h` como se indicó anteriormente y ejecutar de nuevo `configure`.
- Quitar la línea `#define HAVE_TERM` del fichero `config.h` y ejecutar de nuevo `make`.

Si el enlazador no puede hallar `-lz` cuando enlaza programas cliente, probablemente el problema sea que el fichero `libz.so` se instaló en `/usr/local/lib`. Este problema puede resolverse con alguno de los siguientes métodos:

- Agregar `/usr/local/lib` a `LD_LIBRARY_PATH`.
- Agregar un vínculo a `libz.so` desde `/lib`.
- Si se está utilizando Solaris 8, se puede instalar el opcional `zlib` desde el CD de distribución del sistema operativo.
- Ejecutar `configure` con la opción `--with-named-z-libs=no` cuando se compila MySQL.

2.12.3.2. Notas sobre Solaris x86

En Solaris 8 sobre x86, `mysqld` realiza un volcado de núcleo si se quitan los símbolos de depuración utilizando `strip`.

Si se emplea `gcc` o `egcs` en Solaris x86 y se experimentan problemas con volcados de núcleo bajo carga, se debería usar el siguiente comando `configure`:

```
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
```

Esto evita inconvenientes con la biblioteca `libstdc++` y con excepciones de C++.

Si esto no funciona, se deberá compilar una versión de depuración y ejecutarla con un fichero de seguimiento (trace) o bajo `gdb`. Consulte Sección D.1.3, “Depurar `mysqld` con `gdb`”.

2.12.4. Notas sobre BSD

Esta sección proporciona información sobre el uso de MySQL en variantes de BSD Unix.

2.12.4.1. Notas sobre FreeBSD

Para ejecutar MySQL se recomienda FreeBSD 4.x o posterior, porque el paquete de subprocesos está mucho más integrado. Para lograr un sistema seguro y estable, se deberían emplear solamente kernels de FreeBSD marcados con `-RELEASE`.

La forma más fácil (y más empleada) de instalar MySQL es utilizar los ports (herramientas para crear, actualizar, y quitar paquetes de aplicaciones) de `mysql-server` y `mysql-client` disponibles en http://www.freebsd.org/. Utilizar estos ports conlleva los siguiente benficios:

- Un servidor MySQL funcional, con todas las optimizaciones conocidas para la versión de FreeBSD donde se instala.
- Configuración y compilación automáticas.
- Scripts de inicio instalados en `/usr/local/etc/rc.d`.
- Es posible emplear `pkg_info -L` para ver los ficheros que hay instalados.
- Es posible emplear `pkg_delete` para quitar MySQL del ordenador.

Se recomienda utilizar MIT-pthreads en FreeBSD 2.x, y los subprocesos (threads) nativos en la Versión 3 en adelante. En algunas versiones 2.2.x es posible utilizar subprocesos nativos, pero pueden aparecer problemas al finalizar `mysqld`.

Desafortunadamente, ciertas llamadas a funciones en FreeBSD no son todavía totalmente aptas para subprocesos. Sobre todo, esto incluye a la función `gethostbyname()`, la cual es utilizada por MySQL para convertir nombres de host en direcciones IP. Bajo ciertas circunstancias, el proceso `mysqld` de repente absorbe el 100% de la capacidad de la CPU y deja de responder. Si ocurre este problema, inténtese iniciar MySQL usando la opción `--skip-name-resolve`.

Alternativamente puede enlazarse MySQL en FreeBSD 4.x empleando la biblioteca LinuxThreads, la cual evita algunos problemas que tiene la implementación de subprocesos nativa de FreeBSD. Puede encontrarse una muy buena comparación entre LinuxThreads y la implementación nativa en el artículo de Jeremy Zawodny *FreeBSD or Linux for your MySQL Server? (FreeBSD o Linux para su Servidor MySQL?)* en http://jeremy.zawodny.com/blog/archives/000697.html.

Los problemas conocidos al utilizar LinuxThreads en FreeBSD son:

- Los valores de tiempos de conexión (`wait_timeout`, `interactive_timeout` y `net_read_timeout`) no se respetan. El síntoma es que las conexiones persistentes se congelan por

un tiempo muy largo, sin cerrarse, y matar ('kill') el subproceso no tendrá efecto hasta que éste pase al siguiente comando.

Esto probablemente sea un indicio de un problema en el manejo de señales en la biblioteca de procesos, donde la señal no interrumpe una lectura pendiente. Se supone que esto se resolvió en FreeBSD 5.0

El proceso de compilación de MySQL necesita GNU make (`gmake`) para llevarse a cabo. Si no está disponible GNU `make`, habrá que instalarlo antes de compilar MySQL.

La forma recomendada de compilar e instalar MySQL en FreeBSD con `gcc` (2.95.2 y superior) es:

```
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 &
```

Si se tiene conocimiento de que `configure` utiliza MIT-pthreads, se debería leer las notas para MIT-pthreads. Consulte Sección 2.8.5, “Notas sobre MIT-pthreads”.

Si se obtiene un error de `make install` donde anuncia que no se puede hallar el fichero `/usr/include/pthreads`, es porque `configure` no detectó que se necesitan MIT-pthreads. Para solucionar este problema, debe eliminarse `config.cache`, y luego ejecutar nuevamente `configure` con la opción `--with-mit-threads`.

Se debe estar seguro de que la configuración de resolución de nombres es la correcta. De otras maneras, se pueden experimentar demoras o fallas al conectarse a `mysqld`. También hay que verificar que sea correcta la entrada para `localhost` en el fichero `/etc/hosts`. El fichero debería comenzar con una línea similar a esta:

```
127.0.0.1       localhost localhost.your.domain
```

Se sabe que FreeBSD tiene en forma predeterminada un límite de manejadores de ficheros muy bajo. Consulte Sección A.2.17, “No se encontró el fichero”. Hay que iniciar el servidor utilizando la opción `--open-files-limit` para `mysqld_safe`, o elevar en el fichero `/etc/login.conf` el límite para el usuario `mysqld` y recompilarlo con `cap_mkdb /etc/login.conf`. También hay que asegurarse de establecer la clase apropiada de usuario en el fichero de contraseñas si no se está empleando el predeterminado (utilizar `chpass mysqld-user-name`). Consulte Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.

Si se dispone de mucha memoria,se podría considerar la recompilación del kernel para permitirle a MySQL utilizar más de 512Mb de RAM. Para más información, ver la `opción MAXDSIZ` en el fichero de configuración LINT.

Si se tienen problemas con la fecha actual en MySQL, podría ser de ayuda establecer al valor de la variable `TZ`. Consulte Apéndice E, *Variables de entorno*.

2.12.4.2. Notas sobre NetBSD

Para compilar en NetBSD, se necesitará GNU `make`. De otro modo, el proceso fallará cuando `make` intente ejecutar `lint` en ficheros de C++.

2.12.4.3. Notas sobre OpenBSD 2.5

En OpenBSD versión 2.5, se puede compilar MySQL con subprocesos nativos empleando las siguientes opciones:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.12.4.4. Notas sobre OpenBSD 2.8

Si se obtiene un error como `Error in accept:: Bad file descriptor` o el error 9 al intentar abrir tablas o directorios, el problema podría ser que no se tienen asignados suficientes descriptores de fichero para MySQL.

En este caso, hay que intentar iniciar `mysqld_safe` como `root` con las siguientes opciones:

```
mysqld_safe --user=mysql --open-files-limit=2048 &
```

2.12.4.5. Notas sobre BSD/OS Version 2.x

Si se obtiene el siguiente error al compilar con MySQL, es porque el valor de `ulimit` para la memoria virtual es muy bajo:

```
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
```

Debe intentarse emplear `ulimit -v 80000` y ejecutar `make` nuevamente. Si esto no funciona y se está utilizando `bash`, cambiar a `csh` o `sh`; algunos usuarios de BSDI han informado problemas con `bash` y el comando `ulimit`.

Si se está empleando `gcc`, también se tendrá que utilizar el flag `--with-low-memory` para `configure` para poder compilar `sql_yacc.cc`.

Si se tienen problemas con la fecha actual en MySQL, podría ser de ayuda establecer al valor de la variable `TZ`. Consulte Apéndice E, *Variables de entorno*.

2.12.4.6. Notas sobre BSD/OS Version 3.x

Se debe actualizar a BSD/OS Versión 3.1. Si no es posible, instalar el parche BSDI M300-038.

Utilizar el siguiente comando al configurar MySQL:

```
env CXX=shlicc++ CC=shlicc2 \
./configure \
    --prefix=/usr/local/mysql \
    --localstatedir=/var/mysql \
    --without-perl \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

También la siguiente manera funciona:

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure \
    --prefix=/usr/local/mysql \
```

```
    --with-unix-socket-path=/var/mysql/mysql.sock
```

Si se desea se puede cambiar la ubicación de los directorios, o no especificar ninguna opción para que se usen los valores predeterminados.

Si se tienen problemas de rendimiento bajo un uso intensivo, inténtese usar la opción `--skip-thread-priority` con `mysqld`. Esto ejecuta todos los subprocesos con la misma prioridad. En BSDI Versión 3.1, esto dará mejor rendimiento al menos hasta que BSDI mejore su programador de subprocesos.

Si se obtiene el error `virtual memory exhausted` al compilar, se debería intentar con `ulimit -v 80000` y ejecutar `make` nuevamente. Si esto no funciona y se está utilizando `bash`, cambiar a `csh` o `sh`; algunos usuarios de BSDI han informado problemas con `bash` y el comando `ulimit`.

2.12.4.7. Notas sobre BSD/OS Version 4.x

BSDI versión 4.x tiene algunos errores relacionados con los subprocesos. Si se desea usar MySQL en esta versión, se debería instalar todos los parches relacionados con subprocesos. Al menos se debería instalar el parche M400-023.

En algunos sistemas BSDI versión 4.x, se pueden tener problemas con las bibliotecas compartidas. El síntoma es que no se pueden ejecutar programas cliente, por ejemplo, `mysqladmin`. En este caso, se necesitará reconfigurar con la opción `--disable-shared` para que no se usen bibliotecas compartidas.

Algunos clientes han tenido problemas con BSDI 4.0.1 donde el binario `mysqld` no puede abrir tablas pasado un tiempo. Es debido a algunos errores relacionados con el sistema y las bibliotecas que provocan que `mysqld` cambie el directorio actual sin habérselo solicitado.

La solución puede ser actualizar MySQL a la versión 3.23.34 o posterior o bien, después de ejecutar `configure`, quitar la línea `#define HAVE_REALPATH` del fichero `config.h` antes de ejecutar `make`.

Nótese que esto significa que en BSDI no se puede enlazar simbólicamente los directorios de una base de datos a otra base de datos o una tabla a otra base de datos. (Sí es posible establecer un vínculo simbólico a otro disco).

2.12.5. Notas sobre otros Unix

2.12.5.1. Notas sobre HP-UX Version 10.20

Hay un par de pequeños problemas al compilar MySQL en HP-UX. Se recomienda que se utilice `gcc` en lugar del compilador nativo de HP-UX, ya que `gcc` produce mejor código.

Se recomienda utilizar `gcc` 2.95 en HP-UX. No hay que utilizar flags de optimización intensiva (como `-O6`) porque pueden no ser seguros en HP-UX.

La siguiente línea de `configure` debería funcionar con `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
```

La siguiente línea de `configure` debería funcionar con `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
```

2.12.5.2. Notas sobre HP-UX Version 11.x

Debido a algunos errores críticos en las bibliotecas estándar de HP-UX, se deberían instalar los siguientes parches antes de ejecutar MySQL en HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

Esto soluciona el problema de obtener `EWOULDBLOCK` `EWOULDBLOCK` de `recv()` y `EBADF` de `accept()` en aplicaciones con subprocesos o hebradas (threaded).

Si se está empleando `gcc` 2.95.1 en un sistema HP-UX 11.x sin parches, se podría obtener el siguiente 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:
```

El problema es que HP-UX no define `pthreads_atfork()` en forma consistente. Tiene prototipos conflictivos en `/usr/include/sys/unistd.h`:184 y `/usr/include/sys/pthread.h`:440.

Una solución es copiar `/usr/include/sys/unistd.h` dentro de `mysql/include` y editar `unistd.h` y cambiarlo para que coincida con la definición en `pthread.h`. Hay que hallar esta línea:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
                                          void (*child)());
```

Y cambiarla para que sea así:

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                                          void (*child)(void));
```

Después de realizar el cambio, la siguiente línea de `configure` debería funcionar:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

Si se está empleando el compilador HP-UX, se puede utilizar el siguiente comando (el cual fue probado con `cc` B.11.11.04):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
```

```
    --with-extra-character-set=complex
```

Se podrá ignorar cualquier error de este tipo:

```
aCC: warning 901: unknown option: `-3': use +help for online
documentation
```

Si se obtiene el siguiente error desde `configure`, verificar si no se tiene la ruta al compilador K&R antes que la ruta al compilador HP-UX para C y C++:

```
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.
```

Otra razón que puee impedir la compilación es que no se hayan definido los flags `+DD64` tal como se ha descripto.

Otra posibilidad para HP-UX 11 es emplear los binarios MySQL provistos en http://dev.mysql.com/downloads, los cuales fueron compilados y probados por MySQL AB. También se han recibido informes de que los binarios de MySQL provistos con HP-UX 10.20 se ejecutan correctamente en HP-UX 11. Si se encuentran problemas, se debería verificar si HP-UX tiene todos los parches necesarios.

2.12.5.3. Notas sobre IBM-AIX

La detección automática de `xlC` no se encuentra presente en Autoconf, de modo que habrá que inicializar una cantidad de variables antes de ejecutar `configure`. El siguiente ejemplo utiliza el compilador de IBM:

```
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
```

Estas opciones se emplean para compilar la distribución MySQL que se encuentra en http://www-frec.bull.com/.

Si se cambia el `-O3` por `-O2` en la linea de `configure` anterior, también habrá que quitar la opción `-qstrict`. Esto es una limitación en el compilador de C de IBM.

Si se está empleando `gcc` o `egcs` para compilar MySQL, se *debe* utilizar el flag `-fno-exceptions`, porque la gestión de excepciones en `gcc`/`egcs` no es apta para subprocesos. (Esto se probó con `egcs` 1.1.) También hay algunos problemas conocidos con el ensamblador de IBM, el cual puede generar código defectuoso cuando se lo usa con `gcc`.

Se recomienda usar la siguiente línea de `configure` con `egcs` y `gcc` 2.95 en 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
```

La opción `-Wa,-many` se necesita para que el proceso de compilación tenga éxito. IBM está al tanto de este problema pero no tiene urgencia por resolverlo porque hay disponible una solución alternativa. No se sabe si `-fno-exceptions` se requiere con `gcc` 2.95, pero como MySQL no utiliza excepciones y la opción genera código más rápido, se recomienda usarlo siempre con `egcs` / `gcc`.

Si se obtiene un problema con el código ensamblador, hay que cambiar la opción `-mcpu=xxx` para que concuerde con la CPU del usuario. Generalmente puede necesitarse `power2`, `power`, o `powerpc`. O bien podría necesitarse `604` o `604e`. No está confirmado, pero se sospecha que `power` puede muy bien ser seguro la mayoría de las veces, incluso en un ordenador de 2 procesadores.

Si se desconoce el tipo de CPU que se posee, ejecutar un comando `uname -m`. Este produce una cadena con un aspecto similar a `000514676700`, con el formato `xxyyyyyymmss`, donde `xx` y `ss` son siempre `00`, `yyyyyy` es un identificador único de sistema, y `mm` es el identificar de la arquitectura de la CPU. La tabla con estos valores se encuentra en http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm.

Esto proporciona el tipo y modelo del ordenador que se está usando.

Si se tienen problemas con señales (MySQL termina abruptamente al someterlo a carga intensiva), puede tratarse de un error del SO relacionado con subprocesos y señales. En este caso, hay que indicarle a MySQL que no utilice señales, configurándolo como sigue:

```
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
```

Esto no afecta el rendimiento de MySQL, pero tiene el efecto secundario de que no se pueden terminar procesos de clientes que están “durmiendo” (“sleeping”) en una conexión usando `mysqladmin kill` o `mysqladmin shutdown`. El cliente terminará cuando emita su siguiente comando.

En algunas versiones de AIX, el enlazado con `libbind.a` provoca que `getservbyname()` haga un volcado de núcleo. Este es un error en AIX y debería informarse a IBM.

Para AIX 4.2.1 y `gcc`, se deben hacer los siguientes cambios.

Después de configurar, editar `config.h` y `include/my_config.h` y cambiar la línea que dice:

```
#define HAVE_SNPRINTF 1
```

a esto:

```
#undef HAVE_SNPRINTF
```

Y, finalmente, en `mysqld.cc`, se necesitará agregar un prototipo para `initgroups()`.

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

Si se necesita asginar mucha memoria para el proceso `mysqld`, no basta con utilizar `ulimit -d unlimited`. También habrá que modificar `mysqld_safe` agregando una línea como la siguiente:

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

Se puede encontrar más información acerca de usar grandes cantidades de memoria en http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm.

2.12.5.4. Notas sobre SunOS 4

En SunOS 4, se necesita MIT-pthreads para compilar MySQL. Esto a su vez significa que que se necesitará GNU `make`.

Algunos sistemas SunOS 4 tienen problemas con las bibliotecas dinámicas y `libtool`. Se puede usar la siguiente línea en `configure` para evitar este problema:

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

Al compilar `readline`, se pueden obtener advertencias sobre definiciones duplicadas. Estas pueden ignorarse.

Al compilar `mysqld`, se producen algunas advertencias del tipo `implicit declaration of function` que pueden ignorarse.

2.12.5.5. Notas Alpha-DEC-UNIX (Tru64)

Si se está utilizando `egcs` 1.1.2 en Unix Digital, se debería actualizar a `gcc` 2.95.2, porque `egcs` tiene algunos errores serios en DEC.

Al compilar programas hebrados (threaded) en Unix Digital, la documentación recomienda utilizar la opción `-pthread` con `cc` y `cxx` y las bibliotecas `-lmach -lexc` (adicionalmente a `-lpthread`). Se debería ejecutar `configure` de una forma parecida a esta:

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

Cuando se compila `mysqld`, se podrían ver un par de advertencias similares a estas:

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

Pueden ser ignoradas sin problemas. Ocurren porque `configure` sólo puede detectar errores y no advertencias.

Si se inicia el servidor directamente desde la línea de comandos, se podría tener el problema de que finalice al terminar la sesión de usuario en el SO. (Cuando se termina la sesión, los procesos pendientes reciben una señal `SIGHUP`). Si eso sucede, debe intentarse iniciar el servidor de esta manera:

```
nohup mysqld [options] &
```

`nohup` provoca que el comando a continuación ignore cualquier señal `SIGHUP` enviada desde la terminal. Alternativamente, se puede iniciar el servidor ejecutando `mysqld_safe`, lo cual invoca a `mysqld` usando `nohup`. Consulte Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”.

Si se tiene un problema al compilar `mysys/get_opt.c`, quítese la linea `#define _NO_PROTO` del comienzo de dicho fichero.

Si se está empleando el compilador CC de Compaq, la siguiente línea de `configure` debería funcionar:

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed 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
```

Si al compilar `mysql` se tienen problemas con `libtool` usando bibliotecas compartidas como se indicó, se puede evitar el problema empleando estos comandos:

```
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
```

2.12.5.6. Notas sobre Alpha-DEC-OSF/1

Si ocurren problemas al compilar y se tienen instalados DEC `CC` y `gcc`, se debe intentar ejecutar `configure` de este modo:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Si ocurren problemas con el fichero `c_asm.h`, se puede crear y utilizar un fichero `c_asm.h` "silencioso" con:

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Tener en cuenta que los siguientes problemas con el programa `ld` pueden ser corregidos descargando el último conjunto de parches para DEC (Compaq) desde: http://ftp.support.compaq.com/public/unix/.

En OSF/1 V4.0D y el compilador "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)", el compilador tiene algún comportamiento extraño (símbolos `asm` indefinidos) `/bin/ld` también parece estar defectuoso (durante el enlazado de `mysqld` ocurren errores del tipo `_exit undefined`). En estos sistemas se optó por compilar MySQL con la siguiente línea de `configure`, reemplazando `/bin/ld` con la versión para OSF 4.OC.

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

Con el compilador Digital "C++ V6.1-029", se debería hacer lo siguiente:

```
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"
```

En algunas versiones OSF/1, la función `alloca()` está defectuosa. Esto se soluciona quitando la línea de `config.h` que define `'HAVE_ALLOCA'`.

La función `alloca()` puede tener también un prototipo incorrecto en `/usr/include/alloca.h`. Esta advertencia resultante de esa situación puede ignorarse.

`configure` utiliza automáticamente la siguiente libreria de subprocesos: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

Al utilizar `gcc`, se debería ejecutar `configure` de este modo:

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

Si se tienen problemas con señales (MySQL termina abruptamente al someterlo a carga intensiva), puede tratarse de un error del SO relacionado con subprocesos y señales. En este caso, hay que indicarle a MySQL que no utilice señales, configurándolo como sigue:

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

Esto no afecta el rendimiento de MySQL, pero tiene el efecto secundario de que no se pueden terminar procesos de clientes que están “durmiendo” (“sleeping”) en una conexión usando `mysqladmin kill` o `mysqladmin shutdown`. El cliente terminará cuando emita su siguiente comando.

Con `gcc` 2.95.2, se puede encontrar el siguiente error de compilación:

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

Para solucionar esto, habría que posicionarse en el directorio `sql`, y cortar y pegar la última línea de `gcc`, pero cambiando `-O3` por `-O0` (o agregando `-O0` inmediatamente después de `gcc` si no se tiene ninguna opción `-O` en la línea de compilación). Luego de hacer esto, se puede volver al directorio principal (top-level) y ejecutar nuevamente `make`.

2.12.5.7. Notas sobre SGI Irix

Si se está utilizando Irix Versión 6.5.3 o posterior, `mysqld` será capaz de crear procesos únicamente si se lo ejecutó como un usuario con privilegios `CAP_SCHED_MGT` (como el usuario `root`) o bien darle este privilegio al servidor `mysqld` con el siguiente comando del shell:

```
```

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

Es posible que haya que quitar la definición de algunos símbolos en `config.h` luego de ejecutar `configure` y antes de compilar.

En algunas implementaciones de Irix, la función `alloca()` está defectuosa. Si el servidor `mysqld` termina abruptamente en algunas sentencias `SELECT`, deberán quitarse de `config.h` las líneas que definen `HAVE_ALLOC` y `HAVE_ALLOCA_H`. Si `mysqladmin create` no funciona, habrá que quitar de `config.h` la línea que define `HAVE_READDIR_R`. También es posible que haya que quitar la línea `HAVE_TERM_H`.

SGI recomienda que se instalen en conjunto todos los parches de esta página: http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

Como mínimo, se deberían instalar las últimas versiones del kernel, de `rld`, y de `libc`.

Definitivamente serán necesarios todos los parches POSIX de esta página, para dar soporte a pthreads:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

Si se obtiene el siguiente error al compilar `mysql.cc`:

```
"/usr/include/curses.h", line 82: error(1084):
invalid combination of type
```

Habrá que teclear lo siguiente en el directorio principal del árbol de código fuente de MySQL:

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

También se informaron problemas de sincronización (scheduling). Si sólo se está ejecutando un proceso, el rendimiento es bajo. Esto se evita iniciando otro cliente. Esto puede conducir a un incremento en la velocidad de dos a diez veces de ese momento en adelante para el otro hilo. Es este un problema difícil de entender con los suprocesos de Irix; habrá que improvisar para hallar soluciones hasta que sea arreglado.

Si se está compilando con `gcc`, se puede usar el siguiente comando de `configure`:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
    --with-named-thread-libs=-lpthread
```

Lo siguiente funciona en Irix 6.5.11 con compiladores nativos Irix C y C++ versión 7.3.1.2.

```
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
```

2.12.5.8. Notas sobre SCO UNIX y OpenServer 5.0.x

El port actual se probó solamente en sistemas sco3.2V5.0.5, sco3.2v5.0.6, y sco3.2v5.0.7. También está en desarrollo un port para sco3.2v4.2. Open Server 5.0.8 (Legend) tiene soporte nativo para subprocesos y permite ficheros de más de 2GB. Actualmente el máximo tamaño de fichero permitido es 2GB.

En MySQL AB se pudo compilar MySQL con el siguiente comando de `configure` en OpenServer con `gcc` 2.95.3.

```
CC=gcc CXX=gcc ./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-innodb \
    --with-openssl --with-vio --with-extra-charsets=complex
```

`gcc` puede descargarse de ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj.

Este sistema de desarrollo requiere el Suplemento del Entorno de Ejecución de OpenServer (OpenServer Execution Environment Supplement) oss646B en OpenServer 5.0.6 y oss656B y las bibliotecas OpenSource halladas en gwxlibs. Todas las herramientas OpenSource están en el directorio `opensrc`, en ftp://ftp.sco.com/pub/openserver5/opensrc/.

MySQL AB recomienda utilizar el último release en producción de MySQL.

SCO proporciona parches del sistema operativo en ftp://ftp.sco.com/pub/openserver5 para OpenServer 5.0.[0-6] y ftp://ftp.sco.com/pub/openserverv5/507 para OpenServer 5.0.7.

SCO proporciona información sobre problemas de seguridad solucionados en ftp://ftp.sco.com/pub/security/OpenServer para OpenServer 5.0.x.

El máximo tamaño de fichero en un sistema OpenServer 5.0.x es 2GB.

La memoria total que puede direccionarse para buffers de streams, clists, y bloqueo de registros, no puede exceder de 60MB en OpenServer 5.0.x.

Los buffers de streams son direccionados en páginas de 4096 bytes, los clists en páginas de 70 bytes, y los bloqueos de registros en páginas de 64 bytes, de modo que:

```
(NSTRPAGES * 4096) + (NCLIST * 70) + (MAX_FLCKREC * 64) <= 62914560
```

Seguir este procedimiento para configurar la opción Servicios de Base de Datos (Database Services). Si no se está seguro si una aplicación requiere esto, ver la documentación de la aplicación.

1. Iniciar sesión como `root`.
2. Habilitar el driver SUDS mediante la edición del fichero `/etc/conf/sdevice.d/suds`. Cambiar la `N` del segundo párrafo por una `Y`.
3. Utilizar `mkdev aio` o el Gestor de Hardware/Kernel (Hardware/Kernel Manager) para habilitar el soporte de entrada/salida asincrónica, y enlazar nuevamente el kernel. Para permitir a los usuarios reservar memoria para usar con este tipo de E/S, actualizar el fichero aiomemlock(F). Este fichero debería actualizarse para que incluya los nombre de los usuarios que pueden utilizar ESA (Entrada Salida Asincrónica, o AIO, Asynchronoys I/O) y las máximas cantidades de memoria que pueden reservar.
4. Muchas aplicaciones usan binarios setuid de forma que solamente se necesita especificar un único usuario. Ver la documentación provista con la aplicación para ver si este es su caso.

Después de completar este proceso, reiniciar el sistema para crear un nuevo kernel que incorpore estos cambios.

Por defecto, las entradas en `/etc/conf/cf.d/mtune` están configuradas así:

```
```

```
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
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
```

Se recomienda establecer estos valores de la siguiente manera:

`NOFILES` debería ser 4096 o 2048.

`MAXUP` debería ser 2048.

Para hacer cambios al kernel, posicionarse con `cd` en el directorio `/etc/conf/bin` y utilizar `./idtune` *`nombre parámetro`* para realizar los cambios. Por ejemplo, para cambiar `SEMMS` a un valor de `200`, ejecutar estos comandos como usuario `root`:

```
# cd /etc/conf/bin
# ./idtune SEMMNS 200
```

Se recomienda optimizar el sistema, pero los valores a utilizar dependerán del número de usuarios que acceden a la aplicación o base de datos y el tamaño de la base de datos (esto es, el pool de buffer utilizado). Lo siguiente modifica los parámetros de kernel definidos en `/etc/conf/cf.d/stune`:

`SHMMAX` (valor recomendado: 128MB) and `SHMSEG` (valor recomendado: 15). Estos parámetros inciden en el motor de base de datos MySQL para la creación de pools de buffers de usuario).

`NOFILES` y `MAXUP` deberían establecerse en por lo menos 2048.

`MAXPROC` debería establecerse al menos en 3000/4000 (dependiendo del número de usuarios) o más.

También se recomienda el empleo de la siguiente fórmula para calcular el valor para `SEMMSL`, `SEMMNS` y `SEMMNU`:

```
SEMMSL = 13
```

El 13 es lo que se halló como el mejor valor para Progress y MySQL.

`SEMMNS` = `SEMMSL` * cantidad de servidores de bases de datos a ejecutarse en el sistema.

Establecer `SEMMNS` al valor de `SEMMSL` multiplicado por la cantidad de servidores de bases de datos (máximo) que se ejecutan en el sistema al mismo tiempo.

```
SEMMNU = SEMMNS
```

Establecer el valor de `SEMMNU` al mismo valor que tiene `SEMMNS`. Posiblemente se pueda establecer a un 75% del valor de `SEMMNS`, pero esta es una estimación conservadora.

Se necesitará instalar al menos las "Bibliotecas de Desarrollo de Aplicaciones y Enlazador OpenServer de SCO" ("SCO OpenServer Linker and Application Development Libraries") o el Sistema de Desarrollo de OpenServer (OpenServer Development System) para ejecutar `gcc`. No se puede simplemente emplear el sistema de desarrollo GCC sin instalar alguno de los mencionados.

Antes se deberá obtener el paquete FSU Pthreads e instalarlo. Puede descargarse de http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz. También puede descargarse un paquete precompilado en ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz.

FSU Pthreads puede compilarse con SCO Unix 4.2 con tcpip, o utilizando OpenServer 3.0 u OpenDesktop 3.0 (OS 3.0 ODT 3.0) con el SCO Development System instalado utilizando un buen port de GCC 2.5.x. Hay muchos problemas que ocurren sin un buen port. El port para este producto necesita el SCO Unix Development System. Sin él, no se tendrán las bibliotecas y el enlazador que se necesitan. También se requiere el fichero `SCO-3.2v4.2-includes.tar.gz`. Este fichero contiene los cambios a los ficheros de inclusión de SCO Development que se necesitan para lograr compilar MySQL. Habrá que reemplazar los ficheros de inclusión existentes en el sistema con estos ficheros de cabecera modificados. Pueden descargarse de ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

Todo lo que se debería necesitar para compilar FSU Pthreads es ejecutar GNU `make`. El `Makefile` en FSU-threads-3.14.tar.gz está configurado para crear FSU-threads.

Se puede ejecutar `./configure` en el directorio `threads/src` y seleccionar la opción SCO OpenServer. Este comando copia `Makefile.SCO5` a `Makefile`. Luego, se ejecuta `make`.

Para instalar en el directorio predeterminado `/usr/include`, iniciar sesión como `root`, luego posicionarse con `cd` en el directorio `thread/src` y ejecutar `make install`.

Recordar que debe usarse GNU `make` para crear MySQL.

Nota: Si no se inicia `mysqld_safe` como usuario `root`, se deberían obtener solamente por defecto los 110 ficheros abiertos por proceso. `mysqld` deja constancia de esto en el fichero de registro (log).

Con SCO 3.2V4.2, se debería usar FSU Pthreads versión 3.14 o posterior. El siguiente comando `configure` debería funcionar:

```
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"
```

Se pueden tener problemas con algunos ficheros de inclusión. En este caso, pueden hallarse nuevos ficheros de inclusión específicos para SCO en ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

Se debería descompactar este fichero en el directorio `include` del árbol de código fuente de MySQL.

Notas relativas a SCO development:

- MySQL debería detectar automáticamente FSU Pthreads y enlazar `mysqld` con `-lgthreads -lsocket -lgthreads`.
- Las bibliotecas de desarrollo SCO son reentrantes (compartidas por varios usuarios o procesos, una condición indispensable en la programación de multitarea) en FSU Pthreads. SCO afirma que sus bibliotecas de funciones son reentrantes, por lo tanto deben ser reentrantes con FSU Pthreads. FSU Pthreads en OpenServer intenta utilizar el esquema de SCO para hacer bibliotecas reentrantes.
- FSU Pthreads (al menos la versión en ftp::/ftp.zenez.com) viene enlazada con GNU `malloc`. Si ocurren problemas con el uso de memoria, asegurarse de que `gmalloc.o` está incluido en `libgthreads.a` y `libgthreads.so`.
- En FSU Pthreads, las siguientes llamadas de sistema son compatibles con pthreads: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()`, y `wait()`.
- El parche CSSA-2001-SCO.35.2 (habitualmente listado como erg711905-dscr_remap security patch (version 2.0.0)) interrumpe los subprocesos FSU y deja inestable a `mysqld`. Hay que removerlo si se desea ejecutar `mysqld` en un ordenador con OpenServer 5.0.6.
- SCO proporciona parches del sistema operativo OpenServer 5.0.x en ftp://ftp.sco.com/pub/openserver5.
- Las soluciones a problemas de seguridad y `libsocket.so.2` para OpenServer 5.0.x están en ftp://ftp.sco.com/pub/security/OpenServer y ftp://ftp.sco.com/pub/security/sse.
- Soluciones de seguridad para Pre-OSR506. También, la solución para `telnetd` en ftp://stage.caldera.com/pub/security/openserver/ o ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/ así como `libsocket.so.2` y `libresolv.so.1` con instrucciones para instalar en sistemas pre-OSR506.

  Probablemente sea una buena idea instalar estos parches antes de compilar o utilizar MySQL.

A partir de Legend/OpenServer 6.0.0 se dispone de subprocesos nativos y se eliminó el límite de 2GB para el tamaño de los ficheros.

2.12.5.9. Notas sobre SCO UnixWare 7.1.x y OpenUNIX 8.0.0

Se recomienda utilizar el último release de producción de MySQL.

Se ha logrado compilar MySQL con el siguiente comando `configure` en UnixWare versión 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
```

Si se desea utilizar `gcc`, debe ser la versión 2.95.3 o posterior.

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

La versión de Berkeley DB que viene con UnixWare 7.1.4 u OpenServer 6.0.0 no se utiliza cuando se compila MySQL. En su lugar, MySQL utiliza su propia versión de Berkeley DB. El comando `configure` necesita compilar tanto una biblioteca estática como una dinámica en *`src_directory`*`/bdb/build_unix/`, pero no utiliza la versión de Berkeley DB que MySQL necesita. La solución es la siguiente.

1. Configurar para MySQL como de costumbre.
2. `cd bdb/build_unix/`
3. `cp -p Makefile to Makefile.sav`
4. Emplear las mismas opciones y ejecutar `../dist/configure`.
5. Ejecutar `gmake`.
6. `cp -p Makefile.sav Makefile`
7. Posicionarse en el directorio principal o raíz de código fuente y ejecutar `gmake`.

Esto posibilita que tanto la biblioteca dinámica como la estática sean creadas y funcionen.

SCO proporciona parches de sistema operativo en ftp://ftp.sco.com/pub/unixware7 para UnixWare 7.1.1, ftp://ftp.sco.com/pub/unixware7/713/ para UnixWare 7.1.3, ftp://ftp.sco.com/pub/unixware7/714/ para UnixWare 7.1.4, y ftp://ftp.sco.com/pub/openunix8 para OpenUNIX 8.0.0.

SCO proporciona información sobre soluciones a problemas de seguridad en ftp://ftp.sco.com/pub/security/OpenUNIX para OpenUNIX y ftp://ftp.sco.com/pub/security/UnixWare para UnixWare.

En forma predeterminada, el máximo tamaño de fichero en un sistema UnixWare 7.1.1 es de 1GB, pero en UnixWare 7.1.4 es de 1TB con VXFS. Algunas utilidades del Sistema Operativo tienen una limitación de 2GB. El máximo tamaño posible para ficheros de UnixWare 7 es 1TB con VXFS.

En UnixWare 7.1.4 no se necesita ninguna acción en especial para que soporte ficheros de gran tamaño, pero para habilitarlo en versiones anteriores de UnixWare 7.1.x, hay que ejecutar `fsadm`.

```
# fsadm -Fvxfs -o largefiles /
# fsadm /         * Nota
# ulimit unlimited
# cd /etc/conf/bin
# ./idtune SFSZLIM 0x7FFFFFFF     ** Nota
# ./idtune HFSZLIM 0x7FFFFFFF     ** Nota
# ./idbuild -B

* Este comando debería informar "largefiles".
** El valor 0x7FFFFFFF representa "infinito" (sin límite) para esos valores.
```

Reiniciar el sistema empleando `shutdown`.

por defecto, las entradas en `/etc/conf/cf.d/mtune` están establecidas en:

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
SSTKLIM         0x1000000       0x2000          0x7FFFFFFF
HSTKLIM         0x1000000       0x2000          0x7FFFFFFF
```

Se recomienda establecer estos valores como sigue:

```
SDATLIM 0x7FFFFFFF
HDATLIM 0x7FFFFFFF
SSTKLIM 0x7FFFFFFF
HSTKLIM 0x7FFFFFFF
SVMMLIM 0x7FFFFFFF
HVMMLIM 0x7FFFFFFF
SFNOLIM 2048
HFNOLIM 2048
```

Se recomienda ajustar el sistema, pero los valores de los parámetros a emplear dependen del número de usuarios que accederán a la aplicación o la base de datos y el tamaño de la base de datos (es decir, el uso que se hará del buffer pool -un caché de tablas y páginas-). Lo que sigue modifica los parámetros del kernel definidos en `/etc/conf/cf.d/stune`:

`SHMMAX` (valor recomendado: 128MB) y `SHMSEG` (valor recomendado: 15). Estos parámetros influyen en la forma en que el motor de bases de datos crea los buffer pools.

`SFNOLIM` y `HFNOLIM` deberían ser como máximo 2048.

`NPROC` debería establecerse a por lo menos 3000/4000 (dependiendo del número de usuarios).

También se recomienda emplear la siguiente fórmula para calcular los valores de `SEMMSL`, `SEMMNS`, y `SEMMNU`:

```
SEMMSL = 13
```

13 es el valor que se halló como el mejor para Progress y MySQL.

`SEMMNS` = `SEMMSL` * cantidad de servidores de bases de datos a ejecutar en el sistema.

Establecer `SEMMNS` al valor de `SEMMSL` multiplicado por el número máximo de servidores que se ejecutarán en el sistema al mismo tiempo.

`SEMMNU` = `SEMMNS`

Establecer el valor de `SEMMNU` al mismo valor que tiene `SEMMNS`. Posiblemente se pueda establecer a un 75% del valor de `SEMMNS`, pero esta es una estimación conservadora.

2.12.5.10. Notas sobre SCO OpenServer 6.0.x

Las mejoras clave en OpenServer 6 incluyen:

- Soporte para ficheros más grandes, hasta 1 TB
- Soporte para multiprocesadores incrementado de 4 a 32 procesadores.

- Incremento del soporte de memoria hasta 64 GB.
- Se extendió la potencia de UnixWare dentro de OpenServer 6.
- Mejora dramática del rendimiento.

OpenServer 6.0.0 tiene las siguientes particularidades:

- `/bin` es para comandos que se comportan exactamente del mismo modo que OpenServer 5.0.x.
- `/u95/bin` es para comandos que están más en conformidad con los estándares, por ejemplo el soporte para el Sistema de Ficheros Grandes o LFS (Large File System).
- `/udk/bin` es para comandos que se comportan igual que en UnixWare 7.1.4. por defecto, el soporte para LFS.

La siguiente es una guía para configurar PATH en OpenServer 6. Si el usuario desea el OpenServer 5.0.x tradicional entonces `PATH` debería ser en primer lugar `/bin`. Si el usuario desea soporte para LFS entonces el PATH debería ser `/u95/bin:/bin`. Si desea primariamente soporte para UnixWare 7, debería ser `/udk/bin:/u95/bin:/bin:`.

Se recomienda utilizar el último release de producción de MySQL

En OpenServer Versión 6.0.x se ha logrado compilar MySQL con el siguiente comando `configure`:

```
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 \
    --enable-readline
```

Si se desea emplear `gcc`, debe ser `gcc` 2.95.3 o posterior.

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

La versión de Berkeley DB que viene con UnixWare 7.1.4 u OpenServer 6.0.0 no se utiliza cuando se compila MySQL. En su lugar, MySQL utiliza su propia versión de Berkeley DB. El comando `configure` necesita compilar tanto una biblioteca estática como una dinámica en *`src_directory`*`/bdb/build_unix/`, pero no utiliza la versión de Berkeley DB que MySQL necesita. La solución es la siguiente.

1. Configurar para MySQL como de costumbre.
2. `cd bdb/build_unix/`
3. `cp -p Makefile to Makefile.sav`
4. Emplear las mismas opciones y ejecutar `../dist/configure`.
5. Ejecutar `gmake`.
6. `cp -p Makefile.sav Makefile`
7. Posicionarse en el directorio principal o raíz de código fuente y ejecutar `gmake`.

Esto posibilita que tanto la biblioteca dinámica como la estática sean creadas y funcionen. OpenServer 6.0.0 también necesita parches para el árbol de código fuente MySQL y el parche para `config.guess` aplicado sobre `bdb/dist/config.guess`. Los parches pueden descargarse de ftp://ftp.zenez.com/pub/

zenez/prgms/mysql-4.1.12-osr6-patches.tar.gz y de ftp://ftp.zenez.com/pub/zenez/prgms/mysql-4.x.x-osr6-patches. Hay un fichero `README` para obtener ayuda.

Los parches del sistema operativo OpenServer 6 son proporcionados por SCO en ftp://ftp.sco.com/pub/openserver6.

SCO proporciona información sobre soluciones a problemas de seguridad en ftp://ftp.sco.com/pub/security/OpenServer.

En forma predeterminada, el máximo tamaño de fichero en un sistema OpenServer 6.0.0 es de 1TB. Algunas utilidades del Sistema Operativo tienen una limitación de 2GB. El máximo tamaño posible para ficheros de UnixWare 7 es 1TB con VXFS o HTFS.

En forma predeterminada, las entradas en `/etc/conf/cf.d/mtune` están establecidas en:

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
SSTKLIM         0x1000000       0x2000          0x7FFFFFFF
HSTKLIM         0x1000000       0x2000          0x7FFFFFFF
```

Se recomienda configurar estos valores en la siguiente forma:

```
SDATLIM 0x7FFFFFFF
HDATLIM 0x7FFFFFFF
SSTKLIM 0x7FFFFFFF
HSTKLIM 0x7FFFFFFF
SVMMLIM 0x7FFFFFFF
HVMMLIM 0x7FFFFFFF
SFNOLIM 2048
HFNOLIM 2048
```

Se recomienda ajustar el sistema, pero los valores de los parámetros a emplear dependen del número de usuarios que accederán a la aplicación o la base de datos y el tamaño de la base de datos (es decir, el uso que se hará del buffer pool -un caché de tablas y páginas-). Lo que sigue modifica los parámetros del kernel definidos en `/etc/conf/cf.d/stune`:

`SHMMAX` (Valor recomendado: 128MB) y `SHMSEG` (Valor recomendado: 15). Estos parámetros influyen en la forma en que el motor de bases de datos crea los buffer pools.

`SFNOLIM` y `HFNOLIM` deberían tener un valor máximo de 2048.

`NPROC` debería ser por lo menos 3000/4000 (dependiendo de la cantidad de usuarioa).

También se recomienda emplear la siguiente fórmula para calcular los valores de `SEMMSL`, `SEMMNS`, y `SEMMNU`:

```
SEMMSL = 13
```

13 es el valor que se halló como el mejor para Progress y MySQL.

`SEMMNS` = `SEMMSL` * cantidad de servidores de bases de datos a ejecutar en el sistema.

Establecer `SEMMNS` al valor de `SEMMSL` multiplicado por el número máximo de servidores que se ejecutarán en el sistema al mismo tiempo.

`SEMMNU` = `SEMMNS`

Establecer el valor de `SEMMNU` al mismo valor que tiene `SEMMNS`. Posiblemente se pueda establecer a un 75% del valor de `SEMMNS`, pero esta es una estimación conservadora.

2.12.6. Notas sobre OS/2

MySQL emplea varios ficheros abiertos. Debido a esto, se debería agregar al fichero `CONFIG.SYS` algo como lo siguiente:

```
SET EMXOPT=-c -n -h1024
```

Si no se hace así, se pueden producir los siguientes errores:

```
File 'xxxx' not found (Errcode: 24)
```

Al emplear MySQL en OS/2 Warp 3, se requiere el FixPack 29 o posterior. Con OS/2 Warp 4, se necesita el FixPack 4 o posterior. Este es un requisito de la biblioteca Pthreads. MySQL Debe ser instalado en una partición con un tipo que soporte nombres de fichero largos, tal como HPFS, FAT32, y otros.

El script `INSTALL.CMD` debe ejecutarse desde la línea de comandos de OS/2, `CMD.EXE`, y podría no funcionar con shells de remplazo como `4OS2.EXE`.

El script `scripts/mysql-install-db` ha cambiado su nombre. Se llama `install.cmd` y es un script REXX, el cual establece la configuración de seguridad por defecto de MySQL y crea en el Workplace Shell [nombre que recibe el sistema de ventanas de OS/2] los íconos de MySQL.

El soporte para módulo dinámico se compila pero no está completamente testeado. Los módulos dinámicos se deben compilar empleando la biblioteca de tiempo de ejecución de Pthreads.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrtl -I../include -I../regex -I.. \
    -o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Nota: debido a limitaciones en OS/2, el nombre (sin incluir extensión) de un módulo UDF no debe exceder de 8 caracteres. Los módulos se almacenan en el directorio `/mysql2/udf`; el script `safe-mysqld.cmd` coloca este directorio en la variable de entorno `BEGINLIBPATH`. Al utilizar módulos UDF, se ignora cualquier extensión que se especifique. Se asume `.udf`. Por ejemplo, en Unix, el módulo compartido podría llamarse `example.so` y la carga de una función contenida en él se haría así:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';
```

En OS/2, el modulo se llamaría `example.udf`, pero la extensión no se especifica:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.13. Notas sobre la instalación de Perl

En Perl, el soporte para MySQL está proporcionado a través de la interfaz cliente `DBI`/`DBD`. La interfaz requiere Perl Versión 5.6.0 o posterior. La misma *no funciona* en una versión anterior de Perl.

Para poder utilizar transacciones con Perl DBI, se necesita tener `DBD::mysql` versión 1.2216 o posterior. Se recomienda la versión 2.9003 o posterior.

Si se está utilizando la biblioteca cliente de MySQL 4.1, se deberá emplear `DBD::mysql` 2.9003 o posterior.

El soporte en Perl no se incluye con las distribuciones MySQL. Los módulos necesarios pueden descargarse de http://search.cpan.org para Unix, o utilizando el programa ActiveState `ppm` en Windows. Las siguientes secciones describen cómo hacerlo.

Se debe instalar el soporte en Perl para MySQL si se desean ejecutar los scripts de de pruebas de rendimiento de MySQL. Consulte Sección 7.1.4, “El paquete de pruebas de rendimiento (benchmarks) de MySQL”.

2.13.1. Instalación de Perl en Unix

El soporte MySQL en Perl requiere que se hayan instalado el soporte de programación de cliente MySQL (bibliotecas y ficheros de cabecera). La mayoría de los métodos de instalación crean los ficheros necesarios. Sin embargo, si se instaló MySQL en Linux a partir de ficheros RPM, hay que asegurarse de haber instalado el RPM de desarrollo. Los programas cliente están en el RPM de cliente, pero el soporte a la programación se encuentra en el RPM de desarrollo.

Si se desea instalar el soporte en Perl, los ficheros que se necesitarán pueden obtenerse desde la CPAN (Comprehensive Perl Archive Network, Red Integral de Archivo Perl) en http://search.cpan.org.

La forma más sencilla de instalar módulos Perl en Unix es utilizar el módulo `CPAN`. Por ejemplo:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

La instalación de `DBD::mysql` ejecuta una cantidad de pruebas. Estas pruebas intentan conectarse al servidor MySQL local, empleando el nombre de usuario y contraseña por defecto. (El nombre de usuario por defecto es el nombre usado para iniciar sesión en Unix, y en Windows es `ODBC`. La contraseña por defecto es “sin contraseña.”). Si no se puede conectar al servidor con estos valores (por ejemplo si la cuenta tiene una contraseña establecida), la prueba falla. Se puede emplear `force install DBD::mysql` para ignorar las pruebas fallidas.

`DBI` requiere el módulo `Data::Dumper`. Es posible que esté instalado, de lo contrario, se debería instalar antes que `DBI`.

Otra posibilidad es descargar las distribuciones de módulos en forma de ficheros `tar` comprimidos y compilar los módulos manualmente. Por ejemplo, para descompactar y compilar una distribución DBI, debe usarse un procedimiento como el siguiente:

1. Descompactar la distribución en el directorio actual:

   ```
   shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   Este comando crea un directorio llamado `DBI-VERSION`.

2. Hay que posicionarse en el directorio de más alto nivel de la distribución descompactada:

   ```
   shell> cd DBI-VERSION
   ```

3. Compilar la distribución:

   ```
   shell> perl Makefile.PL
   shell> make
   shell> make test
   shell> make install
   ```

El comando `make test` es importante porque verifica que el módulo esté funcionando. Nótese que, al ejecutar este comando durante la instalación de `DBD::mysql` para ejercitar el código de la interfaz, el servidor MySQL debe estar funcionando o de lo contrario la prueba fallará.

Es buena idea recompilar y reinstalar la distribución de `DBD::mysql` cada vez que se instale un nuevo release de MySQL, en particular si se advierte que todos los scripts `DBI` fallan luego de actualizar el servidor.

Si no se tienen privilegios para instalar los módulos Perl en el directorio del sistema, o si se desean instalar los módulos en forma local, la siguiente referencia puede ser útil: http://servers.digitaldaze.com/extensions/perl/modules.html#modules

Ver bajo el título “Installing New Modules that Require Locally Installed Modules.”

2.13.2. Instalar ActiveState Perl en Windows

En Windows, se debe hacer lo siguiente para instalar el módulo `DBD` con ActiveState Perl:

- Debe obtenerse ActiveState Perl en http://www.activestate.com/Products/ActivePerl/ e instalarlo.
- Abrir una ventana de consola (“ventana DOS”).
- Si se necesita, establecer el valor de la variable `HTTP_proxy`. Por ejemplo:

```
set HTTP_proxy=my.proxy.com:3128
```

- Iniciar el programa PPM:

```
C:\> C:\perl\bin\ppm.pl
```

- Si no se hizo antes, instalar `DBI`:

```
ppm> install DBI
```

- Si todo ha ido bien, ejecutar el siguiente comando:

```
install \
ftp://ftp.de.uu.net/pub/CPAN/authors/id/JWIED/DBD-mysql-1.2212.x86.ppd
```

Este procedimiento debería funcionar con ActiveState Perl Versión 5.6 o posterior.

Si no se puede hacer funcionar el procedimiento, se debería en su lugar instalar el driver MyODBC y conectarse al servidos MySQL a través de ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.13.3. Problemas en la utilización de la interfaz Perl `DBI`/`DBD`

Si Perl anuncia que no puede encontrar el módulo `../mysql/mysql.so`, entonces el problema probablemente sea que Perl no ha podido encontrar la biblioteca compartida `libmysqlclient.so`.

Esto debería poder solucionarse a través de alguno de los siguientes métodos:

- Compilar la distribución DBD::mysql con perl Makefile.PL -static -config en lugar de perl Makefile.PL.
- Copiar libmysqlclient.so al directorio donde se ubican las demás bibliotecas compartidas (probablemente, /usr/lib or /lib).
- Modificar las opciones -L utilizadas para compilar DBD::mysql para que reflejen la ubicación real de libmysqlclient.so.
- En Linux, puede agregarse al fichero /etc/ld.so.conf la ruta donde se localiza libmysqlclient.so.
- Agregar a la variable de entorno LD_RUN_PATH el directorio donde se ubica libmysqlclient.so. Algunos sistemas utilizan LD_LIBRARY_PATH en lugar de LD_RUN_PATH.

Hay que notar que si hay otras bibliotecas que el enlazador no puede hallar, se necesitarán modificar las opciones -L. Por ejemplo, si no se puede hallar libc porque está en el directorio /lib y el enlazador está especificando -L/usr/lib, hay que cambiar la opción -L para que sea -L/lib o agregar -L/lib al comando de enlazado existente.

Si se obtienen los siguientes errores de DBD::mysql, probablemente se está utilizando gcc (o un binario antiguo compilado con gcc):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Agregar -L/usr/lib/gcc-lib/... -lgcc al comando de enlazado cuando se compila mysql.so (verificar la salida de make para mysql.so al compilar el cliente Perl). La opción -L debería especificar la ruta donde se localiza libgcc.a.

Otra causa de este problema es que Perl y MySQL no estén compilados (ambos) con gcc. En este caso, se puede resolver la desigualdad compilando a los dos con gcc.

Al ejecutar las pruebas, puede verse el siguiente error de DBD::mysql:.

```
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.
```

Esto significa que se necesita incluir la biblioteca de compresión -lz en la línea de enlazado. Puede hacerse cambiando la siguiente linea en el fichero lib/DBD/mysql/Install.pm.

Copiar libmysqlclient.so al directorio donde se ubican las otras bibliotecas compartidas (probablemente, /usr/lib or /lib).

Modificar las opciones -L usadas para compilar DBD::mysql para que reflejen la ubicación real de libmysqlclient.so.

En Linux, puede agregarse al fichero /etc/ld.so.conf la ruta donde se localiza libmysqlclient.so.

Agregar a la variable de entorno LD_RUN_PATH el directorio donde se ubica libmysqlclient.so. Algunos sistemas utilizan LD_LIBRARY_PATH en lugar de LD_RUN_PATH.

Hay que notar que si hay otras bibliotecas que el enlazador no puede hallar, se necesitarán modificar las opciones -L. Por ejemplo, si no se puede hallar libc:

```
$sysliblist .= " -lm";
```

Cambiar la línea a:

```
$sysliblist .= " -lm -lz";
```

Despues de esto, *debe* ejecutarse `make realclean` y proceder con la instalación desde el principio.

Si se desea instalar DBI en SCO, se tendrá que editar el fichero `Makefile` en DBI-*xxx* y en cada subdirectorio. Notar que lo que sigue asume que se tiene `gcc` 2.95.2 o posterior:

```
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
```

Estos cambios se necesitan porque el cargador dinámico (dynaloader) de Perl no cargará los módulos `DBI` si fueron compilados con `icc` o `cc`.

Si se desea utilizar el módulo Perl en un sistema que no posee soporte para enlazado dinámico (como SCO), se deberá generar una versión estática de Perl que incluya `DBI` y `DBD::mysql`. La forma en que esto funciona es: se genera una versión de Perl con el código `DBI` enlazado y se la instala por encima del Perl actual. Luego se lo utiliza para compilar una versión de Perl que adicionalmente tiene incluído el código de `DBD`, y se lo instala.

En SCO, se deberán configurar las siguientes variables de entorno:

```
          LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

O bien:

```
          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:
```

En primer lugar hay que crear un Perl que incluya un módulo `DBI` enlazado estáticamente mediante la ejecución de estos comandos en el directorio donde se ubica la distribución de `DBI`:

```
          shell> perl Makefile.PL -static -config
```

```
shell> make
shell> make install
shell> make perl
```

Luego debe instalarse el nuevo Perl. La salida de `make perl` indica exactamente el comando `make` necesario para llevar a cabo la instalación. En SCO, este es `make -f Makefile.aperl inst_perl MAP_TARGET=perl`.

A continuación, emplear el recién creado Perl para crear otro Perl que tmabién incluya un `DBD::mysql` creado estáticamente, mediante la ejecución de estos comandos en el directorio donde se ubica la distribución `DBD::mysql`:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Finalmente, se debería instalar este nuevo Perl. De nuevo, la salida de `make perl` indicará el comando a emplear.

Capítulo 3. Curso (tutorial) de MySQL

Tabla de contenidos

Este capítulo le brinda una introducción de aprendizaje a MySQL, a través del uso de `mysql`, el programa cliente de MySQL para crear y utilizar una base de datos simple. `mysql` (a veces denominado “monitor de terminal” o solamente “monitor”) es un programa interactivo que le permite conectarse a un servidor de bases de datos MySQL, ejecutar consultas, y ver los resultados. `mysql` puede usarse también en modo por lotes: se colocan las consultas en un archivo previamente armado, y se le dice a `mysql` que ejecute el contenido del archivo. En este capítulo se tratan ambas formas de uso.

Para ver una lista de las opciones utilizadas con `mysql`, ejecútelo con la opción `--help`:

```
shell> mysql --help
```

Este capítulo asume que `mysql` está instalado en el ordenador y que está disponible un servidor MySQL al cual conectarse. Si no es así, consulte con su administrador MySQL. (Si **Usted** es el administrador, necesitará consultar otras secciones de este manual).

Se describe el proceso de configurar y utilizar una base de datos. Si sólo le interesa acceder a una base de datos existente, es posible que quiera omitir las secciones que muestran cómo crear una base de datos y las tablas que contiene.

Dado que este capítulo es una guía de aprendizaje, muchos detalles son necesariamente omitidos. Para información detallada sobre los temas que se tratan, consulte las secciones relevantes del manual.

3.1. Conectarse al y desconectarse del servidor

Para conectarse al servidor, generalmente se le porporcionará a `mysql` un nombre de usuario y una contraseña. Si el servidor se está ejecutando en un ordenador distinto a donde está estableciendo la conexión, también se deberá especificar el nombre de host. Consulte con su administrador para saber los parámetros de conexión (nombre de usuario, contraseña y host) que debe emplear. Una vez que conozca los parámetros apropiados, debería poder conectarse de este modo:

```
shell> mysql -h host -u user -p
Enter password: ********
```

`host` y `user` representan el nombre del ordenador donde se está ejecutando el servidor de bases de datos MySQL y el nombre de usuario de la cuenta que se usará para conectarse. Reemplácelos por los valores apropiados para elcaso. Los asteriscos (`********`) representan la contraseña, debe ingresarse cuando `mysql` muestra `Enter password:`.

Si todo funciona bien, se verá una información de ingreso seguida por el prompt `mysql>`:

```
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: 5.0.9-beta-standard

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

mysql>
```

El prompt anuncia que `mysql` está listo para procesar comandos.

Algunas instalaciones de MySQL permiten conectarse como usuario anónimo (sin nombre) si el servidor se está ejecutando en el ordenador local. Si esto sucede en su caso, deberia poder conectarse al servidor ejecutando `mysql` sin ningún parámetro:

```
shell> mysql
```

Después de haberse conectado, puede desconectarse en cualquier momento escribiendo `QUIT` (o `\q`) en el prompt `mysql>`:

```
mysql> QUIT
Bye
```

En Unix, también puede desconectarse presionando Control-D.

La mayoría de los ejemplos en las secciones siguientes asumen que ya se ha conectado al servidor. Por eso muestran el prompt `mysql>`.

3.2. Entrar consultas

Cerciórese de haberse conectado al servidor, tal como se describe en la sección anterior. Esto en sí mismo no selecciona ninguna base de datos para trabajar. En este punto es más importante aprender un poco más acerca de cómo realizar consultas que ir directamente a crear tablas, cargar datos, y recuperarlos. Esta sección describe los principios básicos del ingreso de comandos, empleando varias consultas que puede realizar para familiarizarse con la forma en que funciona `mysql`.

Aquí tiene un comando simple que ordena al servidor que muestre su número de versión y la fecha actual. Ingréselo a continuación del prompt `mysql>` y presione 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>
```

Esta consulta le muestra varias cosas acerca de `mysql`:

- Un comando normalmente consiste en una sentencia SQL seguida de punto y coma. (Hay excepciones donde el punto y coma puede omitirse. `QUIT`, mencionado anteriormente, es una de ellas. Luego conocerá otras.)
- Cuando ingresa un comando, `mysql` lo envía al servidor para ser ejecutado e imprime los resultados. A continuación muestra de nuevo el prompt `mysql>` para informarle que está listo para otro comando.
- `mysql` imprime los resultados de la consulta en forma tabulada (filas y columnas). La primera fila contiene etiquetas para las columnas. Las filas siguientes son los resultados de la consulta. Generalmente, el nombre de cada columna es el nombre del campo que trae desde la base de datos. Si está trayendo el valor de una expresión, en lugar del contenido de un campo o columna de una tabla (como en el ejemplo anterior), `mysql` etiqueta la columna usando el texto de la expresión.
- `mysql` informa cuántas filas fueron devueltas y cuánto tiempo le tomó ejecutarse a la consulta, lo cual da una idea aproximada del rendimiento del servidor. Estos valores son imprecisos porque representan tiempo de reloj corriente (no tiempo de CPU), y además porque están afectados por factores como la carga del servidor o la latencia de red. (Para simplificar los ejemplos de este capitulo, a partir de ahora no se mostrará la línea “rows in set”.)

Las palabras clave pueden ingresarse en cualquier combinación de minúsculas y mayúsculas. Las siguientes consultas son equivalentes:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Aqui tiene otra consulta que demuestra que `mysql` puede usarse como calculadora:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

Las consultas mostradas hasta ahora han sido relativamente cortas, sentencias de una sola línea. Se puede inclusive ingresar múltiples sentencias en una misma línea. Solamente deben separarse con punto y coma:

```
mysql> SELECT VERSION(); SELECT NOW();
+----------------+
| 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)
```

No es necesario que un comando sea ingresado en una sola línea, de ese modo, comandos extensos que requieren varias lineas no son un problema. `mysql` determina cuando una sentencia ha llegado a l final observando si termina en un punto y coma, no si se llegó al final de la línea física. (En otras palabras, `mysql` acepta un formato libre para las entradas: recolecta lineas pero no las ejecuta hasta que encuentra el punto y coma.)

Aqui tiene una sentencia de múltiples líneas:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2005-07-11   |
+---------------+--------------+
1 row in set (0.00 sec)
```

Observe en este ejemplo que el prompt cambia de `mysql>` a `->` después que se ha ingresado la primera línea de una consulta de múltiples líneas. Esta es la forma en que `mysql` advierte que no se ha completado la sentencia y aún espera por el resto. El prompt es un aliado, puesto que suministra información valiosa. Si se emplea, siempre se sabrá lo que `mysql` está esperando.

Si durante el ingreso de un comando decide que no quiere ejecutarlo, cancélelo tipeando `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Una vez más observe el prompt. Cambia a `mysql>` después de que ingresa `\c`, informándole que `mysql` está listo para un nuevo comando.

La siguiente tabla muestra cada uno de los indicadores que podrá ver y sintetiza lo que dicen acerca del estado en que se encuentra `mysql`:

Prompt	Significado
`mysql>`	Listo para un nuevo comando.
`->`	Esperando la siguiente línea en un comando de múltiples líneas.
`'>`	Esperando la siguiente línea, se encuentra abierta una cadena que comienza con apostrofo (`''`).
`">`	Esperando la siguiente línea, se encuentra abierta una cadena que comienza con comillas dobles (`""`).
`` `> ``	Esperando la siguiente línea, se encuentra abierta una cadena que comienza con tilde (`` `` ``).
`/*>`	Esperando la siguiente línea, se encuentra abierto un comentario que comienza con `/*`.

El prompt `/*>` fue introducido en la serie 5.0 a partir de MySQL 5.0.6.

Es frecuente que se origine una sentencia de múltiples lineas cuando accidentalmente le da entrada a un comando de una sola línea pero olvida terminarlo con punto y coma. En ese caso, `mysql` aguarda por más caracteres:

```
mysql> SELECT USER()
    ->
```

Si esto le ocurre (considera que ha ingresado una sentencia completa pero solamente obtiene un prompt `->`), la mayoría de las veces es porque `mysql` está esperando por el punto y coma. Si no advierte lo que el indicador trata de decirle, podría demorar un buen tiempo en hacer lo que necesita. Ingrese un punto y coma para completar la sentencia, y `mysql` la ejecutará:

```
mysql> SELECT USER()
    -> ;
+--------------------+
| USER()             |
+--------------------+
| joesmith@localhost |
+--------------------+
```

Los prompts `'>` y `">` aparecen durante el ingreso de cadenas. Puede escribir cadenas delimitadas por '' o "" (por ejemplo, `'hola'` o `"adios"`), y `mysql` le permite ingresar cadenas divididas en múltiples líneas. Cuando ve un prompt `'>` o `">` significa que ha comenzado a ingresar una cadena comenzando con '' o "" pero no ha ingresado el correspondiente caracter de terminación. A menudo esto significa que inadvertidamente omitió este carácter. Por ejemplo:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

Si ingresa esta sentencia `SELECT`, presiona ENTER y espera por el resultado, nada ocurrirá. En lugar de asombrarse por el tiempo que consume la consulta, note lo que el prompt `'>` le está diciendo. Indica que `mysql` espera por el final de una cadena inconclusa. (¿Ve el error en la sentencia? La cadena `'Smith` no tiene el apóstrofo de cierre.)

¿Qué hacer llegado a este punto? Lo más simple es cancelar el comando. No obstante, no puede simplemente teclear `\c` en este caso, porque `mysql` interpretaría que es parte de la cadena que está introduciendo. En lugar de eso, teclee el carácter de cierre que falta y entonces escriba \c.>:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

El prompt cambia de nuevo a `mysql>`, informando que `mysql` está listo para un nuevo comando.

El prompt `` `> `` es similar a `'>` y `">` , pero informa que está pendiente de completar un identificador delimitado por tildes.

Es importante conocer el significado de estos indicadores, ya que si por error se ingresa una cadena incompleta, todo lo que se ingrese posteriormente será aparentemente ignorado por `mysql` — incluyendo el comando `QUIT`. Esto puede ser sumamente desconcertante, en particular si no se conoce lo que debe hacer para terminar la línea y cancelar el comando.

3.3. Crear y utilizar una base de datos

Una vez que se sabe la forma de ingresar comandos, es el momento de acceder a una base de datos.

Suponga que en su hogar posee varias mascotas y desea registrar distintos tipos de información sobre ellas. Puede hacerlo si crea tablas para almacenar sus datos e introduce en ellas la información deseada. Entonces, podrá responder una variedad de preguntas acerca de sus mascotas recuperando datos desde las tablas. Esta sección le muestra como:

- Crear una base de datos
- Crear una tabla
- Introducir datos en la tabla
- Recuperar datos desde la tabla de varias maneras
- Emplear múltiples tablas

La base de datos menagerie (palabra inglesa que en español significa "colección de animales") se ha hecho deliberadamente simple, pero no es difícil imaginar situaciones del mundo real donde podría usarse un tipo similar de base de datos. Por ejemplo, para un granjero que desee hacer el seguimiento de su hacienda, o para los registros de los pacientes de un veterinario. En el sitio web de MySQL pueden descargarse archivos de texto con datos de ejemplo y algunas de las sentencias empleadas en las siguientes secciones. Se encuentran disponibles en formato `tar` (http://downloads.mysql.com/docs/menagerie.tar.gz) y Zip (http://downloads.mysql.com/docs/menagerie.zip).

Mediante la sentencia `SHOW` se encuentran las bases de datos que existen actualmente en el servidor:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

Probablemente la lista obtenida sea distinta en su ordenador, pero es casi seguro que tendrá las bases de datos `mysql` y `test`. La base de datos `mysql` es necesaria porque es la que describe los privilegios de acceso de los usuarios. La base de datos `test` se provee para que los usuarios hagan pruebas.

Tenga en cuenta que si no tiene el privilegio `SHOW DATABASES`, no podrá ver todas las bases de datos que hay en el servidor. Consulte Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”.

Si la base de datos `test` existe, intente acceder a ella:

```
mysql> USE test
Database changed
```

Advierta que, al igual que `QUIT`, `USE` no necesita que ponga un punto y coma al final (aunque puede hacerlo si lo desea). La sentencia `USE` tiene otra particularidad: debe escribirse en una sola linea.

Puede colocar los ejemplos siguientes en la base de datos `test`, si tiene acceso a ella, pero si trabaja en un ambiente compartido, lo que deposite allí puede ser fácilmente borrado por alguien más que tenga el acceso. Por este motivo, debería pedirle a su administrador permiso para usar una base de datos propia. Suponga que quiere llamarla `menagerie`. El administrador necesitará ejecutar un comando como este:

```
```

```
mysql> GRANT ALL ON menagerie.* TO 'su_nombre_mysql'@'su_host_cliente';
```

Donde `su_nombre_mysql` es el nombre de usuario que se le asignó, y `su_host_cliente` es el host u ordenador desde donde se conectará.

3.3.1. Crear y seleccionar una base de datos

Si el administrador crea su base de datos en el mismo momento que le otorga privilegios, puede comenzar a utilizarla, de lo contrario necesitará crearla:

```
mysql> CREATE DATABASE menagerie;
```

En ambientes Unix, los nombres de las bases de datos son case sensitive (al contrario que las palabras clave), de modo que siempre debe referirse a su base de datos como `menagerie`, y no `Menagerie`, `MENAGERIE`, o una variante similar. Esto también se aplica a los nombres de tablas. Esta restricción no existe en Windows, aunque puede utilizar el mismo esquema de mayúsculas cuando se refiera a bases de datos y tablas en una consulta dada.

Al crear una base de datos, ésta no se selecciona para su uso, debe hacerlo explicitamente. Para convertir a `menagerie` en la base de datos actual, use este comando:

```
mysql> USE menagerie
Database changed
```

Las bases de datos sólo necesitan ser creadas una sola vez, pero deben ser seleccionadas cada vez que se inicia una sesión de `mysql`. Puede hacerse a través del comando `USE` como se muestra en el ejemplo, o puede indicar la base de datos en la linea de comandos al ejecutar `mysql`. Simplemente debe indicar el nombre de la base de datos a continuación de los parámetros que necesite ingresar. Por ejemplo:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Advierta en el comando anterior que `menagerie` **no** es la contraseña. Si se quisiera suministrar la contraseña en la linea de comandos, después de la opción `-p`, debe hacerse sin dejar espacios en blanco (por ejemplo, `-pmypassword`, no `-p mypassword`). De todos modos, colocar la contraseña en la linea de comandos no es recomendable porque lo expone a la vista de otros usuarios.

3.3.2. Crear una tabla

La creación de la base de datos ha sido una tarea sencilla, pero hasta ahora permanece vacía, como le muestra `SHOW TABLES`:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

La parte difícil es decidir cómo debería ser la estructura de su base de datos: qué tablas necesitará, y qué columnas habrá en cada tabla.

Querrá una tabla para contener un registro por cada mascota. Esta tabla puede llamarse `pet`, y debería contener, como mínimo, el nombre de cada animal. Dado que el nombre no es muy relevante por sí mismo, tendría que tener más información. Por ejemplo, si más de una persona en su familia tendrá mascotas, querrá listar también el dueño de cada animal. Y algunos otros datos descriptivos básicos, como especie y sexo.

¿Qué hacer con la edad? Podría ser de interés, pero no es un buen dato para almacenar en una base de datos. La edad cambia a medida que pasa el tiempo, lo cual significa que debería actualizar la base de datos a menudo. En lugar de esto, es mejor almacenar un valor fijo, como la fecha de nacimiento. De este modo, cada vez que requiera saber la edad, podrá calcularla como la diferencia entre la fecha de nacimiento y la fecha actual. MySQL provee funciones para realizar cálculos con fechas, por lo que no es dificultoso. Almacenar la fecha de nacimiento en lugar de la edad tiene otras ventajas:

- Puede usar la base de datos para tareas como generar recordatorios para los próximos cumpleaños de mascotas. (Si piensa que este tipo de consultas no es importante, considere que es lo mismo que haría en un contexto de base de datos de negocios para identificar aquellos clientes a los que habrá que enviar una tarjeta por su cumpleaños, para conseguir ese toque personal con la asistencia del ordenador).
- Puede calcular edades en relación a otras fechas además de la actual. Por ejemplo, almacenar la fecha de muerte de una mascota le posibilita calcular la edad que tenía a ese momento.

Probablemente pensará en otros tipos de información que resultarían útiles dentro de la tabla `pet` pero los identificados hasta ahora son suficientes: name (nombre), owner (propietario), species (especie), sex (sexo), birth (nacimiento) y death (muerte).

Debe usar la sentencia `CREATE TABLE` para especificar la estructura de una tabla:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

El tipo de dato `VARCHAR` es una buena elección para las columnas `name`, `owner`, y `species` porque los datos que allí se almacenan no son de longitud uniforme. En realidad no es necesario que todas estas columnas tengan la misma longitud ni que ésta sea `20`. En MySQL 5.0.3 y versiones posteriores, normalmente se puede adoptar cualquier longitud entre `1` y `65535`, según lo que se crea más razonable. (**Nota**: Anteriormente a MySQL 5.0.3, el límite de longitud era 255.) Si en el futuro debiera aumentar la longitud de estos campos, MySQL tiene la sentencia `ALTER TABLE`.

Hay varios tipos de datos que podrían usarse para representar el sexo en los registros de animales, tal como `'m'` y `'f'`, o `'male'` (masculino) y `'female'` (femenino). Lo más simple es usar los caracteres `'m'` y `'f'`.

Es obvio el uso del tipo de dato `DATE` para las columnas `birth` y `death`.

Luego de crear una tabla, `SHOW TABLES` debería producir una salida:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

Para verificar que la tabla ha sido creada en la forma esperada, utilice la sentencia `DESCRIBE`:

```
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    |       |
+---------+-------------+------+-----+---------+-------+
```

`DESCRIBE` puede ser utilizada en cualquier momento, por ejemplo, si olvida los nombres o el tipo de dato de las columnas de la tabla.

3.3.3. Cargar datos en una tabla

Luego de crear la tabla, necesitará completarla con datos. Para esto, le serán de utilidad las sentencias `LOAD DATA` e `INSERT`.

Suponga que los registros de mascotas fueran como los mostrados a continuación. (Observe que MySQL espera que las fechas tengan el formato `'AAAA-MM-DD'`, esto puede ser diferente a lo que acostumbra utilizar).

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	

Dado que está comenzando con una tabla vacía, una forma fácil de completarla es creando un fichero de texto que contenga una línea por cada animal, y luego insertando el contenido del fichero en la tabla mediante una sola sentencia.

Para esto, debería crear un fichero de texto llamado `pet.txt`, conteniendo un registro por linea, con cada valor separado por un carácter de tabulación, y dispuestos en el orden en el cual se especificaron las columnas en la sentencia `CREATE TABLE`. Para valores ausentes (como sexo desconocido o fechas de muerte de animales con vida), puede usar valores `NULL`. Para representar estos valores en el archivo de texto, utilice `\N` (barra diagonal y N mayúscula). Por ejemplo, el registro de Whistler se vería del modo siguiente (el espacio en blanco entre cada valor es un solo carácter de tabulación):

name	owner	species	sex	birth	death
`Whistler`	`Gwen`	`bird`	`\N`	`1997-12-09`	`\N`

Para cargar el fichero `pet.txt` dentro de la tabla `pet`, utilice este comando:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

Si trabaja en Windows, con un editor que emplea `\r\n` (retorno de carro + nueva linea) como caracteres de fin de línea, debería usar:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

(En un ordenador Apple bajo OS X, probablemente quiera utilizar `LINES TERMINATED BY '\r'`.)

Opcionalmente puede especificar en la sentencia `LOAD DATA` los caracteres que actuarán como separador de campo y fin de línea, pero los valores por defecto son tabulación y nueva línea. Estos son suficientes para que la sentencia lea correctamente el fichero `pet.txt`

Si ocurre un error al ejecutar la sentencia, probablemente se deba a que su instalación de MySQL no tiene habilitada por defecto la capacidad de manejar archivos locales. Consulte Sección 5.5.4, “Cuestiones relacionadas con la seguridad y `LOAD DATA LOCAL`” para obtener información sobre cómo cambiar esto.

Cuando lo que desea es agregar nuevos registros de a uno por vez, la sentencia `INSERT` resulta de utilidad. De esta sencilla manera, se suministran valores para cada columna, dispuestos en el orden en el cual se especificaron las columnas en la sentencia `CREATE TABLE` statement. Suponga que Diane obtiene un nuevo hamster llamado "Puffball". Se podría agregar un nuevo registro, usando la sentencia `INSERT` de este modo:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Observe que las cadenas alfanuméricas y las fechas son representados como cadenas delimitadas por apóstrofos. También, con `INSERT`, se pueden insertar valores `NULL` directamente, para indicar un valor ausente. No se debe utilizar `\N` como se hace con `LOAD DATA`.

A partir de este ejemplo queda demostrado que lleva mucho más trabajo realizar una carga inicial de registros empleando varias sentencias `INSERT` que si se hace mediante la sentencia `LOAD DATA`.

3.3.4. Extraer información de una tabla

La sentencia `SELECT` es utilizada para traer información desde una tabla. La sintaxis general de esta sentencia es:

```
SELECT seleccionar_Esto
FROM desde_tabla
WHERE condiciones;
```

`seleccionar_esto` es lo que se quiere ver. Puede ser una lista de columnas, o `*` para indicar “todas las columnas.” *`desde_tabla`*indica la tabla donde están los datos a recuperar. La cláusula `WHERE` es opcional. Si está presente, *`condiciones`* representa las condiciones que cada registro debe cumplir para retornar como resultado.

3.3.4.1. Seleccionar todos los datos

La forma más simple de `SELECT` recupera todo lo que hay en la tabla:

```
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       |
```

```
+----------+--------+---------+------+------------+------------+
```

Esta forma de `SELECT` es útil si se quiere revisar la tabla completa, por ejemplo, despues de haberla cargado con un conjunto de datos inicial. Por ejemplo, puede ocurrir que la fecha de nacimiento de Bowser no parezca correcta. Consultando los papeles de pedigri, se descubre que el año correcto de nacimiento es 1989, no 1979.

Existen al menos dos formas de solucionarlo:

- Editando el fichero `pet.txt` para corregir el error, vaciando la tabla y volviendola a llenar con los datos. Para esto se usan las sentencias `DELETE` y `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  No obstante, si opta por esto, deberá volver a cargar el registro de Puffball.

- Corrigiendo únicamente el registro erróneo. Para esto se usa la sentencia `UPDATE`:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  `UPDATE` modifica solo el registro en cuestión y no requiere que se vuelva a llenar la tabla.

3.3.4.2. Seleccionar registros específicos

Como se ha visto en la sección anterior, es fácil recuperar una tabla en su totalidad. Sólo debe omitir la cláusula `WHERE` en la sentencia `SELECT`. Pero, generalmente, no se desea ver la tabla completa, especialmente cuando alcanza un gran tamaño. En cambio, usualmente, se tiene interés en obtener una respuesta para una consulta en particular, en cuyo caso se especifican algunas restricciones para la información que se traerá. A continuación se verán algunas consultas que responden preguntas acerca de las mascotas.

Se pueden seleccionar sólo algunos registros de la tabla. Por ejemplo, si quisiera verificar los cambios realizados sobre la fecha de nacimiento de Bowser, seleccione el registro de Bowser de esta manera:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

La salida confirma que el año fue correctamente registrado como 1989, ya no es 1979.

Normalmente, las comparaciones de cadenas no son case sensitive, por eso puede escribir el nombre como `'bowser'`, `'BOWSER'`, etc. El resultado de la consulta será el mismo.

Se pueden indicar condiciones a cumplir por cualquier columna, no solamente por `name`. Por ejemplo, si quisiera saber qué animales han nacido luego de 1998, necesita evaluar la columna `birth`:

```
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  |
```

```
+----------+-------+---------+------+------------+-------+
```

Se pueden combinar condiciones, por ejemplo para localizar perros hembra:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

La consulta anterior emplea el operador lógico `AND`. También existe el operador `OR`:

```
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` pueden ser combinadas, si bien `AND` tiene mayor precedencia que `OR`. Si utiliza ambos operadores, es buena idea emplear paréntesis para indicar explicitamente la forma en que las condiciones deben agruparse:

```
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. Seleccionar columnas concretas

Si no se quieren ver filas completas, solo hace falta indicar las columnas en las que se está interesado, separadas por comas. Por ejemplo, si desea saber cuándo nació cada animal, seleccione las columnas `name` y `birth`:

```
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 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

Para saber quien posee mascotas, utilice esta consulta:

```
mysql> SELECT owner FROM pet;
```

```
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

Observe que esta sentencia retorna el campo `owner` de cada registro, y algunos de ellos aparecen más de una vez. Para reducir la salida, recupere solamente una vez cada registro repetido, agregando la palabra clave `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

Puede emplearse una clásula `WHERE` para combinar la selección de ciertas filas y de ciertas columnas. Por ejemplo, para obtener únicamente la fecha de nacimiento de perros y gatos, ejecute esta consulta:

```
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. Ordenar registros

Quizá advirtió, en los ejemplos anteriores, que las filas resultantes se mostraron sin ningún orden en particular. A menudo es más fácil examinar la salida de una consulta cuando las filas se ordenan de algún modo significativo. Para ordenar un resultado, se usa la clásula `ORDER BY`.

Aqui tiene las fechas de cumpleaños de los animales, ordenadas por fecha:

```
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 |
+----------+------------+
```

Por lo general, cuando se trata de columnas de tipo carácter, la ordenación, — al igual que otras operaciones de comparación — no es case-sensitive. Significa que el orden permanece indefinido para las columnas que son idénticas excepto por sus mayúsculas y minúsculas. Puede no obstante forzar a que una columna se ordene en forma sensible a mayúsculas empleando el modificador `BINARY`: `ORDER BY BINARY columna`.

El sentido de ordenación, por defecto, es ascendente, con los valores más pequeños primero. Para ordenar en sentido inverso (descendente), agregue la palabra clave `DESC` luego del nombre de la columna por la que ordena:

```
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 |
+----------+------------+
```

Puede ordenar basándose en varias columnas, y cada columna en un sentido diferente. Por ejemplo, para ordenar por tipo de animal en sentido ascendente y, dentro de cada tipo, ordenar por nacimiento en sentido descendente (los animales más jóvenes primero) utilice la siguiente consulta:

```
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 |
+----------+---------+------------+
```

Advierta que la palabra clave `DESC` se aplica sobre la columna inmediatamente anterior (`birth`); no afecta el sentido de ordenación de la columna `species`.

3.3.4.5. Cálculos sobre fechas

MySQL provee varias funciones que se aplican a cálculos entre fechas, por ejemplo, para calcular edades u obtener partes de una fecha.

Para determinar cuántos años de edad tiene cada mascota, hay que calcular la diferencia entre el año de la fecha actual y el de la fecha de nacimiento, y luego restar 1 al resultado si el dia y mes actuales son

anteriores al día y mes indicados por la fecha de nacimiento. La siguiente consulta devuelve, para cada mascota, el nombre, la fecha de nacimiento, la fecha actual, y la edad en años.

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> 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 |
+----------+------------+------------+------+
```

En el ejemplo anterior, `YEAR()` trae la parte correspondiente al año de una fecha, y `RIGHT()` trae los 5 primeros caracteres contando desde la derecha, que representan la parte `MM-DD` de la fecha. La porción de la expresión que compara los valores `MM-DD` devuelve 1 o 0, lo cual se corresponde con la diferencia de 1 año a restar de la edad si el dia de la fecha devuelto por `CURDATE()` ocurre antes que la fecha de nacimiento `birth`. La expresión completa es un tanto confusa para usar como encabezado, por lo que se emplea un *alias* (`age`) para que el encabezado sea más comprensible.

La consulta funciona bien, pero los resultados podrían revisarse más fácilmente si las filas se presentaran en algún orden. Esto puede hacerse agregando la cláusula `ORDER BY name` para ordenar por nombre la salida:

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> 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 |
+----------+------------+------------+------+
```

Para ordenar la salida por edad (`age`) en lugar de por nombre (`name`), solo hay que utilizar una cláusula `ORDER BY` diferente:

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> 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 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

Una consulta similar se utiliza para determinar la edad a la fecha de muerte de los animales. Se determinan los animales que han muerto verificando si el valor de la columna `death` es `NULL`. Entonces, para todos los valores no `NULL` calcula la diferencia entre las fechas de muerte (`death`) y nacimiento (`birth`):

```
mysql> SELECT name, birth, death,
    -> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

La consulta utiliza la expresión `death IS NOT NULL` en lugar de `death <> NULL` porque `NULL` es un valor especial, que no puede ser comparado mediante los operadores lógicos habituales. Este tema se trata más extensamente más adelante. Consultar Sección 3.3.4.6, “Trabajar con valores NULL”.

¿Qué tal si se quisiera saber qué animales cumplen años el próximo mes? Para esta clase de cálculos, el año y el día son irrelevantes; simplemente se desea extraer de la columna `birth` la parte correspondiente al mes. MySQL cuenta con varias funciones para extraer partes de fechas, como `YEAR()`, `MONTH()`, y `DAYOFMONTH()`. `MONTH()` es la función apropiada para este caso. Para verla en funcionamiento, ejecute una consulta que muestra tanto el valor de `birth` como el de `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 |
+----------+------------+--------------+
```

Encontrar los animales que cumplen años el mes siguiente es también sencillo. Suponga que el mes actual es abril. De modo que su número es `4`, y se buscan los animales nacidos en Mayo (mes `5`), de esta forma:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
```

```
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

Esto se complica ligeramente cuando el mes actual es Diciembre. No se puede simplemente sumarle 1 al número del mes (`12`) y buscar animales nacidos en el mes `13`, porque no existe tal mes. En lugar de eso, se debe buscar por animales nacidos en Enero (mes `1`).

Se puede incluso escribir la consulta de forma que funcione sin importar cual es el mes actual. Así, no se necesitará indicar un mes en particular en la consulta. `DATE_ADD()` sirve para sumar un intervalo de tiempo a una fecha dada. Si se adiciona un mes al valor de `CURDATE()`, y se extrae el mes mediante `MONTH()`, el resultado será el mes en el que se buscarán cumpleaños:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

Una manera alternativa de alcanzar el mismo resultado es sumar `1` al mes actual para obtener el mes siguiente (después de emplear la función módulo (`MOD`) para dejar el número de mes en `0` si resultara ser `12`:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

Advierta que `MONTH` devuelve un número entre `1` y `12`. Y `MOD(algun_valor,12)` devuelve un número entre `0` y `11`. La suma debe ser realizada después de `MOD()`, en otro caso se estaría pasando de Noviembre (`11`) a Enero (`1`).

3.3.4.6. Trabajar con valores `NULL`

El valor `NULL` puede resultar un poco desconcertante hasta que se comienza a utilizar. Conceptualmente, `NULL` significa valor inexistente o desconocido, y es tratado de forma diferente a otros valores. Para verificar que un valor es `NULL`, no se pueden emplear operadores de comparación aritmética como `=`, `<`, o `<>`. Para comprobar esto, intente la siguiente consulta:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Claramente, no se obtienen valores significtivos a partir de estas comparaciones. Use en su lugar los operadores `IS NULL` y `IS NOT NULL`:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

Observe que en MySQL, `0` o `NULL` se intepretan como falso, y cualquier otro valor, como verdadero. El valor por defecto para una operación booleana es `1`.

Este tratamiento especial de `NULL` es debido a que, en la sección anterior, fue necesario determinar qué animales ya no estaban vivos utilizando `death IS NOT NULL` en lugar de `death <> NULL`.

Dos valores `NULL` son considerados iguales por la cláusula `GROUP BY`.

Cuando se realiza un `ORDER BY`, los valores `NULL` se presentan en primer lugar si se emplea `ORDER BY ... ASC`, y al final si se ordena con `ORDER BY ... DESC`.

Un error muy común cuando se trabaja con valores `NULL` es asumir que es imposible insertar un valor cero o una cadena vacía en una columna definida como `NOT NULL`, pero no es así. Los mencionados son efectivamente valores, mientras que `NULL` significa "no hay un valor". Puede comprobar esto fácilmente empleando `IS [NOT] NULL` como se muestra aquí:

```
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 |
+-----------+---------------+------------+----------------+
```

Por lo tanto, es totalmente posible insertar cadenas vacias o ceros en columnas marcadas como `NOT NULL`, ya que son valores `NOT NULL`. Consultar Sección A.5.3, “Problemas con valores NULL”.

3.3.4.7. Coincidencia de patrones

MySQL posee capacidades estándar para utilizar patrones así como también una forma de patrones basada en expresiones regulares extendidas similares a las que se encuentran en utilidades de UNIX, como ser `vi`, `grep`, y `sed`.

Los patrones SQL permiten emplear el carácter '`_`' para representar coincidencia con un carácter individual y '`%`' En MySQL, por defecto, los patrones SQL no son case-sensitive. Abajo se muestran algunos ejemplos. Advierta que no se emplean los operadores `=` o `<>` para trabajar con patrones SQL, en lugar de eso se usan los operadores de comparación `LIKE` o `NOT LIKE`.

Para encontrar nombres que comiencen con '`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 |
+--------+--------+---------+------+------------+------------+
```

Para encontrar nombres que terminen con '`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  |
+--------+--------+---------+------+------------+-------+
```

Para encontrar nombres que contengan '`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       |
+----------+-------+---------+------+------------+------------+
```

Para encontrar nombres que contengan exactamente 5 caracteres, use 5 veces el carácter patrón '_':

```
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  |
+-------+--------+---------+------+------------+-------+
```

Los otros patrones que pueden emplearse con MySQL usan expresiones regulares extendidas. Cuando busque coincidencias con este tipo de patrones, use los operadores `REGEXP` y `NOT REGEXP` (o bien los sinónimos `RLIKE` y `NOT RLIKE`).

Algunas características de las expresiones regulares extendidas:

- '`.`' detecta coincidencia con cualquier carácter individual.
- Una clase de carácter '`[...]`' detecta coincidencia con cualquier carácter entre los corchetes. Por ejemplo, '`[abc]`' coincidirá con 'a', 'b', o 'c'. Para hacer referencia a un rango de caracteres, use un guión. '`[a-z]`' detecta coincidencia con cualquier letra, mientras que '`[0-9]`' lo hace con cualquier dígito.
- '`*`' detecta coincidencia con cero o más apariciones de los caracteres que lo preceden. Por ejemplo, '`x*`' detecta cualquier número de caracteres 'x', '`[0-9]*`' detecta cualquier cantidad de dígitos, y '`.*`' coincidirá con cualquier número de cualquier carácter.
- `REGEXP` tendrá éxito si el patrón suministrado encuentra coincidencia en cualquier parte del valor examinado (esto difiere de `LIKE` en que este último solo tiene éxito si el patrón concuerda con todo el valor).
- Para lograr que un patrón detecte coincidencias solamente al principio o al final del valor examinado, utilice '`^`' al principio o '`$`' al final del patrón.

Para demostrar el funcionamiento de las expresiones regulares extendidas, las consultas con `LIKE` expuestas anteriormente se han reescrito utilizando `REGEXP`.

Para hallar nombres que comiencen con 'b', use '`^`' para buscar coincidencia al principio del valor:

```
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 |
+--------+--------+---------+------+------------+------------+
```

En MySQL 5.0, si realmente quiere forzar a que la comparación realizada por `REGEXP` sea case sensitive, utilice la palabra clave `BINARY` para convertir a una de las cadenas en una cadena binaria. Esta consulta solamente encontrará coincidencia con 'b' minúsculas al comienzo de un nombre:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

Para hallar nombres finalizados en '`fy`', emplee '`$`' para buscar la coincidencia en el final del nombre:

```
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  |
+--------+--------+---------+------+------------+-------+
```

Para encontrar nombres conteniendo una '`w`', utilice esta consulta:

```
mysql> SELECT * FROM pet WHERE name REGEXP '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       |
+----------+-------+---------+------+------------+------------+
```

Debido a que un patrón de expresión regular encuentra coincidencia sin importar el lugar del valor donde se produce, en la consulta previa no es necesario colocar un comodín a cada lado del patrón para obtener coincidencia en cualquier parte del valor, como hubiera sucedido de utilizar un patrón SQL

Para hallar nombres conteniendo exactamente cinco caracteres, use '`^`' y '`$`' para obligar a que la coincidencia deba estar al principio y al final del nombre, y cinco instancias de '`.`' entre ellas.

```
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  |
+-------+--------+---------+------+------------+-------+
```

La consulta anterior también se podría haber escrito empleando el operador '`{n}`' “repetir-`n`-veces”:

```
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  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.8. Contar registros

Una pregunta frecuente que deben responder las bases de datos es: “¿qué tan a menudo aparece en la tabla un cierto tipo de dato?” Por ejemplo, se podría querer averiguar la cantidad de mascotas de que se dispone, o cuantas mascotas tiene cada propietario, o varios otros recuentos sobre los animales.

Contar la cantidad total de animales es la misma pregunta que “¿cuántos registros hay en la tabla `pet`?”, ya que hay un registro por mascota. `COUNT(*)` cuenta el número de filas, por ello, la consulta para contar animales luce así:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
```

```
|        9 |
+----------+
```

Anteriormente se recuperaban los nombres de la gente que poseía mascotas. Se puede usar `COUNT()` para hallar cuantas mascotas tiene cada propietario:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

Observe el uso de `GROUP BY` para agrupar todos los registros de cada propietario. Sin dicha cláusula, todo lo que se hubiera obtenido sería un mensaje de error:

```
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
```

`COUNT()` y `GROUP BY` son útiles para presentar datos en varias formas. Los siguientes ejemplos muestran diferentes operaciones:

Cantidad de animales por especies:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Cantidad de animales por sexo:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(En esta salida, `NULL` indica "sexo desconocido")

Cantidad de animales por combinación de especies y sexo:

```
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 |
+---------+------+----------+
```

No es necesario examinar una tabla entera cuando se emplea `COUNT()`. Por ejemplo, la consulta anterior, se podria limitar a perros y gatos de esta manera:

```
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 |
| dog     | m    |        2 |
+---------+------+----------+
```

O si desea la cantidad de animales de cada sexo contando solamente los que tienen sexo conocido:

```
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 |
+---------+------+----------+
```

3.3.4.9. Utilizar más de una tabla

La tabla `pet` mantiene el registro de las mascotas que se poseen. Si quisiera registrar otros datos acerca de ellas, como eventos de su vida tales como visitas al veterinario o nacimiento de crías, necesitaría otra tabla. ¿Cómo debería ser esta tabla? Se necesita:

- Un campo con el nombre de la mascota para saber a quien pertenece cada evento registrado.
- La fecha en que ocurrió el evento.
- Un campo con la descripción del evento.
- Un campo con el tipo de evento, a fin de poder clasificarlo.

Teniendo en cuenta estas consideraciones, la sentencia `CREATE TABLE` para la tabla `event` ("eventos", en inglés) podría ser así:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

Como se hizo con la tabla `pet`, es más fácil realizar la carga inicial de datos si se crea un archivo de texto delimitado con tabulaciones que contenga la información a agregar:

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

Los registros se cargan así:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Con base en lo que se ha aprendido a partir de las consultas efectuadas sobre la tabla `pet`, se debería poder recuperar registros de la tabla `event`; los principios son los mismos. Pero en un momento dado la tabla `event` por sí sola es insuficiente para responder las preguntas que pueden formularse.

Suponga que se desea saber a qué edad tuvo sus crías cada mascota. Anteriormente se aprendió a calcular edades a partir de dos fechas. La fecha en que la mascota tuvo sus crias está en la tabla `event`, pero para calcular su edad, se necesita su fecha de nacimiento, la cual está localizada en la tabla `pet`. Esto significa que la consulta requiere ambas tablas:

```
mysql> SELECT pet.name,
    -> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
    -> remark
    -> FROM pet, event
    -> WHERE pet.name = event.name AND 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         |
+--------+------+-----------------------------+
```

Hay varias cosas para observar en esta consulta:

- La cláusula `FROM` menciona dos tablas porque la consulta necesita traer datos de ambas
- Cuando se combina (también se denomina join -unión, en inglés-) información desde múltiples tablas, se necesita indicar qué registro de una tabla se combinará con qué registro de la otra. Esto es sencillo porque ambas tablas tienen una columna `name`. La consulta emplea la cláusula `WHERE` para hacer coincidir registros de las dos tablas basándose en el valor de `name`.
- Dado que la columna `name` aparece en ambas tablas, se debe especificar a cuál tabla pertenece la columna al hacer referencia a ella. Esto se hace anteponiendo el nombre de la tabla al nombre de la columna.

No es necesario tener dos tablas diferentes para establecer una unión. A veces es útil combinar una tabla consigo misma, si se desea comparar entre sí registros de una misma tabla. Por ejemplo, para formar parejas de mascotas para reproducción, podría unir la tabla `pet` consigo misma para generar pares de animales macho y hembra de la misma especie:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1, pet AS p2
    -> WHERE 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     |
+--------+------+--------+------+---------+
```

En la consulta anterior se especificaron alias para la tabla con el fin de indicar a qué instancia de la tabla pertenece cada columna referenciada.

3.4. Obtener información sobre bases de datos y tablas

¿Qué tal si no se recuerda el nombre de una base de datos o una tabla, o cómo es su estructura (por ejemplo, nombres de columnas)? MySQL aborda este problema a través de varias sentencias que proveen información acerca de las bases de datos y tablas que soporta.

Ya se ha visto `SHOW DATABASES`, la cual informa las bases de datos gestionadas por el servidor. Para conocer la base de datos actualmente seleccionada, se utiliza la funcion `DATABASE()`:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

Si aún no se hubiese seleccionado ninguna base de datos, el resultado sería `NULL`.

Para conocer las tablas contenidas en la base de datos actual (por ejemplo, si no se está seguro del nombre de una tabla) se usa el siguiente comando:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

Si lo que se desea es ver la estructura de una tabla, el comando `DESCRIBE` es útil; muestra información acerca de cada columna de la tabla:

```
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` contiene el nombre de la columna, `Type` es el tipo de dato, `NULL` señala si la columna puede contener valores `NULL`, `Key` indica si la columna está indexada, y `Default` informa el valor por defecto de la columna.

Si una tabla tiene índices, `SHOW INDEX FROM tbl_name` muestra información sobre ellos.

3.5. Usar `mysql` en modo batch

En las secciones previas `mysql` se utilizó interactivamente para ejecutar consultas y ver resultados. También se puede ejecutar en un modo por lotes. Para ello, los comandos que se desea ejecutar deben colocarse en un archivo, y posteriormente indicarle a `mysql` que acepte como entrada el contenido del mismo.

```
shell> mysql < batch-file
```

Si está ejecutando `mysql` en Windows y el archivo contiene algunos caracteres que causan problemas, el comando es así:

```
C:\> mysql -e "source batch-file"
```

Si se necesitara incluir parámetros de conexión en la línea de comandos, el comando podría verse así:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

Cuando se procede de este modo lo que se está haciendo es crear un archivo script, que luego es ejecutado.

Si se desea que el script continúe su ejecución aunque alguna de sus sentencias produzca errores, se debe usar la opción de línea de comandos `--force`.

¿Por qué usar scripts? Algunas razones:

- Si se utiliza una consulta repetidamente (por ejemplo cada día o cada semana), hacer un script evitará volver a teclearla cada vez que se desea ejecutarla.
- Se pueden crear nuevas consultas a partir de otras existentes que se le parezcan, copiando y editando el archivo de script.
- El modo por lotes también puede ser útil cuando se está creando una consulta, en especial si tiene comandos de múltiples líneas o múltiples sentencias. Si se comete un error, no se necesita reteclearlo todo, sino sólo editar el script para corregir el error, y volver a ejecutarlo mediante `mysql`.
- Si se ejecuta una consulta que produce una salida muy extensa, se puede ejecutar a traves de un paginador en lugar de verla desaparecer rápidamente por la parte superior de la pantalla:

  ```
  shell> mysql < batch-file | more
  ```

- Se puede enviar la salida a un archivo, para posterior proceso:

  ```
  shell> mysql < batch-file > mysql.out
  ```

- Se puede distribuir el script a otras personas, para que puedan tambien ejecutar los comandos.
- Algunas situaciones no permiten la interactividad, por ejemplo, cuando se ejecuta una consulta a través de una tarea de `cron` (en Unix). En este caso, debe emplearse el modo por lotes.

El formato de salida es más breve cuando se usa modo por lotes que cuando se utiliza `mysql` interactivamente. Por ejemplo, la salida devuelta para `SELECT DISTINCT species FROM pet` se ve así cuando se ejecuta en modo interactivo:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

Mientras que, en modo por lotes, presenta este aspecto:

```
species
bird
cat
dog
hamster
snake
```

Si desea obtener el formato por lotes para una salida producida interactivamente, utilice `mysql -t`. Para incluir en la salida los comandos que se ejecutan, utilice `mysql -vvv`.

También pueden ejecutarse archivos de script desde el prompt `mysql` utilizando los comandos `source` o `\.`

```
mysql> source filename;
mysql> \. filename
```

3.6. Ejemplos de consultas comunes

Aquí tiene ejemplos de como resolver algunos problemas comunes mediante MySQL.

Algunos de los ejemplos emplean la tabla `shop` para contener el precio de cada artículo (número de item) para ciertos distribuidores (dealers). Suponiendo que cada distribuidor tiene un único precio fijo por cada artículo, entonces (`article`, `dealer`) es una clave primaria para los registros.

Inicie la utilidad de línea de comandos `mysql` y seleccione una base de datos:

```
shell> mysql base-de-datos
```

(En la mayoría de las instalaciones de MySQL, podrá emplear la base de datos `test`).

Puede crear e ingresar datos a la tabla del ejemplo utilizando estas sentencias:

```
mysql> 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));
mysql> 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);
```

Luego de ejecutar estas sentencias, la tabla debería tener el siguiente contenido:

```
mysql> 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. El valor máximo de una columna

"¿Cuál es el número de ítem más alto?"

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2. El registro que tiene el valor máximo de determinada columna

Tarea: Encontrar el número, distribuidor y precio del artículo más costoso.

En MySQL 5.0 (y en SQL estándar), esto se hace fácilmente con una subconsulta:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);
```

Otra solución es ordenar las columnas por precio, en forma descendente, y obtener solamente el primer registro utilizando la cláusula `LIMIT`, específica de MySQL:

```
SELECT article, dealer, price
FROM   shop
ORDER BY price DESC
LIMIT 1;
```

Nota: Si hubiera varios artículos que presenten el precio más alto, cada uno a 19.95, la solución `LIMIT` sólo mostraría el primero de ellos.

3.6.3. Máximo de columna por grupo

Tarea: Encontrar el precio más alto por artículo.

```
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. Los registros de un grupo que tienen el máximo valor en alguna columna

Tarea: Para cada artículo, encontrar el o los distribuidores con el precio más alto.

En MySQL 5.0 (y en SQL estándar), este problema puede resolverse con una subconsulta como esta:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);
```

3.6.5. Utilización de variables de usuario

Se pueden emplear variables de usuario de MySQL para retener resultados sin necesidad de almacenarlos en variables del lado del cliente. (Consulte Sección 9.3, “Variables de usuario”.)

Por ejemplo, para encontrar los artículos con el precio más alto y más bajo se puede hacer lo siguiente:

```
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 |
+---------+--------+-------+
```

3.6.6. Usar claves foráneas (foreign keys)

En MySQL, las tablas `InnoDB` soportan restricciones de claves foráneas. Consulte Capítulo 15, *El motor de almacenamiento `InnoDB`*. Consulte también Sección 1.7.5.5, “Claves foráneas (foreign keys)”.

No se requiere una restricción de clave foránea para simplemente unir dos tablas. Para otros tipos de tabla que no sean `InnoDB`, es posible, al momento de definir una columna, utilizar una cláusula `REFERENCES`*`tbl_name (col_name)`*, la cual no tiene efecto real y *funciona solamente como un recordatorio o comentario de que la columna que se está definiendo está dirigida a hacer referencia a una columna en otra tabla*. Al emplear esta sintaxis es muy importante comprender que:

- MySQL no efectúa ningún tipo de `CHECK` o comprobación para asegurarse de que *`col_name`* realmente existe en *`tbl_name`* (o incluso que *`tbl_name`* existe).
- MySQL no realiza ningún tipo de acción sobre *`tbl_name`* tal como borrar filas en respuesta a acciones ejecutadas sobre filas en la tabla que se está definiendo; en otras palabras, esta sintaxis no produce

por sí misma un comportamiento `ON DELETE` u `ON UPDATE`. (Inclusive cuando se puede escribir una cláusula `ON DELETE` u `ON UPDATE` como parte de la cláusula `REFERENCES`, estas son también ignoradas).

- Esta sintaxis crea una *columna*; **no** crea ninguna clase de índice o campo clave.
- Esta sintaxis causará un error si se la emplea durante la definición de una tabla `InnoDB`.

Una columna creada de esta forma se puede utilizar como columna de unión, como se muestra aquí:

```
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 |
+----+---------+--------+-------+
|  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, shirt s
 WHERE p.name LIKE 'Lilliana%'
   AND s.owner = p.id
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

Cuando se usa de esta manera, la cláusula `REFERENCES` no es mostrada en la salida de `SHOW CREATE TABLE` o `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
```

El uso de `REFERENCES` como comentario o "recordatorio" en la definición de una columna funciona en tablas `MyISAM` y `BerkeleyDB`.

3.6.7. Buscar usando dos claves

Un `OR` empleando una única clave es bien optimizado, como es el manejo de `AND`

El único caso difícil es la búsqueda sobre dos diferentes claves combinadas con `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

Esto se ha optimizado a partir de MySQL 5.0.0. Consulte Sección 7.2.6, “Index Merge Optimization”.

En MySQL 5.0 tambien se puede resolver eficientemente este problema utilizando una `UNION` que combine la salida de dos sentencias `SELECT` separadas. Consulte Sección 13.2.7.2, “Sintaxis de `UNION`”.

Cada sentencia `SELECT` busca en solamente una clave y puede ser optimizada:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8. Calcular visitas diarias

El siguiente ejemplo muestra cómo se pueden utilizar las funciones de bits para calcular la cantidad de dias de un mes que un usuario ha visitado una página Web.

```
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);
```

La tabla del ejemplo contiene valores de año, mes y dia que representan las visitas de los usuarios a la página. Para determinar en cuántos días diferentes del mes se produjeron las visitas, se emplea esta consulta:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
      GROUP BY year,month;
```

La cual devuelve:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

La consulta calcula cuantos días diferentes aparecen en la tabla para cada combinación de año y mes, removiendo automáticamente las entradas duplicadas.

3.6.9. Utilización de AUTO_INCREMENT

El atributo AUTO_INCREMENT puede utilizarse para generar un identificador único para cada nueva fila:

```
CREATE TABLE animals (
             id MEDIUMINT NOT NULL AUTO_INCREMENT,
             name CHAR(30) NOT NULL,
             PRIMARY KEY (id)
             );
INSERT INTO animals (name) VALUES ('dog'),('cat'),('penguin'),
                                  ('lax'),('whale'),('ostrich');
SELECT * FROM animals;
```

Lo cual devuelve:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

Para obtener el valor AUTO_INCREMENT más recientemente generado se puede utilizar la funcion SQL LAST_INSERT_ID() o la función del API de C mysql_insert_id(). Estas funciones son específicas de cada conexión, de modo que su valor de retorno no es afectado por las inserciones realizadas a través de otras conexiones.

Nota: Para una inserción de múltiples filas, LAST_INSERT_ID()/mysql_insert_id() retornan el valor AUTO_INCREMENT de la **primera** de las filas insertadas. Esto permite que las inserciones de múltiples filas sean reproducidas correctamente en otros servidores en una configuración de replicación.

Para tablas `MyISAM` y `BDB` se puede especificar `AUTO_INCREMENT` sobre una columna secundaria en un índice de múltiples columnas. En este caso, el valor generado para la columna `AUTO_INCREMENT` es calculado como `MAX(auto_increment_column)+1 WHERE prefix=given-prefix`. Esto es útil cuando se desea colocar datos en grupos ordenados.

```
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)
             );
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;
```

Lo cual devuelve:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

Nótese que en este caso (cuando la columna `AUTO_INCREMENT` es parte de un índice de múltiples columnas), los valores `AUTO_INCREMENT` son reutilizados si se elimina la fila con el valor `AUTO_INCREMENT` más alto en cualquier grupo. Esto ocurre incluso para tablas `MyISAM`, en las que los valores `AUTO_INCREMENT` normalmente no son reutilizados

Si la columna `AUTO_INCREMENT` es parte de varios índices, MySQL generará valores secuenciales empleando el índice que comienza con la columna `AUTO_INCREMENT`, si hay uno. Por ejemplo, si la tabla `animals` contiene los índices `PRIMARY KEY (grp, id)` e `INDEX (id)`, MySQL ignoraría el índice `PRIMARY KEY` al generar valores secuenciales. Como resultado, la tabla contendría una secuencia simple, sin considerar el valor `grp`.

3.7. Consultas del proyecto Mellizos (Twin)

En Analytikerna y Lentus, hemos estado realizando los sistemas y el trabajo de campo para un gran proyecto de investigación. Este proyecto es en colaboración entre el Institute of Environmental Medicine del Karolinska Institutet Stockholm y la Sección de Investigación Clínica sobre Envejecimiento y Psicología de la Universidad de California del Sur.

El proyecto comprende una parte de selección, en la cual se entrevista por teléfono a todos los gemelos de Suecia mayores de 65 años. Aquellos que satisfacen ciertos criterios son pasados a la siguiente etapa. En esta, los gemelos que desean participar son visitados por un equipo de médico y enfermera. Algunos de los éxamenes practicados son físico, neuropsicológico, laboratorio, diagnóstico neurológico por imágenes, evaluación de estado psicológico, y recolección de la historia familiar. Adicionalmete, se recogen datos sobre factores de riesgo médicos y ambientales.

Se puede ver más información sobre estudio de Gemelos en: http://www.mep.ki.se/twinreg/index_en.html

La última parte del proyecto es administrada mediante una interface Web escrita usando Perl y MySQL.

Cada noche, los datos de las entrevistas son volcados en una base de datos MySQL.

3.7.1. Encontrar todos los mellizos no repartidos

La siguiente consulta es empleada para determinar quiénes pasan a la segunda parte del proyecto:

```
SELECT
    CONCAT(p1.id, p1.tvab) + 0 AS tvid,
    CONCAT(p1.christian_name, ' ', p1.surname) AS Name,
    p1.postal_code AS Code,
    p1.city AS City,
    pg.abrev AS Area,
    IF(td.participation = 'Aborted', 'A', ' ') AS A,
    p1.dead AS dead1,
    l.event AS event1,
    td.suspect AS tsuspect1,
    id.suspect AS isuspect1,
    td.severe AS tsevere1,
    id.severe AS isevere1,
    p2.dead AS dead2,
    l2.event AS event2,
    h2.nurse AS nurse2,
    h2.doctor AS doctor2,
    td2.suspect AS tsuspect2,
    id2.suspect AS isuspect2,
    td2.severe AS tsevere2,
    id2.severe AS isevere2,
    l.finish_date
FROM
    twin_project AS tp
    /* For Twin 1 */
    LEFT JOIN twin_data AS td ON tp.id = td.id
              AND tp.tvab = td.tvab
    LEFT JOIN informant_data AS id ON tp.id = id.id
              AND tp.tvab = id.tvab
    LEFT JOIN harmony AS h ON tp.id = h.id
              AND tp.tvab = h.tvab
    LEFT JOIN lentus AS l ON tp.id = l.id
              AND tp.tvab = l.tvab
    /* For Twin 2 */
    LEFT JOIN twin_data AS td2 ON p2.id = td2.id
              AND p2.tvab = td2.tvab
    LEFT JOIN informant_data AS id2 ON p2.id = id2.id
              AND p2.tvab = id2.tvab
    LEFT JOIN harmony AS h2 ON p2.id = h2.id
              AND p2.tvab = h2.tvab
    LEFT JOIN lentus AS l2 ON p2.id = l2.id
              AND p2.tvab = l2.tvab,
    person_data AS p1,
    person_data AS p2,
    postal_groups AS pg
WHERE
    /* p1 gets main twin and p2 gets his/her twin. */
    /* ptvab is a field inverted from tvab */
    p1.id = tp.id AND p1.tvab = tp.tvab AND
    p2.id = p1.id AND p2.ptvab = p1.tvab AND
    /* Just the screening survey */
    tp.survey_no = 5 AND
    /* Skip if partner died before 65 but allow emigration (dead=9) */
    (p2.dead = 0 OR p2.dead = 9 OR
     (p2.dead = 1 AND
      (p2.death_date = 0 OR
       (((TO_DAYS(p2.death_date) - TO_DAYS(p2.birthday)) / 365)
        >= 65))))
    AND
```

```
    (
    /* Twin is suspect */
    (td.future_contact = 'Yes' AND td.suspect = 2) OR
    /* Twin is suspect - Informant is Blessed */
    (td.future_contact = 'Yes' AND td.suspect = 1
                               AND id.suspect = 1) OR
    /* No twin - Informant is Blessed */
    (ISNULL(td.suspect) AND id.suspect = 1
                        AND id.future_contact = 'Yes') OR
    /* Twin broken off - Informant is Blessed */
    (td.participation = 'Aborted'
     AND id.suspect = 1 AND id.future_contact = 'Yes') OR
    /* Twin broken off - No inform - Have partner */
    (td.participation = 'Aborted' AND ISNULL(id.suspect)
                                  AND p2.dead = 0))
    AND
    l.event = 'Finished'
    /* Get at area code */
    AND SUBSTRING(p1.postal_code, 1, 2) = pg.code
    /* Not already distributed */
    AND (h.nurse IS NULL OR h.nurse=00 OR h.doctor=00)
    /* Has not refused or been aborted */
    AND NOT (h.status = 'Refused' OR h.status = 'Aborted'
    OR h.status = 'Died' OR h.status = 'Other')
ORDER BY
    tvid;
```

Algunas explicaciones:

- `CONCAT(p1.id, p1.tvab) + 0 AS tvid`

  Se desea ordenar por la concatenación de `id` y `tvab` en orden numérico. El agregado de `0` al resultado provoca que MySQL lo trate como un número.

- columna `id`

  Identifica una pareja de gemelos. Es un campo clave en todas las tablas.

- columna `tvab`

  Identifica a un gemelo en una pareja. Toma un valor de `1` o `2`.

- columna `ptvab`

  Es lo inverso de `tvab`. Cuando `tvab`. vale `1`, este vale `2`, y viceversa. El motivo de su existencia es para ahorrar tipeo y facilitarle a MySQL la optimización de la consulta.

Esta consulta muestra, entre otras cosas, cómo realizar búsquedas en una tabla a partir de la misma tabla con una unión (`p1` y `p2`). En el ejemplo, esto se usa para verificar cuándo una pareja de gemelos murieron antes de cumplir los 65 años. Si sucede eso, la fila no se devuelve.

Todo lo mencionado anteriormente existe en todas las tablas con información relativa a gemelos. Hay un índice definido sobre los campos `id,tvab` (en todas las tablas) y sobre `id,ptvab` (`person_data`) para realizar las consultas más rápidamente.

En nuestra máquina de producción (un UltraSPARC a 200Mhz), esta consulta devuelve cerca de 150-200 filas y toma menos de un segundo.

La cantidad actual de registros en las tablas usadas en la consulta:

Table	Rows

person_data	71074
lentus	5291
twin_project	5286
twin_data	2012
informant_data	663
harmony	381
postal_groups	100

3.7.2. Mostrar una tabla de estado de mellizos

Cada entrevista termina con un código de estado llamado `event`. La consulta mostrada aquí se emplea para mostrar una tabla sobre todas las parejas de gemelos combinadas por evento. Esto indica en cuantas parejas ambos gemelos llegaron al final, en cuantas uno llego y el otro fue rechazado, etc.

```
SELECT
        t1.event,
        t2.event,
        COUNT(*)
FROM
        lentus AS t1,
        lentus AS t2,
        twin_project AS tp
WHERE
        /* We are looking at one pair at a time */
        t1.id = tp.id
        AND t1.tvab=tp.tvab
        AND t1.id = t2.id
        /* Just the screening survey */
        AND tp.survey_no = 5
        /* This makes each pair only appear once */
        AND t1.tvab='1' AND t2.tvab='2'
GROUP BY
        t1.event, t2.event;
```

3.8. Usar MySQL con Apache

Existen programas que permiten autenticar usuarios a partir de una base de datos MySQL y también escribir ficheros de log en una tabla MySQL.

Se puede modificar el formato de logging de Apache para que MySQL pueda interpretarlo, colocando lo siguiente en el fichero de configuración de Apache:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

Para cargar dentro de MySQL un fichero de log en dicho formato, se puede emplear una sentencia como esta:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

La tabla destino debería ser creada de forma que contenga las columnas tal como las especifica la línea `LogFormat`.

Capítulo 4. Usar los programas MySQL

Tabla de contenidos

Este capítulo proporciona una descripción concisa de las utilidades de línea de comandos provistas por MySQL AB y de las opciones que se puden suministrar al ejecutarlas. La mayoría de los programas tienen opciones que son propias de su operación, pero la sintaxis para especificarlas es idéntica para todos. En los capítulos posteriores se brinda una descripción más detallada de cada utilidad, incluyendo las opciones que reconoce cada una

MySQL AB también suministra tres programas con interfaz gráfica de usuario para utilizar con el servidor de bases de datos MySQL:

- MySQL Administrator: Esta herramienta se emplea para la administración de servidores, bases de datos, tablas y usuarios de MySQL.
- MySQL Query Browser: Esta herramienta gráfica es provista por MySQL AB para crear, ejecutar, y optimizar consultas dirigidas a bases de datos MySQL.
- MySQL Migration Toolkit: Herramienta orientada a brindar asistencia en el proceso de migración de esquemas y datos desde otros sistemas gestores de bases de datos relacionales hacia MySQL.

4.1. Panorámica de programas MySQL

MySQL AB proporciona varios tipos de programas:

- El servidor MYSQL y los scripts de inicio del servidor:
  - `mysqld` es el servidor MySQL
  - `mysqld_safe`, `mysql.server`, y `mysqld_multi` son scripts de inicio del servidor
  - `mysql_install_db` inicializa el directorio "data" y las bases de datos que MySQL instala por defecto.

  Estos programas son comentados posteriormente en Capítulo 5, *Administración de bases de datos*.
- Programas cliente que acceden al servidor:
  - `mysql` es un programa cliente que porporciona una interfaz de linea de comandos para ejecutar sentencias SQL en modo interactivo o por lotes.
  - `mysqladmin` es un cliente para administración.
  - `mysqlcheck` ejecuta operaciones de mantenimiento de tablas.
  - `mysqldump` y `mysqlhotcopy` son utilidades para copia de respaldo.

- `mysqlimport` realiza importación de ficheros de datos.
- `mysqlshow` muestra información relativa a tablas y bases de datos.

Estos programas son comentados posteriormente en Capítulo 8, *Programas cliente y utilidades MySQL*.

- Programas que operan independientemente del servidor:
  - `myisamchk` ejecuta operaciones de mantenimiento de tablas.
  - `myisampack` genera tablas comprimidas, de sólo lectura.
  - `mysqlbinlog` es una herramienta para procesar archivos de registro binario (binary logs).
  - `perror` informa el significado de un código de error.

  `myisamchk` es comentado posteriormente en Capítulo 5, *Administración de bases de datos*. Los demás programas tienen su descripción en Capítulo 8, *Programas cliente y utilidades MySQL*.

La mayoría de las distribuciones de MySQL incluyen todos los programas mencionados, con excepción de los que son específicos de cada plataforma. (Por ejemplo, los scripts de inicio de servidor no son necesarios en Windows). Otra excepción es que las distribuciones RPM son más especializadas. Existe una RPM para el servidor, otra para los programas cliente, etc. En el caso de no hallar uno o más programas, consulte Capítulo 2, *Instalar MySQL* para ver información sobre los tipos de distribuciones y su contenido. Es posible que se necesite realizar una instalación adicional.

4.2. Invocar programas MySQL

Para invocar un programa MySQL desde la línea de comandos (desde el shell o el intérprete de comandos), introduzca el nombre del programa seguido de cualquier opción u otro argumento necesario para indicarle al programa la tarea que se desea llevar a cabo. Los siguientes comandos muestran algunos ejemplos de cómo invocar un programa. "`shell>`" representa el prompt del intérprete de comandos, no es parte de lo que debe teclearse. El prompt que se verá depende del intérprete de comandos utilizado. Algunos prompts típicos son: `$` para `sh` o `bash`, `%` para `csh` or `tcsh`, y `C:\>` para el `command.com` o `cmd.exe` de Windows.

```
shell> mysql test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump --user=root personnel
```

Los argumentos precedidos por guiones son las opciones del programa. Estas opciones, habitualmente, especifican el tipo de conexión al servidor o afectan el modo de operación del programa. Las opciones tienen una sintaxis que se describe en Sección 4.3, "Especificar opciones de programa".

Los argumentos que no son opciones (que no estan precedidos por guiones) suministran información adicional al programa. Por ejemplo, el programa `mysql` interpreta que el primer argumento no opción proporcionado es el nombre de la base de datos a utilizar, de forma que el comando `mysql test` indica que se desea emplear la base de datos `test`.

Las secciones siguientes, donde se describe individualmente cada programa, informan las opciones que es posible utilizar y el significado de los argumentos no opción adicionales.

Algunas opciones son comunes a un número de programas. Las más corrientes son `--host`, `--user`, y `--password`, las cuales especifican parámetros de conexión. Indican el ordenador host donde se está ejecutando el servidor de bases de datos MySQL, y el nombre de usuario y contraseña de la cuenta

MySQL empleada para acceder. Estas tres opciones son aplicables a todos los programas cliente MySQL, permiten especificar el servidor al cual conectarse y qué cuenta usar para ello.

Es posible que necesite invocar a los programas MySQL utilizando la ruta al directorio `bin` en el que están instalados. Este es el caso más probable si obtiene un mensaje de error “program not found” cada vez que se intenta ejecutar un programa MySQL desde cualquier directorio que no sea `bin`. A fin de hacer más conveniente el uso de MySQL, puede agregar la ruta al directorio `bin` dentro de la variable de entorno `PATH`. Entonces, para ejecutar un programa sólo se necesitará ingresar su nombre, no la ruta completa.

Consulte la documentación específica de su intérprete de comandos para obtener instrucciones sobre cómo establecer el valor de `PATH`, dado que la sintaxis para establecer variables de entorno es propia de cada uno.

4.3. Especificar opciones de programa

Al ejecutar programas MySQL se les pueden indicar opciones en varias formas:

- En la línea de comandos, a continuación del nombre del programa. Este es el modo más común para opciones que se aplican a una ejecución específica del programa.
- En un fichero de opciones, que el programa lee al iniciarse. Esto es lo acostumbrado para opciones que se desea que el programa use cada vez que se ejecuta.
- En variables de entorno. Esto es útil para opciones que se desean aplicar cada vez que el programa se ejecuta, si bien en la práctica, para este propósito, es más común emplear ficheros de opciones. En (Sección 5.11.2, “Ejecutar varios servidores en Unix” se expone una situación donde las variables de entorno pueden ser muy útiles. Allí, se describe una práctica técnica que usa estas variables para indicar el número de puerto TCP/IP y el fichero socket de Unix, tanto para los programas cliente como para el servidor.

Los programas MySQL determinan qué opciones les fueron suministradas examinando en primer lugar las variables de entorno, luego los ficheros de opciones, y, finalmente, la línea de comandos. Si una opción se especifica más de una vez, la última tiene precedencia. Esto significa que las variables de entorno tienen la prioridad más baja, y la línea de comandos, la más alta.

Se puede aprovechar la forma en que los programas MySQL procesan las opciones si se especifica las opciones por defecto de un programa dentro de un fichero de opciones. De ese modo, no es necesario teclearlas cada vez que se ejecuta el programa, pero los valores por defecto indicados pueden ser reemplazados por otros que se incluyan en la línea de comandos.

4.3.1. Usar opciones en la línea de comandos

Las opciones de programa indicadas en la línea de comandos están sujetas a estas reglas:

- Las opciones se colocan después del nombre del comando.
- Una opción comienza con uno o dos guiones, dependiendo de si se ha utilizado la forma corta o larga de su nombre. Muchas opciones permiten ambas formas. Por ejemplo, `-?` y `--help` son, respectivamente, las formas corta y larga de la opción que solicita a un programa MySQL que muestre un mensaje de ayuda.
- Los nombres de opciones son case sensitive. Tanto `-v` como `-V` son correctos pero tienen distinto significado. (Corresponden a la forma corta de las opciones `--verbose` y `--version`.)
- Algunas opciones aceptan que se indique un valor a continuación del nombre. Por ejemplo, `-h localhost` o `--host=localhost` indican a un programa cliente MySQL el servidor de bases de

datos MySQL a utilizar. El valor de la opción le dice al programa el nombre del ordenador host donde el servidor de bases de datos MySQL se está ejecutando.

- Los nombres de opción largos se separan del valor asignado (si se les asigna uno) con un signo '='. Para un nombre de opción corto, el valor puede escribirse inmediatamente a continuación de la letra de la opción, o puede haber un espacio entre ambos. (`-hlocalhost` y `-h localhost` son equivalentes.) Una excepción a esta regla es la opción para suministrar la contraseña MySQL. Esta opción puede escribirse en forma larga como `--password=pass_val` o como `--password`. En el último caso, donde no se incluyó la contraseña, el programa la solicitará. También puede emplearse una forma corta como `-ppass_val` o como `-p`. Sin embargo, en la forma corta, si se suministra una contraseña debe figurar a continuación de la letra de opción *sin espacios intermedios*. La razón de esto es que, si hay un espacio a continuación de la letra de opción, el programa no tiene modo de saber si el argumento que sigue es la contraseña o alguna otra clase de argumento. Por esto, los siguientes dos comandos tienen significados completamente diferentes:

  ```
  shell> mysql -ptest
  shell> mysql -p test
  ```

  El primer comando le dice a `mysql` que utilice la contraseña `test`, pero no indica una base de datos por defecto. El segundo le dice a `mysql` que solicite la contraseña y que utilice `test` como base de datos por defecto.

Algunas opciones controlan comportamientos que deben habilitarse o deshabilitarse. Por ejemplo, el cliente `mysql` soporta una opción `--column-names` que determina si se mostrará o no una fila con los nombres de cada columna al principio de los resultados de una consulta. Por defecto, esta opción se encuentra habilitada. Sin embargo, en algunos casos se podria desear que permanezca inhabilitada, por ejemplo cuando la salida producida por `mysql` debe ser enviada a otro programa que espera sólo datos y no una linea inicial de encabezados.

Para deshabilitar los nombres de columnas, se especifica la opción empleando cualquiera de estas formas:

```
--disable-column-names
--skip-column-names
--column-names=0
```

Los prefijos `--disable` y `--skip` y el sufijo `=0` tienen el mismo efecto: deshabilitar la opción.

El modo “enabled” (habilitado) de la opción puede ser especificado en cualquiera de estas formas:

```
--column-names
--enable-column-names
--column-names=1
```

Si una opción se antecede con el prefijo `--loose`, el programa no terminará con un error si no es capaz de reconocer la opción, en lugar de ello emitirá una advertencia:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

El prefijo `--loose` puede ser útil cuando se ejecutan programas pertenecientes a múltiples versiones de MySQL en el mismo ordenador. Es particularmente útil cuando se utiliza un fichero de opciones. Una opción que puede no ser reconocida por todas las versiones de un programa, se antecede con el prefijo `--loose` (o bien `loose` en el caso de un fichero de opciones). Aquellas versiones de un programa que no reconozcan la opción emitirán una advertencia y la ignorarán. **Nota**: Esta estrategia requiere que las

versiones involucradas sean la 4.0.2 o posteriores, dado que antes de esa versión no existía el prefijo `--loose`.

Otra opción que puede ser ocasionalmente útil con `mysql` es `-e` o `--execute`, la cual se utiliza para enviar sentencias SQL al servidor. Las sentencias deben estar colocadas entre comillas (dobles o simples). No obstante, si se desea incluir dentro de la consulta valores colocados entre comillas, habría que emplear comillas dobles para delimitar la consulta y simples para los valores en su interior. Cuando se utiliza esta opción, `mysql` retorna al prompt del intérprete de comandos inmediatamente después de ejecutar la sentencia.

Por ejemplo, para obtener una lista de cuentas de usuario, puede hacerse lo siguiente:

```
shell> mysql -u root -p -e "SELECT User, Host FROM user" mysql
Enter password: ******
+------+-----------+
| User | Host      |
+------+-----------+
|      | gigan     |
| root | gigan     |
|      | localhost |
| jon  | localhost |
| root | localhost |
+------+-----------+
shell>
```

Observe que el nombre de la base de datos `mysql` fue pasado como un argumento separado. Sin embargo, la misma consulta se hubiera ejecutado indicando `mysql -u root -p -e "SELECT User, Host FROM mysql.user"` en el intérprete de comandos.

De este modo se pueden ejecutar múltiples sentencias SQL, si se las separa con punto y coma:

```
shell> mysql -u root -p --execute="SELECT Name FROM Country WHERE Name LIKE 'AU%';SELECT COUNT(*) FROM City
Enter password: ******
+-----------+
| Name      |
+-----------+
| Australia |
| Austria   |
+-----------+
+----------+
| COUNT(*) |
+----------+
|     4079 |
+----------+
```

Observe que la forma larga (`--execute`) debe ser seguida por un signo igual(`=`).

La opción `-e` puede usarse del mismo modo para pasar comandos al cliente de administración para MySQL Cluster `ndb_mgm`. Consulte Sección 16.3.6, “Apagado y encendido seguros” para ver un ejemplo.

4.3.2. Usar ficheros de opciones

Los programas MySQL pueden leer opciones de inicio desde ficheros de opciones (también llamados a veces ficheros de configuración). Los ficheros de opciones porporcionan una forma conveniente de especificar opciones comúnmente usadas sin que sea necesario ingresarlas en la línea de comandos cada vez que se ejecuta el programa.

Los siguientes programas soportan ficheros de opciones: `myisamchk`, `myisampack`, `mysql`, `mysql.server`, `mysqladmin`, `mysqlbinlog`, `mysqlcc`, `mysqlcheck`, `mysqld_safe`, `mysqldump`, `mysqld`, `mysqlhotcopy`, `mysqlimport`, y `mysqlshow`.

Nota: el uso de ficheros de opciones con programas de MySQL Cluster se cubre en Sección 16.4, “Configuración de MySQL Cluster”.

En Windows, los programas MySQL leen sus opciones de inicio en los siguientes ficheros:

Fichero	Contenido
WINDIR\my.ini	Opciones globales
C:\my.cnf	Opciones globales
INSTALLDIR\my.ini	Opciones globales
defaults-extra-file	El archivo especificado con --defaults-extra-file=*ruta*, si existe.

WINDIR representa la ubicación del directorio Windows. Por lo general, es C:\WINDOWS o C:\WINNT. Se puede determinar la localización exacta a través del valor de la variable de entorno WINDIR utilizando el siguiente comando:

```
C:\> echo %WINDIR%
```

INSTALLDIR representa el directorio de instalación de MySQL. Este es generalmente C:*PROGRAMDIR*\MySQL\MySQL 5.0 Server, donde *PROGRAMDIR* representa el directorio de programas (usualmente Archivos de Programa en versiones de Windows en español), donde se instaló MySQL 5.0 mediante los asistentes de instalación y configuración. Consulte Sección 2.3.5.14, “Dónde está el fichero my.ini”.

En Unix, los programas MySQL leen sus opciones de inicio en los siguientes ficheros:

Fichero	Contenido
/etc/my.cnf	Opciones globales
$MYSQL_HOME/my.cnf	Opciones específicas del servidor
defaults-extra-file	El archivo especificado con --defaults-extra-file=*ruta*, si existe.
~/.my.cnf	Opciones específicas del usuario

MYSQL_HOME es una variable de entorno que contiene la ruta al directorio donde reside el fichero my.cnf específico del servidor. (Este era DATADIR anteriormente a MySQL versión 5.0.3.)

Si MYSQL_HOME no tiene un valor establecido y hay un fichero my.cnf en *DATADIR* y no hay un fichero my.cnf en *BASEDIR*, mysqld_safe establece el valor de MYSQL_HOME en *DATADIR*. De otro modo, si MYSQL_HOME no tiene un valor establecido y no hay un fichero my.cnf en *DATADIR*, entonces mysqld_safe establece el valor de MYSQL_HOME en *BASEDIR*.

Generalmente es /usr/local/mysql/data para una instalación binaria o /usr/local/var para una instalación de código fuente. Observe que se trata de la ubicación del directorio de datos que se indicó al momento de la configuración, no de la especificada con --datadir cuando se inicia mysqld. El uso de --datadir no tiene efecto sobre el lugar donde el servidor busca los ficheros de opciones, porque esta búsqueda se produce antes de procesar cualquier argumento de línea de comandos.

MySQL busca ficheros de opciones exactamente en el orden descripto en la tabla y lee cualquiera que exista. Si se desea utilizar un fichero de opciones que no existe, se lo debe crear con un editor de texto plano. De existir múltiples ficheros de opciones, las opciones leidas en último lugar prevalecen sobre las anteriores.

Nota: En plataformas Unix, MySQL ignorará todo fichero de configuración que tenga permiso world-writable (esto es, modificable por todos los usuarios). Esto ha sido implementado intencionalmente como medida de seguridad.

Cualquier opción en formato de nombre largo que pueda suministrarse en la línea de comandos al ejecutar un programa MySQL puede ser colocada también en un fichero de opciones. Para obtener la lista de opciones disponibles para un programa determinado, el mismo debe ejecutarse con la opción `--help`.

La sintaxis para especificar opciones en un fichero es similar a cuando se hace en la línea de comandos, con la excepción de que se deben omitir los dos guiones iniciales.Por ejemplo, `--quick` o `--host=localhost` en la linea de comandos debería especificarse como `quick` o `host=localhost` en un fichero de opciones. Para indicar una opción de la forma `--loose-opt_name` en un fichero, debe escribirse como `loose-opt_name`.

Las lineas vacías de los ficheros de opciones se ignoran. Las líneas no vacías pueden tomar cualquiera de las siguientes formas:

- `#comentario`, `;comentario`

  Las líneas de comentario comienzan con '`#`' o '`;`'. Un comentario '`#`' puede aparecer incluso en el medio de una linea.

- `[grupo]`

  `grupo` es el nombre del programa o grupo para el cual se desea establecer opciones. Después de una linea de este tipo, cualquier linea `opción` o `set-variable` se aplicará a ese grupo hasta el final del fichero o hasta que se encuentre otra línea `grupo`.

- `opción`

  Equivale a `--opción` en la línea de comandos.

- `opción=valor`

  Equivale a `--opción=valor` en la línea de comandos. En un fichero de opciones está permitido colocar espacios a ambos lados del carácter '`=`', algo que no es posible en la línea de comandos. En MySQL 5.0 se puede encerrar el valor entre comillas simples o dobles. Esto es útil si el valor contiene un carácter como comentario '`#`' o espacios en blanco.

Los espacios en blanco sobrantes son automáticamente eliminados de los nombres de opciones y valores. se pueden utilizar las secuencias de escape '`\b`', '`\t`', '`\n`', '`\r`', '`\\`', y '`\s`' al especificar el valor de una opción si es necesario representar los caracteres backspace, tab, salto de linea, retorno de carro y espacio.

En Windows, si el valor de una opción representa una ruta a un directorio o un fichero, se debería especificar el valor utilizando '`/`' en lugar '`\`' como separador. Si se emplea '`\`', debe duplicarse y poner '`\\`', puesto que '`\`' es el caracter de escape en MySQL.

Si el nombre de un grupo es igual que el de un programa, las opciones en el grupo se aplicarán específicamente a ese programa.

El grupo de opciones `[client]` es leido por todos los programas clientes (pero **no** por `mysqld`). Esto permite especificar opciones aplicables a todos los clientes. Por ejemplo, `[client]` es el grupo perfecto para indicar la contraseña que se utiliza para conectarse al server. (Pero es necesario asegurarse que el fichero de opciones es accesible para lectura y escritura solamente por Usted, de modo que otras personas no puedan conocer la contraseña). Una opción no debe colocarse en el grupo `[client]` a

menos que sea reconocida por *todos* los programas cliente que se utilizan. Los programas que no la soporten terminarán después de mostrar un mensaje de error si se los intenta ejecutar.

A partir de MySQL 5.0.4 en la serie 5.0, es posible emplear directivas `!include` en los ficheros de opciones para incluir ficheros específicos y `!includedir` para incluir directorios Por ejemplo, para incluir el fichero `/home/mydir/myopt.cnf`, se puede usar lo siguiente:

```
!include /home/me/myopt.cnf
```

Para buscar en el directorio `/home/mydir` todos los ficheros con extensión `.cnf` y leerlos como ficheros de opciones, se debería utilizar:

```
!includedir /home/mydir
```

Observe que estas opciones son específicas de cada sección. Por ejemplo, suponga que fuera a utilizar en `my.cnf` algo como lo siguiente:

```
[mysqld]
!include /home/mydir/myopt.cnf
```

En ese caso, el fichero `myopt.cnf` sólo sería procesado por el servidor, y la directiva `!include` sería ignorada por cualquier aplicación cliente. Sin embargo, si empleara:

```
[mysqldump]
!includedir /home/mydir/my-dump-options
```

entonces el directorio `/home/mydir/my-dump-options` sería verificado en busca de ficheros de opciones con extensión `.cnf` únicamente por `mysqldump` y no por el servidor o por otras aplicaciones cliente.

Nota: En la actualidad, cualquier fichero que deba ser encontrado e incluido al usar la directiva `!includedir` **debe** tener en su nombre la extensión `.cnf`. En Windows, esta directiva también verifica en busca de ficheros con extensión `.ini`.

En la versión 4.0.14 de MySQL y posteriores, si se desea crear un grupo de opciones que deba ser leido únicamente por una versión específica de `mysqld`, se puede hacer dando a los grupos de opciones nombres como los siguientes: `[mysqld-4.0]`, `[mysqld-4.1]`, `[mysqld-5.0]`, y así sucesivamente. El siguiente grupo indica que la opción `--new` debería ser aplicada sólo por servidores de bases de datos MySQL versión 5.0.x:

```
[mysqld-5.0]
new
```

Aquí hay un fichero de opciones globales típico:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M
```

```
[mysqldump]
quick
```

El fichero de opciones anterior utiliza la sintaxis *`var_name=value`* para las líneas que establecen los valores de las variables `key_buffer_size` y `max_allowed_packet`.

Este es un fichero de opciones de usuario típico:

```
[client]
# El siguiente password se enviará a todos los clientes MySQL estándar
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

Si tiene una distribución de código fuente, podrá encontrar ficheros de opciones de ejemplo llamados `my-xxxx.cnf` en el directorio `support-files`. Si tiene una distribución binaria, busque en el directorio `support-files` bajo el directorio de instalación de MySQL. En Windows, los ficheros de opciones de ejemplo también se encuentran en el directorio de instalación de MySQL. (vea anteriormente en esta sección o Capítulo 2, *Instalar MySQL* si no sabe dónde se encuentra este directorio). Actualmente hay ficheros de opciones para sistemas pequeños, medios, grandes y muy grandes. Para experimentar con uno de estos ficheros, se lo debe copiar como `C:\my.cnf` en Windows o como `.my.cnf` en el directorio home en Unix.

Nota: La extensión `.cnf` de los ficheros de opciones podría no mostrarse en Windows

Todos los programas MySQL que soportan ficheros de opciones manejan las siguientes opciones de línea de comandos:

- `--no-defaults`

  No lee ningún fichero de opciones.

- `--print-defaults`

  Imprime el nombre del programa y todas las opciones que obtiene desde los ficheros de opciones.

- `--defaults-file=`*`path_name`*

  Utiliza solamente el fichero de opciones especificado. *`path_name`* es la ruta completa al fichero.

- `--defaults-extra-file=`*`path_name`*

  Utiliza el fichero de opciones especificado, lo procesa luego del fichero global de opciones pero antes del fichero de opciones del usuario. *`path_name`* es la ruta completa al fichero.

Para funcionar correctamente, cada una de estas opciones debe colocarse en la línea de comandos inmediatamente a continuación del nombre del comando, a excepción de `--print-defaults` que puede aparecer luego de `--defaults-file` o `--defaults-extra-file`.

En los scripts del shell puede utilizar el programa `my_print_defaults` para procesar ficheros de opciones. El siguente ejemplo muestra la salida que `my_print_defaults` produciría al solicitarle que muestre las opciones halladas en los grupos `[client]` y `[mysql]`:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Nota para desarrolladores: El manejo de ficheros de opciones está implementado en la librería cliente C simplemente a través del procesamiento de todas las opciones coincidentes (esto es, opciones en el grupo apropiado) antes que cualquier argumento de la línea de comandos. Esto funciona bien con programas que emplean la última aparición de una opción especificada múltiples veces. Si se tiene un programa C o C++ que maneja opciones especificadas múltiples veces pero que no lee ficheros de opciones, se necesita agregar solamente dos lineas para darle esa capacidad. Examine el código fuente de cualquiera de los programas cliente estándar de MySQL para ver cómo se hace.

Varias otras interfaces con MySQL pertenecientes a otros lenguajes se basan en la librería cliente de C, y algunas de ellas proporcionan una forma de acceder al contenido de ficheros de opciones. Esto incluye a Perl y Python. Consulte la documentación de su interface preferida para más detalles.

4.3.3. Usar variables de entorno para especificar opciones

Para especificar una opción empleando una variable de entorno, se establece el valor de la variable usando la sintaxis apropiada del procesador de comandos. Por ejemplo, en Windows o NetWare se establece la variable `USER` para indicar el nombre de la cuenta MySQL. Para hacerlo, se usa esta sintaxis:

```
SET USER=your_name
```

La sintaxis en Unix depende del shell. En el supuesto de que se quisiera especificar el número de puerto TCP/IP empleando la variable `MYSQL_TCP_PORT`, la sintaxis típica (tal como se hace en `sh`, `bash`, `zsh`, etc.) es la siguiente:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

El primer comando establece el valor de la variable, y el comando `export` exporta la variable hacia el entorno del shell de forma que su valor sea accesible a MySQL y otros procesos.

`csh` y `tcsh` son un caso similar. Cuando se ejecuta estos shells, se debe usar `setenv` para que el valor de la variable esté disponible para el entorno:

```
setenv MYSQL_TCP_PORT 3306
```

El comando que establece el valor de las variables de entorno puede ejecutarse en el intérprete de comandos para que tenga efecto inmediatamente. Estos valores existen hasta que se cierra la sesión de usuario. Para que los valores tomen efecto cada vez que se inicia sesión, deben ubicarse los comandos en un fichero de inicio que el intérprete de comandos lea en cada arranque. Algunos ficheros de inicio típicos son `AUTOEXEC.BAT` para Windows, `.bash_profile` para `bash`, o `.tcshrc` para `tcsh`. Consulte la documentación de su intérprete de comandos para detalles más específicos.

En Apéndice E, *Variables de entorno* se listan todas las variables de entorno que afectan la operación de los programas MySQL.

4.3.4. Utilización de opciones para establecer variables de programa

Muchos programas MySQL tienen variables internas que pueden ser establecidas en tiempo de ejecución. En MySQL 5.0, las variables de los programas reciben un valor del mismo modo que cualquier otra opción

de nombre largo que tome valores. Por ejemplo, `mysql` tiene una variable `max_allowed_packet` que controla el máximo tamaño de su buffer de comunicación. Para establecer el valor de la variable `max_allowed_packet` de `mysql` en 16MB, use cualquiera de los siguientes comandos:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

El primer comando especifica el valor en bytes. El segundo especifica el valor en megabytes. Los valores de las variables pueden tener un sufijo `K`, `M`, o `G` (ya sea en mayúsculas o minúsculas) para indicar la unidad, que puede ser kilobytes, megabytes, o gigabytes.

En un fichero de opciones, la variable se coloca sin precederla con dos guiones:

```
[mysql]
max_allowed_packet=16777216
```

O bien:

```
[mysql]
max_allowed_packet=16M
```

Si lo prefiere, los caracteres de subrayado en una variable pueden cambiarse por guiones:

Nota: La antigua sintaxis `--set-variable = opcion=valor` aún se reconoce en MySQL 5.0, pero está obsoleta

Algunas variables del servidor pueden recibir valores en tiempo de ejecución. Para más detalles, consulte Sección 5.3.3.1, “Variables de sistema dinámicas”.

Capítulo 5. Administración de bases de datos

Tabla de contenidos

Este capítulo cubre tópicos que tratan la administración de una instalación de MySQL, como configurar el servidor, administrar cuentas de usuario y realizar copias de seguridad.

5.1. El servidor MySQL y scripts de arranque del servidor

MySQL server, `mysqld`, es el programa principal que realiza la mayoría del trabajo en una instalación MySQL. El servidor está acompañado por varios scripts que realizan operaciones de inicialización cuando instala MySQL o se tratan de programas de ayuda para asistirle en la inicialización y parada del servidor.

Esta sección proporciona una visión global del servidor y de los programas relacionados, e información acerca de los scripts de inicialización del servidor. La información acerca de configurar el servidor se proporciona en Sección 5.3, “Configuración del servidor MySQL”.

5.1.1. Panorámica de los programas scripts y las utilidades del lado del servidor (server-side)

Todos los programas MySQL aceptan diferentes opciones. Sin embargo, cada programa MySQL proporciona una opción `--help` que puede usar para obtener una descripción de las opciones del programa. Por ejemplo, pruebe `mysqld --help`.

Puede cambiar las opciones por defecto en todos los programas estándar especificando opciones en la línea de comandos o en un fichero de opciones. Sección 4.3, “Especificar opciones de programa”.

La siguiente lista describe brevemente MySQL server y sus programas relacionados:

- `mysqld`

  El demonio SQL (esto es, el servidor MySQL). Para usar programas clientes, este programa debe estar en ejecución, ya que los programas ganan el acceso a la base de datos conectándose al servidor. Consulte Sección 5.3, “Configuración del servidor MySQL”.

- mysqld-max

  Una versión del servidor que incluye características adicionales. Consulte Sección 5.1.2, “El servidor extendido de MySQL mysqld-max”.

- mysqld_safe

  Un script de arranque del servidor. mysqld_safe intenta inicializar mysqld-max si existe, y mysqld en caso contrario. Consulte Sección 5.1.3, “El script de arranque del servidor mysqld_safe”.

- mysql.server

  Un script de arranque del servidor. Este script se usa en sistemas que utilizan directorios de ejecución que contienen scripts que inicializan servicios para niveles de ejecución particular. Invoque mysqld_safe para inicializar el servidor MySQL. Consulte Sección 5.1.4, “El script mysql.server para el arranque del servidor”.

- mysqld_multi

  Un script de arranque del servidor que puede arrancar o parar varios servidores instalados en el sistema. Consulte Sección 5.1.5, “El programa mysqld_multi para gestionar múltiples servidores MySQL”.

- mysql_install_db

  Este script crea las tablas de permisos de MySQL con privilegios por defecto. Normalmente se ejecuta sólo una vez, cuando se instala por primera vez MySQL en el sistema. Consulte Sección 2.9.2, “Pasos a seguir después de la instalación en Unix”.

- mysql_fix_privilege_tables

  Este script se usa tras una actualización, para actualizar las tablas de permisos con cualquier cambio que se hayan hecho en nuevas versiones de MySQL. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Hay otros programas que también se ejecutan en la máquina del servidor:

- myisamchk

  Una utilidad para describir, testear, optimizar y reparar tablas MyISAM. myisamchk se describe en Sección 5.8.3, “Mantenimiento de tablas y recuperación de un fallo catastrófico (crash)”.

- make_binary_distribution

  Este programa crea una publicación binaria de un MySQL compilado. Puede enviarse por FTP a /pub/mysql/upload/ a ftp.mysql.com para el uso de otros usuarios de MySQL.

- mysqlbug

  El script para reportar bugs. Puede usarse para enviar un reporte de bug a la lista de correo de MySQL. (Puede visitar http://bugs.mysql.com/ para rellenar un reporte de bug en línea. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.)

5.1.2. El servidor extendido de MySQL mysqld-max

El servidor MySQL-Maxk es una versión del servidor MySQL mysqld compilada para añadir características adicionales.

La distribución a usar depende de la plataforma:

- Para Windows, las distribuciones binarias de MySQL incluyen ambos servidores (mysqld.exe) y el servidor MySQL-Max (mysqld-max.exe), por lo que no es necesario adquirir ninguna distribución especial. Símplemente use una distribución normal para Windows, disponible en http://dev.mysql.com/downloads/. Consulte Sección 2.3, “Instalar MySQL en Windows”.
- Para Linux, si instala MySQL utilizando una distribución RPM, use el RPM MySQL-server en primer lugar para instalar una versión estándar del servidor llamada mysqld. A continuación use el RPM MySQL-Max para instalar el servidor llamado mysqld-max. El RPM MySQL-Max presupone que el RPM con el servidor normal está instalado. Consulte Sección 2.4, “Instalar MySQL en Linux” para más información sobre los paquetes RPM para Linux.
- Todas las otras distribuciones MySQL-Max contienen un único servidor llamado mysqld pero que tiene las características adicionales incluídas.

Puede encontrar los binarios para MySQL-Max en la página Web de MySQL AB en http://dev.mysql.com/downloads/.

MySQL AB compila el servidor MySQL-Max usando las siguientes opciones de configure:

- --with-server-suffix=-max

  Esta opción añade un sufijo -max a la cadena de caracteres mysqld de la versión.

- --with-innodb

  Esta opción activa el soporte para el motor de almacenamiento InnoDB. Los servidores MySQL-Max siempre incluyen soporte para InnoDB . Desde MySQL 4.0 en adelante, se incluye por defecto InnoDB en todas las distribuciones binarias, por lo que no necesita un servidor MySQL-Max simplemente para obtener soporte para InnoDB.

- --with-bdb

  Esta opción activa el soporte para el motor de almacenamiento Berkeley DB (BDB).

- USE_SYMDIR

  Esta definición está activada para activar el soporte para links simbólicos en Windows. En MySQL 5.0, el soporte para links simbólicos está disponible para todos los servidores Windows, así que un servidor Max no es necesario para aprovechar esta característica.

- --with-ndb-cluster

  Esta opción activa el soporte para el motor de almacenamiento NDB Cluster . Actualmente (como en5.0.9-beta), el Cluster se soporta en Linux, Solaris, y Mac OS X solamente. Algunos usuarios han reportado éxitos al utilizar MySQL Cluster compilado de las fuentes en sistemas operativos basados en BSD, pero no están soportados oficialmente de momento.

Las distribuciones binarias de MySQL-Max son útiles para aquéllos que quieran instalar programas precompilados. Si compila MySQL a partir de una distribución fuente, puede construir su propio servidor de estilo Max activando las mismas características en tiempo de configuración que usan las distribuciones binarias de MySQL-Max al ser creadas.

Los servidores MySQL-Max incluyen el motor de almacenamiento BerkeleyDB (BDB) cuando es posible, pero no todas las plataformas soportan BDB.

Los servidores MySQL-Max para Solaris, Mac OS X, y Linux (en la mayoría de plataformas) incluyen soporte para el motor de almacenamiento NDB Cluster . Tenga en cuenta que el servidor debe reiniciarse

con la opción ndbcluster para ejecutar el servidor como parte de un MySQL Cluster. (Para más detalles, consulte Sección 16.4, “Configuración de MySQL Cluster”.)

La siguiente tabla muestra en qué plataformas los binarios de MySQL-Max incluyen soporte para BDB y/o NDB Cluster:

Sistema	Soporte BDB	Soporte NDB
AIX 4.3	N	N
HP-UX 11.0	N	N
Linux-Alpha	N	S
Linux-IA-64	N	N
Linux-Intel	S	S
Mac OS X	N	N
NetWare	N	N
SCO OSR5	S	N
Solaris-SPARC	S	S
Solaris-Intel	N	S
UnixWare	S	N
Windows NT/2000/XP	S	N

Para ver los motores de almacenamiento que soporta su servidor, ejecute el siguiente comando:

```
mysql> SHOW ENGINES;
+------------+---------+----------------------------------------------------------------+
| Engine     | Support | Comment                                                        |
+------------+---------+----------------------------------------------------------------+
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance         |
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables      |
| HEAP       | YES     | Alias for MEMORY                                               |
| MERGE      | YES     | Collection of identical MyISAM tables                          |
| MRG_MYISAM | YES     | Alias for MERGE                                                |
| ISAM       | NO      | Obsolete storage engine, now replaced by MyISAM                |
| MRG_ISAM   | NO      | Obsolete storage engine, now replaced by MERGE                 |
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys     |
| INNOBASE   | YES     | Alias for INNODB                                               |
| BDB        | YES     | Supports transactions and page-level locking                   |
| BERKELEYDB | YES     | Alias for BDB                                                  |
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables                 |
| NDB        | NO      | Alias for NDBCLUSTER                                           |
| EXAMPLE    | NO      | Example storage engine                                         |
| ARCHIVE    | YES     | Archive storage engine                                         |
| CSV        | NO      | CSV storage engine                                             |
| FEDERATED  | YES     | Federated MySQL storage engine                                 |
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it disappears) |
+------------+---------+----------------------------------------------------------------+
18 rows in set (0.00 sec)
```

(Consulte también Sección 13.5.4.8, “Sintaxis de SHOW ENGINES”.)

Antes de MySQL 4.1.2, SHOW ENGINES no está disponible. Use el siguiente comando en su lugar y compruebe que el valor de la variable para el motor de almacenamiento en que está interesado:

```
mysql> SHOW VARIABLES LIKE 'have%';
+----------------------+-------+
```

```
| Variable_name         | Value |
+-----------------------+-------+
| have_archive          | YES   |
| have_bdb              | YES   |
| have_blackhole_engine | YES   |
| have_compress         | YES   |
| have_crypt            | NO    |
| have_csv              | NO    |
| have_example_engine   | NO    |
| have_federated_engine | YES   |
| have_geometry         | YES   |
| have_innodb           | YES   |
| have_isam             | NO    |
| have_ndbcluster       | NO    |
| have_openssl          | YES   |
| have_query_cache      | YES   |
| have_raid             | NO    |
| have_rtree_keys       | YES   |
| have_symlink          | YES   |
+-----------------------+-------+
17 rows in set (0.06 sec)
```

La salida precisa de estos comandos SHOW variará en función de la versión de MySQL usada (y las características que haya activadas). Los valores en la segunda columna indican el nivel de soporte por parte del servidor para cada característica, como se muestra:

Valor	Significado
YES	La característica se soporta y está activa.
NO	La característica no se soporta.
DISABLED	La característica se soporta pero no está activa.

Un valor NO significa que el servidor está compilado sin soporte para la característica, por lo que no puede activarse en tiempo de ejecución.

Un valor DISABLED aparece porque el servidor se arrancó con una opción que deshabilita la característica, o porque no todas las opciones requeridas para activarla se han dado. En el último caso, el fichero de log de error *host_name*.err debería contener la razón indicando porqué la opción está deshabilitada.

Puede ver el mensaje DISABLED para los motores de almacenamiento InnoDB, BDB, o ISAM si el servidor está compilado para soportarlos pero se arrancó con las opciones --skip-innodb, --skip-bdb, o --skip-isam en tiempo de ejecución.

Todos los servidores MySQL soportan tablas MyISAM , ya que MyISAM es el motor de almacenamiento por defecto.

5.1.3. El script de arranque del servidor mysqld_safe

mysqld_safe es la manera recomendada de iniciar mysqld un servidor en Unix y NetWare mysqld_safe añade algunas características de seguridad como reiniciar el servidor cuando ocurre un error y guardar la información en tiempo de ejecución en un registro de errores. Los comportamientos especificos de NetWare se mencionan más adelante en esta sección.

Nota: Para preservar la compatibilidad con antiguas versiones de MySQL, las distribuciones binarias de MySQL todavía incluyen safe_mysqld como en enlace simbólico a mysqld_safe. Aún asi, no se debería confiar en esto ya que con toda certeza será eliminado en el futuro.

Por defecto, mysqld_safe intenta lanzar un ejecutable llamado mysqld-max si existe, o mysqld en otro caso. Deben tenerse en cuenta las implicaciones de este comportamiento:

- En Linux, el paquete RPM de MySQL-Max se basa en este comportamiento de mysqld_safe. El RPM instala un ejecutable llamado mysqld-max, que causa que mysqld_safe use automáticamente ese ejecutable a partir de ese momento.
- Si se instala una distribucion MySQL-Max que incluye un servidor llamado mysqld-max, y después se actualiza a una version no-Max de MySQL, mysqld_safe todavía intentará ejecutar el viejo servidor mysqld-max. Si se realiza una actualizacion tal, se debe eliminar manualmente el viejo servidor mysqld-max para asegurarse de que mysqld_safe ejecuta el nuevo servidor mysqld.

Para reemplazar el comportamiento por defecto y especificar explícitamente qué servidor se quiere ejecutar, se debe especificar la opción --mysqld o la opción --mysqld-version de mysqld_safe.

Muchas de las opciones de mysqld_safe son las mismas que las opciones de mysqld. Consulte Sección 5.3.1, “Opciones del comando mysqld”.

Todas las opciones específicas de mysqld_safe en la línea de comandos se pasan a mysqld. Si se desea utilizar alguna opción que es específica de mysqld_safe y que mysqld no soporta, no debe especificarse en la línea de comandos. En vez de eso, debe listarse en el grupo [mysqld_safe] de un archivo de opciones. Consulte Sección 4.3.2, “Usar ficheros de opciones”.

mysqld_safe lee todas las opciones de las secciones [mysqld], [server], y [mysqld_safe] de los archivos de opciones. Por compatibilidad con versiones anteriores, también lee las secciones [safe_mysqld], aunque deben renombrarse dichas secciones a [mysqld_safe] en MySQL 5.0.

mysqld_safe soporta las siguientes opciones:

- --help

  Muestra un mensaje de ayuda y finaliza. (Añadido en MySQL 5.0.3)

- --basedir=*ruta*

  La ruta al directorio de instalacion de MySQL.

- --core-file-size=*tamaño*

  El tamaño del archivo de volcado de memoria que mysqld debería ser capaz de crear. El valor de la opción se pasa a ulimit -c.

- --datadir=*ruta*

  La ruta al directorio de datos.

- --defaults-extra-file=*ruta*

  El nombre de un archivo de opciones para ser leído además de los habituales.

- --defaults-file=*ruta*

  El nombre de un archivo de opciones para ser leído en vez de los habituales.

- --ledir=*ruta*

  La ruta a el directorio que contiene el programa mysqld. Se utiliza esta opción para indicar explícitamente la localización del servidor.

- --log-error=*ruta*

Escribir el registro de errores en el archivo dado. Consulte Sección 5.10.1, “El registro de errroes (Error Log)”.

- `--mysqld=nombre_prog`

  El nombre del programa servidor (en el directorio `ledir`) que se quiere ejecutar. Esta opción es necesaria si se utiliza la distribución binaria de MySQL pero el directorio de datos está fuera de la distribución binaria.

- `--mysqld-version=sufijo`

  Esta opción es similar a la opción `--mysqld`, pero se especifica únicamente el sufijo para el nombre del programa servidor. El nombre base se asume que es `mysqld`. Por ejemplo, si se usa `--mysqld-version=max`, `mysqld_safe` inicia el programa en el directorio `ledir`. Si el argumento de `--mysqld-version` está vacio, `mysqld_safe` usa `mysqld` en el directorio `ledir`.

- `--nice=prioridad`

  Se utiliza el programa `nice` para establecer la prioridad del servidor a un valor dado.

- `--no-defaults`

  No leer ningún archivo de opciones.

- `--open-files-limit=número`

  El número de ficheros que `mysqld` debería ser capaz de abrir. El valor de la opción se pasa a `ulimit -n`. Nótese que se necesita iniciar `mysqld_safe` como `root` para que esto funcione correctamente.

- `--pid-file=ruta`

  La ruta al archivo de ID del proceso.

- `--port=num_puerto`

  El número de puerto a usar cuando se esperan conexiones TCP/IP.

- `--socket=ruta`

  El archivo de socket de unix a utilizar para conexiones locales.

- `--timezone=zona`

  Establece la variable de ambiente de zona horaria `TZ` a el valor dado. Consúlte la documentación del sistema operativo para formatos legales de especificación de zonas horarias.

- `--user={nombre_usuario | id_usuario}`

  Ejecuta el servidor `mysqld` como el usuario con nombre *`nombre_usuario`* o el ID numérico de usuario *`id_usuario`*. (``Usuario'' en este contexto se refiere a una cuenta de login del sistema, no a un usuario MySQL incluído en las tablas grant.)

El script `mysqld_safe` está escrito de manera que normalmente puede iniciar un servidor que ha sido instalado tanto desde código fuente o desde una distribución binaria de MySQL, aún cuando típicamente estos tipos de distribuciones instalan el servidor en lugares ligeramente diferentes. (Consulte Sección 2.1.5, “Conformación de la instalación”.) `mysqld_safe` espera que una de las siguientes condiciones sea cierta:

- El servidor y las bases de datos pueden ser encontradas en una ruta relativa al directorio desde el que mysqld_safe es invocado. Para distribuciones binarias, mysqld_safe busca bajo su directorio de trabajo los directorios bin y data. En distribuciones de código fuente, busca los directorios libexec y var. Esta condición debe cumplirse si se ejecuta mysqld_safe desde el directorio de instalación de MySQL (por ejemplo, /usr/local/mysql para una distribución binaria).
- Si el servidor y las bases de datos no pueden encontrarse en una ruta relativa al directorio de trabajo, mysqld_safe intenta localizarlos mediante rutas absolutas. /usr/local/libexec y /usr/local/var son localizaciones típicas. Las localizaciones efectivas se determinan por los valores configurados en la distribución en el momento en que fue creada. Deberían ser correctos si MySQL está instalado en la localización especificada en el momento de la configuración.

Debido a que mysqld_safe trata de encontrar el servidor y las bases de datos de manera relativa a su propio directorio de trabajo, puede instalarse una distribución binaria de MySQL en cualquier lugar, siempre y cuando se ejecute mysqld_safe desde el directorio de instalación de MySQL:

```
shell> cd directorio_instalacion_mysql
shell> bin/mysqld_safe &
```

Si mysqld_safe falla, aún cuando ha sido invocado desde el directorio de instalación de MySQL, se pueden especificar las opciones --ledir y --datadir para indicar los directorios en los que el servidor y las bases de datos están dentro del sistema.

Normalmente, no se debería editar el script mysqld_safe. En vez de ello, ha de configurarse mysqld_safe utilizando opciones de línea de comandos u opciones en la sección [mysqld_safe] de un archivo de opciones my.cnf. En casos aislados, podría ser necesario editar mysqld_safe para que inicie el servidor apropiadamente. No obstante, si se hace esto, la versión modificada de mysqld_safe podría ser sobreescrita si se actualiza la versión de MySQL en el futuro, así que debería hacerse una copia de la versión editada que pudiera reinstalarse.

En NetWare, mysqld_safe es un NetWare Loadable Module (NLM) que ha sido portado desde el script original de Unix. Hace lo siguiente:

1. Ejecuta un número de comprobaciones del sistema y de opciones.
2. Ejecuta comprobaciones sobre tablas MyISAM.
3. Provee de una presencia en pantalla a el servidor MySQL.
4. Inicia mysqld, lo supervisa, y lo reinicia si termina con error.
5. Envía mensajes de error desde mysqld a el archivo *host_name*.err en el directorio de datos.
6. Envía la salida por pantalla de mysqld_safe hacia el archivo *host_name*.safe en el directorio de datos.

5.1.4. El script mysql.server para el arranque del servidor

Las distribuciones de MySQL en Unix incluyen un script llamado mysql.server. Puede usarse en sistemas tales como Linux y Solaris que usan directorios de ejecución estilo System V para arrancar y parar servicios del sistema. También lo usa el Startup Item de Mac OS X para MySQL.

mysql.server puede encontrarse en el directorio support-files bajo el directorio de instalación de MySQL o en el árbol fuente de MySQL.

Si usa el paquete de Linux RPM para el servidor (MySQL-server-*VERSION*.rpm), el script mysql.server se instalará en el directorio /etc/init.d con el nombre mysql. No necesita instalarlo

manualmente. Consulte Sección 2.4, “Instalar MySQL en Linux” para más información acerca de los paquetes RPM para Linux.

Algunos vendedores proporcionan paquetes RPM que instalan un script de instalación bajo nombres diferentes tales como `mysqld`.

Si instala MySQL de una distribución fuente o usando un formato binario de distribución que no instala `mysql.server` automáticamente, puede instalarlo manualmente. Las instrucciones se proporcionan en Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente”.

`mysql.server` lee opciones de las secciones `[mysql.server]` y `[mysqld]` de los ficheros de opciones. (Para compatibilidad con versiones anteriores, también lee las secciones `[mysql_server]`, aunque debe renombrar dichas secciones como `[mysql.server]` cuando use MySQL 5.0.)

5.1.5. El programa `mysqld_multi` para gestionar múltiples servidores MySQL

`mysqld_multi` se utiliza para administrar diversos procesos `mysqld` que esperan conexiones en diferentes archivos socket en Unix y puertos TCP/IP. Puede arrancar o parar servidores, o reportar su estado actual.

El programa busca grupos llamados `[mysqld#]` en `my.cnf` (o en el fichero nombrado por la opción `--config-file`). `#` puede ser cualquier entero positivo. Nos referiremos a este número como el número del grupo de opciones en la siguiente discusión, o *`GNR`* (N. del T.: acrónimo en ingés). Los números de grupo distinguen grupos de opciones de otros y se usan como argumentos para `mysqld_multi` para especificar qué servidores quiere arrancar, parar, u obtener un reporte de estatus. Las opciones listadas en esos grupos son las mismas que usaría en el grupo `[mysqld]` para arranacar `mysqld`. (Consulte, por ejemplo, Sección 2.9.2.2, “Arrancar y parar MySQL automáticamente”.) Sin embargo, cuando use múltiples servidores es necesario que cada uno use su propio valor para opciones tales como el fichero de socket de Unix y el número del puerto TCP/IP. Para más información sobre qué opciones deben ser únicas por servidor en un entorno de múltiples servidores, consulte Sección 5.11, “Ejecutar más de un servidor MySQL en la misma máquina”.

Para invocar `mysqld_multi`, use la siguiente sintaxis:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR]...]
```

`start`, `stop`, y `report` indican qué operaciones desea realizar. Puede realizar la operación designada en un único servidor o en múltiples servidores, dependiendo de la lista *`GNR`* que sigue al nombre de la opción. Si no hay ninguna lista, `mysqld_multi` realiza la operación para todos los servidores en el fichero de opciones.

Cada valor *`GNR`* representa un número de grupo de opciones o rango de números de grupo. El valor debe ser un número al final de un nombre de grupo en el fichero de opciones. Por ejemplo, el *`GNR`* para un grupo llamado `[mysqld17]` es `17`. Para especificar un rango de números, separe el primero y último número por un guión. El valor *`GNR`* `10-13` reperesenta los grupos desde `[mysqld10]` hasta `[mysqld13]`. Multiples grupos o rangos de grupos pueden especificarse en la línea de comandos, separados por comas. No deben haber caráceteres de espacios en blanco (espacios o tabuladores) en la lista+ *`GNR`* ; cualquier cosa después de un carácter de espacio en blanco se ignora.

Este comando arranca un único servidor usando el grupo de opciones `[mysqld17]`:

```
shell> mysqld_multi start 17
```

Este comando para diversos servidores, usando los grupos de opciones `[mysql8]` y del `[mysqld10]` hasta el `[mysqld13]`:

```
shell> mysqld_multi stop 8,10-13
```

Para un ejemplo sobre cómo puede crear un fichero de opciones, use este comando:

```
shell> mysqld_multi --example
```

mysqld_multi soporta las siguientes opciones:

- --config-file=*nombre*

  Especifique el nombre de un fichero de opciones alternativo. Esto afecta a dónde mysqld_multi busca grupos de opciones [mysqld#] . Sin esta opción todas las opciones se leen del fichero habitual my.cnf. La opción no afecta a dónde mysqld_multi lee sus propias opciones, que siempre se toman del grupo [mysqld_multi] en el fichero habitual my.cnf.

- --example

  Muestra un fichero de opciones de ejemplo..

- --help

  Muestra un mensaje de ayuda y sale.

- --log=*nombre*

  Especifica el nombre del fichero de log. Si el fichero existe, la salida de log se añade al mismo.

- --mysqladmin=*prog_name*

  El binario mysqladmin a usar para parar los servidores.

- --mysqld=*prog_name*

  El binario mysqld a usar. Tenga en cuenta que puede especificar mysqld_safe como el valor para esta opción. Las opciones se pasan a mysqld. Sólo asegúrese que tiene el directorio donde se encuentra mysqld en su variable de entorno PATH o fije mysqld_safe.

- --no-log

  Muestra información del log en el stdout en lugar del fichero de log. Por defecto, la salida va al fichero de log.

- --password=*password*

  La constraseña de la cuenta MySQL a usar cuando invoque mysqladmin. Tenga en cuenta que el valor de la contraseña no es opcional para esta opción, no como en otros programas MySQL.

- --silent

  Desactiva los mensajes de advertencia.

- --tcp-ip

  Se conecta a cada servidor MySQL via puerto TCP/IP en lugar del fichero socket Unix. (Si un fichero socket no se encuentra, el servidor puede ejecutarse, pero accesible sólo via puerto TCP/IP.) Por defecto, las conexiones se hacen usando un fichero socket Unix. Esta opción afecta las operaciones stop y report.

- --user=*user_name*

El nombre de usuario de la cuenta MySQL a usar al invocar mysqladmin.

- --verbose

  Es más detallado.

- --version

  Muestra información sobre la versión y sale.

Apuntes acerca de mysqld_multi:

- Asegúrese que la cuenta MySQL usada para parar los servidores mysqld (con el programa mysqladmin) tienen el mismo nombre de usuario y contraseña para cada servidor. También asegúrese que la cuenta tiene el privilegio SHUTDOWN. Si los servidores que quiere administrar tienen distintos nombres de usuario o contraseñas para las cuentas administrativas, puede querer crear una cuenta en cada servidor que tenga el mismo nombre de usuario y contraseña. Por ejemplo, puede inicializar una cuenta común multi_admin ejecutando el siguiente comando en cada servidor:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -proot_password
  mysql> GRANT SHUTDOWN ON *.*
      -> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  ```

  Consulte Sección 5.6.2, “Cómo funciona el sistema de privilegios”. Tiene que hacerlo para cada servidor mysqld. Cambie los parámetros de conexión apropiadamente cuando se conecte a cada uno. Tenga en cuenta que la parte de servidor del nombre de la cuenta debe permitirle conectarse como multi_admin desde el servidor desde el que quiere ejecutar mysqld_multi.

- La opción --pid-file es muy importante si está usando mysqld_safe para arrancar mysqld (por ejemplo, --mysqld=mysqld_safe) Cada mysqld debe tener su propio fichero con el ID de proceso. La ventaja de usar mysqld_safe en lugar de mysqld es que mysqld_safe ``guarda'' su proceso mysqld y lo reinicia si el proceso termina debido a una señal enviada usando kill -9 o por otras razones, tales como un segmentation fault. Por favor, tenga en cuenta que el script mysqld_safe puede requerir que lo arranque desde un lugar determinado. Esto significa que puede tener que cambiar la localización a un cierto directorio antes de ejecutar mysqld_multi. Si tiene problemas arrancando, por favor consulte el script mysqld_safe. Compruebe especialmente las líneas:

  ```
  ----------------------------------------------------------------
  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
  ----------------------------------------------------------------
  ```

  Consulte Sección 5.1.3, “El script de arranque del servidor mysqld_safe”. Los chequeos realizados por estas líneas deberían tener éxito, o podría encontrar problemas.

- El fichero socket de Unix y el puerto TCP/IP deben ser diferentes para cada mysqld.

- Puede usar la opción --user para mysqld, pero para hacerlo debe ejecutar el script mysqld_multi como el usuario root en Unix. Tener la opción en el fichero de opciones no importa; sólo obtiene una advertencia si no es el súper usuario y los procesos mysqld se inician bajo su propia cuenta Unix.

- **Importante**: Asegúrese que el directorio de datos es completamente accesible para todas las cuentas Unix con las que puedea iniciarse el proceso mysqld. *No* use la cuenta root de Unix para ello, a no ser que *sepa* lo que hace.

- **Más importante**: Antes de usar `mysqld_multi` asegúrese de entender el significado de las opciones que se pasan a los servidores `mysqld` y *porqué* quiere tener procesos `mysqld` separados. Cuidado con los peligros de usar múltipes servidores `mysqld` con el mismo directorio de datos. Use diferentes directorios de datos, a no ser que *sepa* lo que hace. Iniciar múltiples servidores con el mismo directorio de datos *no* proporciona mejor rendimiento en un entorno threaded. Consulte Sección 5.11, “Ejecutar más de un servidor MySQL en la misma máquina”.

El siguiente ejemplo muestra como podría inicializar un fichero de opciones para usar con `mysqld_multi`. El primer y quinto grupo `[mysqld#]` se ha omitido intencionadamente del ejemplo para ilustrar que puede tener ``vacíos'' en el fichero de opciones. Esto proporciona una mayor flexibilidad. El order en que los programas `mysqld` arrancan o se paran depende del order en que aparecen en el fichero de opciones.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = multipass

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/hostname.pid2
datadir    = /usr/local/mysql/var2
language   = /usr/local/share/mysql/english
user       = john

[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/hostname.pid3
datadir    = /usr/local/mysql/var3
language   = /usr/local/share/mysql/swedish
user       = monty

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/var4/hostname.pid4
datadir    = /usr/local/mysql/var4
language   = /usr/local/share/mysql/estonia
user       = tonu

[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/var6/hostname.pid6
datadir    = /usr/local/mysql/var6
language   = /usr/local/share/mysql/japanese
user       = jani
```

Consulte Sección 4.3.2, “Usar ficheros de opciones”.

5.2. El gestor de instancias de MySQL

Esta sección discute el uso del MySQL Instance Manager (IM). Este es el demonio que corre en el puerto TCP/IP, el cual provee monitoreo y administración de las instancias del servidor de datos MySQL. MySQL Instance Manager está disponible para sistemas operativos basados en Unix.

MySQL Instance Manager se incluye en las distribuciones de MySQL desde la versión 5.0.3, y puede usarse en lugar del script `mysqld_safe` para arrancar y parar MySQL Server, **incluso desde una máquina remota**. MySQL Instance Manager implementa la funcionalidad (y la mayoría de la sintaxis) del script `mysqld_multi` . Una descripción más detallada de MySQL Instance Manager a continuación.

5.2.1. Arrancar el servidor MySQL con el gestor de instancias MySQL

Normalmente, el servidor de bases de datos MySQL se arranca con el script `mysql.server`, que normalmente reside en el directorio `/etc/init.d/` . En MySQL 5.0.3 este script invoca `mysqlmanager` (el binario de MySQL Instance Manager) para arrancar MySQL. (En versiones previas de MySQL el script `mysqld_safe` se usa con este propósito.) A partir de MySQL 5.0.4 el comportamiento del script de inicio ha cambiado de nuevo para incorporar ambos esquemas de inicialización. En la versión 5.0.4, el scrip de arranque usa el antiguo esquema (invocando `mysqld_safe`) por defecto, pero se puede cambiar la variable `use_mysqld_safe` en el script a `0` (cero) para usar el MySQL Instance Manager para arranacar un servidor.

El comportamiento del Instance Manager en este caso depende de las opciones dadas en el fichero de configuración de MySQL. Si no hay fichero de configuración, el MySQL Instance Manager crea una instancia llamada `mysqld` y trata de arrancarla con los valores por defectos (compilados). Esto significa que el IM no puede adivinar la localización de `mysqld` si no está instalado en la localización por defecto. Si ha instalado MySQL server en una localización no estándar, debe usar un fichero de configuración. Consulte Sección 2.1.5, “Conformación de la instalación”.

Si hay un fichero de configuración, el IM parseará el fichero de configuración en búsqueda de las secciones `[mysqld]` (P.e. `[mysqld]`, `[mysqld1]`, `[mysqld2]`, etc.) Cada una de esas secciones especifica una instancia. Al arrancar, el IM arrancará todas las instancias encontradas. El IM para todas las instancias al cerrar por defecto.

Tenga en cuenta que hay una opción especial `mysqld-path` (`mysqld-path = <path-to-mysqld-binary>`) reconocida sólo por el IM. Use esta variable para que IM conozca dónde reside el binario `mysqld`. También debe inicializar las opciones `basedir` y `datadir` para el servidor.

El típico ciclo de arranque/cierre para un servidor MySQL con el MySQL Instance Manager habilitado es como sigue:

- El MySQL Instance Manager se arranca con el script `/etc/init.d/mysql`.
- El MySQL Instance Manager arranca todas las instancias y las monitoriza.
- Si una instancia de un servidor cae, el MySQL Instance Manager la reinicia.
- Si el MySQL Instance Manager se cierra (por ejemplo con el comando `/etc/init.d/mysql stop`), todas las instancias se apagan con el MySQL Instance Manager.

5.2.2. Conexión al gestor de instancias de MySQL y creación de cuentas de usuario

La comunicación con el MySQL Instance Manager se hace usando el protocolo cliente-servidor de MySQL. Con el mismo, puede conectarse al IM usando el program cliente estándar `mysql` , así como con la API de C MySQL. El IM soporta la versión del protocolo cliente- servidor MySQL usada por las herramientas de cliente y bibliotecas distribuidas a partir de la versión mysql-4.1.

El IM almacena su información de usuario en un fichero de contraseñas. La localización por defecto para el fichero de contraseñas es `/etc/mysqlmanager.passwd`

Las entradas para las contraseñas se parecen a las siguiente:

```
petr:*35110DC9B4D8140F5DE667E28C72DD2597B5C848
```

Para generar una entrada así debe invocarse IM con la opción `--passwd` . Entonces puede redirigir la salida a `/etc/mysqlmanager.passwd` para añadir un nuevo usuario. Un comando de ejemplo a continuación.

```
./mysqlmanager --passwd >> /etc/mysqlmanager.passwd
Creating record for new user.
Enter user name: mike
Enter password: <password>
Re-type password: <password>
```

La siguiente línea se añade a `/etc/mysqlmanager.passwd`:

```
mike:*00A51F3F48415C7D4E8908980D443C29C69B60C9
```

Si no hay entradas en el fichero `/etc/mysqlmanager.passwd` no puede conectarse al IM.

5.2.3. Opciones de los comandos del gestor de instancias MySQL

El MySQL Instance Manager soporta varias opciones de línea de comando. Una breve lista está disponible ejecutando el comando `./mysqlmanager --help` . Los siguientes comandos están disponibles:

- `-?, --help`

  Muestra la ayuda y sale..

- `--log=name`

  Ruta al fichero de log del IM. Se usa con la opción `--run-as-service`.

- `--pid-file=name`

  Fichero Pid a usar. Por defecto es `mysqlmanager.pid`.

- `--socket=name`

  Fichero socket a usar por las conexiones. Por defecto es `/tmp/mysqlmanager.sock`.

- `-P, --passwd`

  Prepara entrada para fichero passwd y salir.

- `--bind-address=name`

  Dirección enlazada para usar en conexiones.

- `--port=#`

  Número de puerto para usar en conexiones (número de puerto por defecto, asignado por la IANA, es el 2273).

- `--password-file=name`

Busca los usuarios y contraseñas para el Instance Manger aquí. El fichero por defecto es `/etc/mysqlmanager.passwd`.

- `--default-mysqld-path=name`

  Dónde buscar el binario para el MySQL Server si no se proporciona un path en la sección de instancias. Ejemplo: `default-mysqld-path = /usr/sbin/mysqld`.

- `--monitoring-interval=#`

  Intervalo en segundos para monitorizar instancias. El IM tratará de conectar a cada una de las instancias monitorizadas para comprobar si están vivas / no colgadas. En caso de un fallo el IM realizará varios (de hecho muchos) intentos de reiniciar la instancia. Puede desactivar este comportamiento para instancias particulares con la opción `nonguarded` en la sección de instancia apropiada. Si no se proporciona ningún valor, se usan 20 segundos por defecto.

- `--run-as-service`

  Demoniza y arranca el proceso ángel. El proceso ángel es simple y difícil de que falle. Reinicia el IM en caso de fallo.

- `--user=name`

  Nombre de usuario para arrancar y ejecutar `mysqlmanager`. Se recomienda ejecutar `mysqlmanager` bajo la misma cuenta de usuario usada para ejectuar el servidor `mysqld`.

- `-V, --version`

  Muestra información de la versión y sale.

5.2.4. Ficheros de configuración del gestor de instancias de MySQL

El Instance Manager usa el fichero estándar `my.cnf`. Usa la sección `[manager]` para leer opciones para sí mismo y la sección `[mysqld]` para crear instancias. La sección `[manager]` contiene algunas de las opciones listadas anteriormente. Un ejemplo de la sección `[manager]` a continuación:

```
# 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
```

Las secciones de instancias especifican opciones dadas a cada instancia al arrancar. La mayoría son comunes con las opciones de MySQL Server, pero hay algunas específicas para el IM:

- `mysqld-path = <path-to-mysqld-binary>`

  La ruta al binario del servidor `mysqld`.

- `shutdown-delay = #`

  Número de segundos que IM debe esperar para que una instancia se cierre. Por defecto son 35 segundos. Cuando acaba el tiempo, IM asume que la instancia está colgada y trata de hacer un `kill -9` . Si usa InnoDB con tablas grandes, debe incrementar este valor.

- `nonguarded`

  Esta opcion debe activarse si se quiere desactivar la funcionalidad de monitoreo de IM para una instancia concreta.

Diversas secciones de instancias de ejemplo a continuación.

```
[mysqld]
mysqld-path=/usr/local/mysql/libexec/mysqld
socket=/tmp/mysql.sock
port=3307
server_id=1
skip-stack-trace
core-file
skip-bdb
log-bin
log-error
log=mylog
log-slow-queries

[mysqld2]
nonguarded
port=3308
server_id=2
mysqld-path= /home/cps/mysql/trees/mysql-4.1/sql/mysqld
socket     = /tmp/mysql.sock4
pid-file   = /tmp/hostname.pid4
datadir= /home/cps/mysql_data/data_dir1
language=/home/cps/mysql/trees/mysql-4.1/sql/share/english
log-bin
log=/tmp/fordel.log
```

5.2.5. Los comandos que reconoce el gestor de instancias de MySQL

Una vez que se ha inicializado un fichero de contraseñas para el MySQL Instance Manager y que el IM está ejecutándose, puede conectarse al mismo. Puede usar la herramienta cliente `mysql` para conectar a través de la API MySQL . A continuación se muestra la lista de comandos que el MySQL Instance Manager acepta actualmente, con ejemplos.

- `START INSTANCE <instance_name>`

  Este comando intenta arrancar una instancia:

```
mysql> START INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- `STOP INSTANCE <instance_name>`

  Esto trata de parar una instancia:

```
mysql> STOP INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- `SHOW INSTANCES`

  Muestra los nombres de todas las intancias cargadas:

```
mysql> show instances;
```

```
+---------------+---------+
| instance_name | status  |
+---------------+---------+
| mysqld3       | offline |
| mysqld4       | online  |
| mysqld2       | offline |
+---------------+---------+
3 rows in set (0,04 sec)
```

- `SHOW INSTANCE STATUS <instance_name>`

  Muestra el estado e información de la versión de la instancia seleccionada:

```
mysql> SHOW INSTANCE STATUS mysqld3;
+---------------+--------+---------+
| instance_name | status | version |
+---------------+--------+---------+
| mysqld3       | online | unknown |
+---------------+--------+---------+
1 row in set (0.00 sec)
```

- `SHOW INSTANCE OPTIONS <instance_name>`

  Muestra las opciones usadas por una instancia:

```
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 |
+---------------+-----------------------------------------------------+
7 rows in set (0.01 sec)
```

- `SHOW <instance_name> LOG FILES`

  El comando poroporciona un listado de todos los ficheros de log usados por la instancia. El conjunto resultado contiene el path al fichero de log y al fichero de configuración (i.e. `log=/var/mysql.log`), el IM trata de adivinar su ubicación. Si IM no es capaz de localizar el fichero de logs, debe especificarlo explícitamente.

```
mysql> SHOW mysqld LOG FILES;
+-------------+-----------------------------------+----------+
| Logfile     | Path                              | Filesize |
+-------------+-----------------------------------+----------+
| 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     |
+-------------+-----------------------------------+----------+
3 rows in set (0.01 sec)
```

- `SHOW <instance_name> LOG {ERROR | SLOW | GENERAL} size[,offset_from_end]`

  Este comando recibe una porción del fichero de log especificado. Ya que la mayoría de usuarios están interesados en los últimos mensajes de log, el parámetro `size` define el número de bytes que quiere recibir empezando por el final del log. Puede recibir datos del medio del fichero de log especificando el

parámetro opcioneal `offset_from_end` . El siguiente ejemplo recibe 21 bytes de datos, empezando 23 bytes desde el final del fichero de log y acabando 2 bytes al final del fichero de log.:

```
mysql> SHOW mysqld LOG GENERAL 21, 2;
+--------------------+
| Log                |
+--------------------+
| using password: YES |
+--------------------+
1 row in set (0.00 sec)
```

- `SET instance_name.option_name=option_value`

  Este comando edita la configuración de la instancia especificada para cambiar/añadir opciones a la instancia. El IM asume que el fichero de configuración está localizado en `/etc/my.cnf`. Debe comprobar que el fichero existe y que tiene los permisos apropiados.

  ```
  mysql> SET mysqld2.port=3322;
  Query OK, 0 rows affected (0.00 sec)
  ```

  Los cambios hecho en el fichero de configuración no tendrán efecto hasta reiniciar el servidor MySQL. Además, estos cambios no se guardan en la cache local de configuración del Instance Manager hasta que se ejecuta un comando `FLUSH INSTANCES`.

- `UNSET instance_name.option_name`

  Este comando elimina una opción de un fichero de configuración de una instancia.

  ```
  mysql> UNSET mysqld2.port;
  Query OK, 0 rows affected (0.00 sec)
  ```

  Los cambios hecho en el fichero de configuración no tendrán efecto hasta reiniciar el servidor MySQL. Además, estos cambios no se guardan en la cache local de configuración del Instance Manager hasta que se ejecuta un comando `FLUSH INSTANCES`.

- `FLUSH INSTANCES`

  Este comando fuerza a IM a releer el fichero de configuración y a refrescar estructuras internas. Este comando debe ejectuarse tras editar el fichero de configuración. Este comando no reinicia las instancias:

  ```
  mysql> FLUSH INSTANCES;
  Query OK, 0 rows affected (0.04 sec)
  ```

5.3. Configuración del servidor MySQL

Esta sección discute los siguientes tópicos de MySQL Server:

- Opciones de arranque que soporta el servidor
- Cómo configurar el modo SQL del servidor
- Variables de sistema del servidor
- Variables de estado del servidor

5.3.1. Opciones del comando `mysqld`

Cuando arranca el servidor mysqld , puede especificar opciones de programa usando cualquiera de los métodos descritos en Sección 4.3, “Especificar opciones de programa”. Los métodos más comunes son proporcionar opciones en un fichero de opciones o por línea de comandos. Sin embargo, en la mayoría de los casos es deseable asegurar que el servidor usa las mismas opciones cada vez que se ejecuta. La mejor manera de asegurarlo es listarlas en un fichero de opciones. Consulte Sección 4.3.2, “Usar ficheros de opciones”.

mysqld lee opciones de los grupos [mysqld] y [server]. mysqld_safe lee opciones de los grupos [mysqld], [server], [mysqld_safe], y [safe_mysqld]. mysql.server lee opciones de los grupos [mysqld] y [mysql.server] . Un servidor MySQL incrustrado normalmente lee opciones de los grupos [server], [embedded], y [*xxxxx*_SERVER] , donde *xxxxx* es el nombre de la aplicación en la que el servidor está incrustado.

mysqld acepta varias opciones de línea de comando. Para una breve lista, ejecute mysqld --help. Para ver la lista completa, use mysqld --verbose --help.

La siguiente lista muestra algunas de las opciones de servidor más comunes. Opciones adicionales se describen en los siguientes links:

- Opciones que afectan la seguridad: Consulte Sección 5.5.3, “Opciones de arranque para mysqld relacionadas con la seguridad”.
- Opciones relacionadas con SSL: Consulte Sección 5.7.7.5, “Opciones relativas a SSL”.
- Opciones de control del log binario: Consulte Sección 5.10.3, “El registro binario (Binary Log)”.
- Opciones relacionadas con replicación: Consulte Sección 6.8, “Opciones de arranque de replicación”.
- Opciones específicas a motores de almacenamiento particulares: Consulte Sección 14.1.1, “Opciones de arranque de MyISAM”, Sección 14.4.3, “Opciones de arranque de BDB”, and Sección 15.4, “Opciones de arranque de InnoDB”.

También puede cambiar los valores de una variable de sistema del servidor usando el nombre de variable como opción, tal y como se describe más tarde en esta sección.

- --help, -?

  Muestra un mensaje de ayuda corto y sale. Use las opciones --verbose y --help simultáneamente para ver el mensaje entero.

- --allow-suspicious-udfs

  Este opción controla si las funciones definidas por el usuario que sólo tienen un símbolo xxx para la función principal pueden cargarse. Por defecto, la opción está desactivada y sólo UDFs que tengan al menos un símbolo auxiliar pueden cargarse. Esto previene intentos de cargar funciones de ficheros con objetos compartidos que no contengan UDFs legítimos. En las series de MySQL 5.0 está opción se añadió en la versión 5.0.3. Consulte Sección 27.2.3.6, “Precauciones de seguridad en funciones definidas por usuarios”.

- --ansi

  Usa sintaxis SQL (ANSI) en lugar de sintaxis MySQL.Consulte Sección 1.7.3, “Ejecutar MySQL en modo ANSI”. Para un control más preciso sobre el modo SQL del servidor, use la opción --sql-mode.

- --basedir=*path*, -b *path*

El path al directorio de instalación de MySQL. Todas las rutas se resuelven normalmente relativas a ésta.

- `--bind-address=IP`

  La dirección IP a ligar.

- `--console`

  Escribe los mensajes de error por `stderr` y `stdout` incluso si `--log-error` está especificado. En Windows, `mysqld` no cierra la pantalla de consola si se usa esta opción.

- `--character-sets-dir=path`

  El directorio donde los conjuntos de caracteres están instalados. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--chroot=path`

  Pone el servidor `mysqld` en un entorno cerrado durante el arranque usando la llamada de sistema `chroot()`. Esta es una medida de seguridad recomendada. Tenga en cuenta que el uso de esta opción limita de alguna manera `LOAD DATA INFILE` y `SELECT ... INTO OUTFILE`.

- `--character-set-server=charset`

  Usa `charset` como el conjunto de caracteres por defecto del servidor. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--core-file`

  Escribe un fichero core si `mysqld` muere. Para algunos sistemas, también puede especificar la opción `--core-file-size` en `mysqld_safe`. Consulte Sección 5.1.3, “El script de arranque del servidor `mysqld_safe`”. Tenga en cuenta que en algunos sistemas como Solaris, no obtiene un fichero core si está usando la opción `--user`.

- `--collation-server=collation`

  Usa `collation` como la colación del servidor por defecto. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--datadir=path, -h path`

  La ruta al directorio de datos.

- `--debug[=debug_options], -# [debug_options]`

  Si MySQL está configurado con `--with-debug`, puede usar esta opción para obtener un fichero de traza de qué está haciendo `mysqld` . La cadena de caracteres `debug_options` a menudo es `'d:t:o,file_name'`. Consulte Sección D.1.2, “Crear ficheros de traza”.

- (*DEPRECATED*) `--default-character-set=charset`

  Usa `charset` como el conjunto de caracteres por defecto. Esta opcíon está obsoleta a favor de `--character-set-server`. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--default-collation=collation`

Usa *collation* como colación por defecto. Esta opción está obsoleta a favor de --collation-server. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- --default-storage-engine=*type*

  Esta opción es un sinónimo para --default-table-type.

- --default-table-type=*type*

  Cambia el valor por defecto de tipo de tablas. Consulte Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

- --default-time-zone=*type*

  Cambia la zona horaria del servidor. Esta opción cambia la variable global de sistema time_zone . Si no se da esta opción, la zona horaria por defecto es la misma que la del sistema (dada por el valor de la variable de sistema system_time_zone.

- --delay-key-write[= OFF | ON | ALL]

  Coómo debe usarse la opción DELAYED KEYS. La escritura retardada de claves provoca que los buffers de claves no se vuelquen entre escrituras para tablas MyISAM . OFF desactiva escritura de claves retardada. ON activa escritura de claves reatardada para aquellas tablas creadas con la opción DELAYED KEYS . ALL retarda la escritura de claves para todas las tablas MyISAM . Consulte Sección 7.5.2, “Afinar parámetros del servidor”. Consulte Sección 14.1.1, “Opciones de arranque de MyISAM”.

  Nota: Si asigna a esta variable el valor ALL, no debe usar tablas MyISAM de otro programa (como de otro servidor MySQL o con myisamchk) cuando una tabla esté en uso. Hacerlo provoca corrupción de los índices.

- --des-key-file=*file_name*

  Lee las claves por defecto usadas por DES_ENCRYPT() y DES_DECRYPT() de este fichero.

- --enable-named-pipe

  Activa el soporte para named pipes. Esta opción se aplica sólo en Windows NT, 2000, XP y 2004, y sólo pueden usarse con servidores mysqld-nt y mysqld-max-nt que soporten conexiones para named pipes.

- --exit-info[=*flags*], -T [*flags*]

  Esta es una máscara de bits de diferentes flags y que puede usar para debugar el servidor mysqld . No use esta opción a no ser que sepa *exactamente* lo que hace!

- --external-locking

  Activa bloqueo del sistema. Tenga en cuenta que si usa esta opción en un sistema en que lockd no funcione plenamente (comom en Linux), es fácil para mysqld caer en un deadlock. Esta opción préviamente era --enable-locking.

  Nota: Si usa esta opción para permitir actualizaciones en tablas MyISAM en cualquier proceso MySQL, debe asegurarse que las siguiente condiciones se satisfacen:

  - No debe usar la caché de consultas para consultas que usen tablas actualizadas por otros procesos.
  - No debe usar --delay-key-write=ALL o DELAY_KEY_WRITE=1 en ninguna tabla compartida.

La forma más fácil de asegurar esto es usar siempre `--external-locking` junto a `--delay-key-write=OFF --query-cache-size=0`.

(Esto no se hace por defecto ya que en muchas configuraciones es útil tener una mezcla de las opciones anteriores.)

- `--flush`

  Escribe todos los cambios a disco después de cada comando SQL. Normalmente MySQL escribe todos los cambios en disco después de cada comando SQL y deja al sistema operativo la sincronización con el disco. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”.

- `--init-file=`*`file`*

  Lee comandos SQL de este fichero al arrancar. Cada comando debe ser de una sola línea y no debe incluir comentarios.

- `--innodb-safe-binlog`

  Añade garantía de consistencia entre el contenido de las tablas `InnoDB` y el log binario. Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

- `--language=`*`lang_name`*`, -L `*`lang_name`*

  Mensajes de error del cliente en el idioma dado. *`lang_name`* puede darse como el nombre del idioma o como la ruta al directorio donde los ficheros de idioma están instalados. Consulte Sección 5.9.2, “Escoger el idioma de los mensajes de error”.

- `--large-pages`

  Algunas arquitecturas hardware o de sistemas operativos soportan paginación de memoria mayor a la que hay por defecto (normalmente 4 KB). La implementación de este soporte depende del hardware subyacente y del SO. Aplicaciones que necesiten mucha memoria pueden obtener mejoras de rendimiento usando páginas grandes gracias a reducir los fallos en el Translation Lookaside Buffer (TLB).

  Actualmente, MySQL soporta sólo implementaciones en Linux de soporte para páginas grandes (que se llama HugeTLB en Linux). Tenemos planes de extender este soporte a FreeBSD, Solaris y posiblemente otras plataformas.

  Antes de poder usar páginas grandes en Linux, es necesario configurar el pool de memoria de HugeTLB. Como referencia, consulte el fichero `hugetlbpage.txt` en la fuente del kernel Linux.

  Esta opción está desactivada por defecto. Se añadió en MySQL 5.0.3.

- `--log[=`*`file`*`], -l [`*`file`*`]`

  Log de conexiones y consultas en este fichero. Consulte Sección 5.10.2, “El registro general de consultas”. Si no especifica un nombre de fichero, MySQL usa *`host_name`*`.log` como nombre de fichero.

- `--log-bin=[`*`file`*`]`

  El fichero de logs binario. Loguea todas las consultas que cambian datos en este fichero. Se usa para copias de seguridad y replicación. Consulte Sección 5.10.3, “El registro binario (Binary Log)”. Se recomienda especificar un nombre de fichero (consulte Sección A.8.4, “Cuestiones abiertas en MySQL”

para la razón) en caso contrario MySQL usa *host_name*-bin como el nombre base para el fichero de logs.

- --log-bin-index[=*file*]

  El fichero índice para log binario. Consulte Sección 5.10.3, “El registro binario (Binary Log)”. Si no especifica un nombre de fichero, y si no especifica uno en --log-bin, MySQL usa *host_name*-bin.index como el nombre de fichero.

- --log-bin-trust-routine-creators[={0|1}]

  Sin argumento o un argumento de 1, esta opción inicializa la variable de sistema log_bin_trust_routine_creators a 1. Con un argumento de 0, esta opción actualiza la variable de sistema a 0.. log_bin_trust_routine_creators afecta cómo MySQL fuerza las restricciones en la creación de rutinas almacenadas. Consulte Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

  Esta opción se añadió en MySQL 5.0.6.

- --log-error[=*file*]

  Mensajes de error y de arranque en este fichero. Consulte Sección 5.10.1, “El registro de errroes (Error Log)”. Si no especifica un nombre de fichero, MySQL usa *host_name*.err como nombre de fichero. Si el nombre de fichero no tiene extensión, una extensión .err se añade al nombre.

- --log-isam[=*file*]

  Loguea todos los cambios ISAM/MyISAM en este fichero (usado sólo al debugar ISAM/MyISAM).

- (*DEPRECATED*) --log-long-format

  Loguea información extra a cualquiera de los logs que haya activados (log de actualización, log de consultas lentas y log binario). Por ejemplo, se añade el nombre de usuario y tiempo de la consulta para todas las consultas. Esta opción está obsoleta en MySQL 5.0, y ahora representa el comportamiento por defecto para logueo. (Consulte la descripción para --log-short-format.) La opción --log-queries-not-using-indexes está disponible para los propósitos de loguear consultas que no usan índices en el log de consultas lentas

- --log-queries-not-using-indexes

  Si usa esta opción con --log-slow-queries, las consultas que no usan índices también se loguean en el log de consultas lentas. Consulte Sección 5.10.4, “El registro de consultas lentas (Slow Query Log)”.

- --log-short-format

  Loguea menos información en cualquiera de los logs activados (log de actualización, log binario y log de consultas lentas). Por ejemplo, el nombre de usuario y el tiempo en que se produce la consulta no se guardan para las consultas.

- --log-slow-admin-statements

  Loguea comandos lentos administrativos tales como OPTIMIZE TABLE, ANALYZE TABLE, y ALTER TABLE en el log de consultas lentas.

- --log-slow-queries[=*file*]

Loguea todas las consultas que han tardado más de long_query_time segundos en ejecutarse en este fichero.. See Sección 5.10.4, “El registro de consultas lentas (Slow Query Log)”. Consulte las descripciones de las opciones --log-long-format y --log-short-format para más detalles.

- --log-warnings, -W

  Muestra advertencias tales como Aborted connection... en el log de errores. Se recomienda activar esta opción, por ejemplo, si usa replicación (obtiene mayor información acerca de lo que está ocurriendo, tal como mensajes acerca de fallos de red y reconexiones). Esta opción está activada por defecto en MySQL 5.0; para desactivarla, use --skip-log-warnings. Las conexiones abortadas no se loguean en el log de errores a no ser que el valor sea mayor que 1. Consulte Sección A.2.10, “Errores de comunicación y conexiones abortadas”.

- --low-priority-updates

  Operaciones que modifiquen la tabla (INSERT, REPLACE, DELETE, UPDATE) tiene una prioridad inferior a las selecciones. También puede hacerse vía {INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ... para bajar la prioridad de sólo una consulta, o con SET LOW_PRIORITY_UPDATES=1 para cambiar la prioridad en un thread. Consulte Sección 7.3.2, “Cuestiones relacionadas con el bloqueo (locking) de tablas”.

- --memlock

  Bloquea el proceso mysqld en memoria. Funciona con sistemas tales como Solaris que soportan la llamada de sistema mlockall() . Esto puede ser útil si tiene un problema en el que el sistema operativo cause que mysqld realice swap en el disco. Tenga en cuenta que el uso de esta operación requiere que ejecute el servidor como root, lo que normalmente no es una buena idea por razones de seguridad.

- --myisam-recover [=*option*[,*option*...]]]

  Inicializa el modo de recuperación en el motor de almacenamiento MyISAM . El valor para esta opción es cualquier combinación de los valores DEFAULT, BACKUP,FORCE, o QUICK. Si especifica múltiples valores, sepárelos con comas. Puede usarse como valor "" para desactivar esta opción. Si esta opción se usa, mysqld, cuando abre una tabla MyISAM , comprueba si la tabla tiene marca de que haya fallado o no se haya cerrado correctamente. (La última opción sólo funciona si está ejecutando con la opción --skip-external-locking.) En este caso, mysqld realiza una comprobación sobre la tabla. Si la tabla está corrupta, mysqld intenta repararla.

  La siguientes opciones afectan el funcionamiento de la reparación:

Opción	Descrición
DEFAULT	Lo mismo que no dar ninguna opción --myisam-recover.
BACKUP	Si el fichero de datos ha cambiado durante la recuperación, guarda una copia de seguridad del fichero *tbl_name*.MYD como *tbl_name-datetime*.BAK.
FORCE	Ejecuta una recuperación incluso si perdemos más de un registro del fichero .MYD.
QUICK	No comprueba los registros en la tabla si no hay ningún bloque borrado.

  Antes de reparar una tabla automáticamente, MySQL añade una nota acerca de ello en el log de errores. Si quiere poder recuperarse de la mayoría de problemas sin intervención por parte del usuario, debe usar las opciones BACKUP,FORCE. Esto fuerza la reparación de una tabla incluso si algunos registros se borran durante el proceso, pero guarda el antiguo fichero de datos como una copia de seguridad, de forma que posteriormente pueda examinar qué ocurrió.

- --ndb-connectstring=*connect_string*

  Cuando use el motor de almacenamiento NDB , es posible determinar el servidor de administración que distribuye la configuración del cluster mediante la inicialización de la opción de la cadena de conexión. Consulte Sección 16.4.4.2, “El connectstring de MySQL Cluster” para la sintaxis.

- --ndbcluster

  Si el binario incluye soporte para el motor de almacenamiento NDB Cluster , la opción por defecto de desactivar el soporte para MySQL Cluster puede ignorarse usando esta opción. Consulte Capítulo 16, *MySQL Cluster*.

- --old-passwords

  Fuerza al servidor a generar un pequeño hash de contraseñas (pre-4.1) para nuevas contraseñas. Esto es útil para compatibilidad con antiguos programas cliente. Consulte Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1”.

- --one-thread

  Sólo usa un thread (para debugar bajo Linux). Esta opción está disponible sólo si el servidor está compilado con la opción de debugar. Consulte Sección D.1, “Depurar un servidor MySQL”.

- --open-files-limit=*count*

  Para cambiar el número de descriptores de fichero disponibles para mysqld. Si no se inicializa o se asigna otro valor a 0, entonces mysqld usa este valor para reservar descriptores de fichero para usar con setrlimit(). Si el valor es 0, entonces mysqld reserva max_connections*5 o max_connections + table_cache*2 (lo que sea mayor) número de ficheros. Puede tratar de incrementar este valor si mysqld le muestra el error "Too many open files."

- --pid-file=*path*

  El path al fichero con el ID de proceso usado por mysqld_safe.

- --port=*port_num*, -P *port_num*

  El número de puerto que se usa para escuchar conexiones TCP/IP.

- --safe-mode

  Omite algunas etapas de optimización.

- (*DEPRECATED*) --safe-show-database

  Consulte Sección 5.6.3, “Privilegios de los que provee MySQL”.

- --safe-user-create

  Si está activado, un usuario no puede crear nuevos usuarios con el comando GRANT , si el usuario no tiene el privilegio INSERT para la tabla mysql.user o cualquiera de sus columnas.

- --secure-auth

  No permite autenticación para cuentas que usan las contraseñas antiguas (pre-4.1).

- --shared-memory

Activa conexiones mediante memoria compartida con clientes locales. Esta opción sólo está disponible en Windows.

- `--shared-memory-base-name=name`

  El nombre a usar en conexiones mediante memoria compartida. Esta opción sólo está disponible en Windows.

- `--skip-bdb`

  Deshabilita el motor de almacenamiento `BDB` . Esto ahorra memoria y puede acelerar algunas operaciones. No use esta opción si necesita tablas `BDB`.

- `--skip-concurrent-insert`

  Desactiva la habilidad de seleccionar e insertar al mismo tiempo en tablas `MyISAM` . (Esto sólo debe usarse si cree que ha encontrado un bug en esta funcionalidad.)

- `--skip-external-locking`

  No usa sistema de bloqueo. Para usar `myisamchk`, debe apagar el servidor. (Consulte Sección 1.4.3, “Estabilidad de MySQL”.) Para evitar este requerimiento, use `CHECK TABLE` y `REPAIR TABLE` desde el MySQL Monitor para comprobar y reparar tablas `MyISAM`.

- `--skip-grant-tables`

  Esta opción hace que el servidor no use el sistema de privilegios en absoluto. Esta opción da a cualquiera que tenga acceso al servidor *acceso ilimitado* a *todas las bases de datos*. Puede hacer que un servidor en ejecución empiece a usar las tablas de privilegios de nuevo ejecutando `mysqladmin flush-privileges` o el comando `mysqladmin reload` desde una consola de sistema, o utilizando el comando MySQL `FLUSH PRIVILEGES`.

- `--skip-host-cache`

  No use la cache interna de nombres de servidor para una resolución rápida de nombre-a-IP. En lugar de ello, interrogue al servidor DNS cada vez que un cliente se conecte. Consulte Sección 7.5.6, “Cómo usa MySQL las DNS”.

- `--skip-innodb`

  Deshabilita el motor de almacenamiento `InnoDB` . Esto ahorra memoria y espacio en disco y puede acelerar algunas operaciones. No use esta opción si necesita tablas `InnoDB` .

- `--skip-name-resolve`

  No resuelva los nombres de servidor cuando compruebe las conexiones de clientes. Use sólo números IP. Si usa esta opción, todos los valores de la columna `Host` en la tabla de privilegios deben ser números IP o `localhost`. Consulte Sección 7.5.6, “Cómo usa MySQL las DNS”.

- `--skip-ndbcluster`

  Deshabilite el motor de almacenamiento `NDB Cluster` . Este es el comportamiento por defecto para los binarios compilados con el soporte para el motor de almacenamiento de `NDB Cluster` , lo que significa que el sistema reserva memoria y otros recursos para este motor de almacenamiento sólo si `--skip-ndbcluster` está subeditado explícitamente por la opción `--ndbcluster` . Consulte Sección 16.4.3, “Rápido montaje de prueba de MySQL Cluster” para un ejemplo de uso.

- `--skip-networking`

No escucha conexiones TCP/IP en absoluto. Toda interacción con mysqld debe hacerse via named pipes o memoria compartida (en Windows) o ficheros socket en Unix. Esta opción se recomienda encarecidamente en sistemas donde sólo se permitan clientes locales. Consulte Sección 7.5.6, “Cómo usa MySQL las DNS”.

- --standalone

  Sólo para sistemas basados en Windows-NT; le dice al servidor MySQL que no se ejecute como servicio.

- --symbolic-links, --skip-symbolic-links

  Activa o desactiva el soporte para links simbólicos. Esta opción tiene diferentes efectos en Windows y Unix:

  - En Windows, habilitar links simbólicos permite establecer un link simbólico a un directorio de base de datos mediante la creación de un fichero directory.sym que contiene el path al directorio real. Consulte Sección 7.6.1.3, “Usar enlaces simbólicos para bases de datos en Windows”.

  - En Unix, habilitar links simbólicos significa que puede lincar un fichero de índice o de datos MyISAM a otro directorio con las opciones INDEX DIRECTORY o DATA DIRECTORY del comando CREATE TABLE . Si borra o renombra la tabla, el fichero cuyo link simbólico apunta a dicha tabla también se borra o renombra. Consulte Sección 13.1.5, “Sintaxis de CREATE TABLE”.

- --skip-safemalloc

  Si MySQL está configurado con --with-debug=full, todos los programs MySQL chequean para desbordamientos de memoria durante cada reserva y liberación de memoria. Este chequeo es muy lento, de forma que puede desactivarlo cuando no lo necesite mediante la opción --skip-safemalloc .

- --skip-show-database

  Con esta opción, el comando SHOW DATABASES sólo se permite a usuarios que tengan el privilegio SHOW DATABASES , y el comando muestra todos los nombres de bases de datos. Sin esta opción, SHOW DATABASES se permite a todos los usuarios, pero muestra cada nombre de base de datos sólo si el usuario tiene el privilegio SHOW DATABASES o algún privilegio sobre la base de datos.

- --skip-stack-trace

  No escribe la traza de la pila. Esta opción es útil cuando está ejecutando mysqld con un debugger. En algunos sistemas, puede usar esta opción para obtener un fichero core. Consulte Sección D.1, “Depurar un servidor MySQL”.

- --skip-thread-priority

  Desactiva el uso de prioridades de threads para un mejor tiempo de respuesta.

- --socket=*path*

  En Unix, esta opción especifica el fichero socket de Unix para usar en conexiones locales. El valor por defecto es /tmp/mysql.sock. En Windows, la opción especifica el nombre de pipe a usar en conexiones locales que usen una named pipe. El valor por defecto es MySQL.

- --sql-mode=*value*[,*value*[,*value*...]]

  Inicializa el modo SQL para MySQL. Consulte Sección 5.3.2, “El modo SQL del servidor”.

- --temp-pool

  Esta opción provoca que la mayoría de ficheros temporales creados por el servidor usen un pequeño conjunto de nombes, en lugar de un nombre único para cada fichero nuevo. Esto evita el problema del kernel de Linux de crear muchos nuevos ficheros con nombres diferentes. Con el antiguo comportamiento, parece que a Linux le ``falte'' memoria, ya que se está reservando en la entrada del cache del directorio en lugar de la cache de disco.

- --transaction-isolation=*level*

  Especifica el nivel de aislamiento de las transacciones, que puede ser READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, o SERIALIZABLE. Consulte Sección 13.4.6, “Sintaxis de SET TRANSACTION”.

- --tmpdir=*path*, -t *path*

  El path del directorio a usar para crear ficheros temporales. Puede ser útil si el directorio por defecto /tmp reside en una partición que sea demasiado pequeña para alojar tablas temporales. En MySQL 5.0, esta opción acepta diversas rutas que se usan con una política round-robin. Las rutas deben estar separados por comas en Unix (':') y puntos y comas (';') en Windows, NetWare, and OS/2. Si MySQL server está actuando como un esclavo de replicación, no debe hacer que --tmpdir apunte a un directorio en un sistema de ficheros basado en memoria o a un directorio que se limpie al reiniciar el servidor. Un esclavo de replicación necesita que sobrevivan algunos de sus ficheros temporales tras un reinicio de la máquina de forma que pueda replicar tablas temporales u operaciones LOAD DATA INFILE. Si los ficheros en el directorio temporal se pierden al reiniciar el servidor, la replicación falla.

- --user={*user_name* | *user_id*}, -u {*user_name* | *user_id*}

  Ejecuta el servidor mysqld como si el usuario tuviese el nombre *user_name* o el ID de usuario *user_id*. (``Usuario'' en este contexto se refiere a la cuenta de login del sistema, no a un usuario MySQL listado en las tablas de permisos.)

  Esta opción es *obligatoria* cuando se ejecute mysqld como root. El servidor cambia su ID de usuario durante la secuencia de arranque, causando que se ejecute como ese usuario en particular en lugar de root. Consulte Sección 5.5.1, “Guía de seguridad general”.

  Para evitar un posible agujero de seguridad donde un usuario añade una opción --user=root a un fichero my.cnf (por lo tanto, causando que el servidor se ejecute como root), mysqld usa sólo la primera opción --user especificada y produce una adevertencia si hay múltiples opciones --user . Las opciones en /etc/my.cnf y $MYSQL_HOME/my.cnf se procesan antes que las opciones de línea de comandos, así que se recomiendo que ponga una opción --user en /etc/my.cnf y especifique un valor distinto a root. La opción en /etc/my.cnf se encuentra antes que cualquier otra opción --user , lo que asegura que el servidor se ejecute como un usuario distinto a root, y que se muestre una advertencia como resultado si cualquier otra opción --user se encuentra.

- --version, -V

  Muestra información de versión y sale.

En MySQL 5.0, puede asignar un valor a una variable de sistema del servidor usando una opción de la forma --*nombre_variable*=*valor*. Por ejemplo, --key_buffer_size=32M inicializa la variable key_buffer_size con un valor de 32MB.

Tenga en cuenta que al asignar un valor a una variable, MySQL puede corregir automáticamente la asignación para permanecer dentro de un rango concreto, o ajustar el valor al valor permitido más próximo si sólo se permiten ciertos valores.

También puede inicializar variables usando las sintaxis `--set-variable=nombre_variable=valor` o `-O nombre_variable=valor`. Sin embargo esta sintaxis está obsoleta

Puede encontrar una descripción concreta de todas las variables en Sección 5.3.3, “Variables de sistema del servidor”. La sección de ajustar los parámetros del servidor incluye información sobre cómo optimizarlos. Consulte Sección 7.5.2, “Afinar parámetros del servidor”.

Puede cambiar los valores de la mayoría de variables de sistema en un servidor en ejecución con el comando `SET`. Consulte Sección 13.5.3, “Sintaxis de `SET`”.

Si desea restringir el máximo valor que se puede asignar a una opción de arranque con `SET`, puede definirlo con la opción de línea de comandos `--maximum-var_name`.

5.3.2. El modo SQL del servidor

MySQL server puede operar en distintos modos SQL, y puede aplicar estos modos de forma distinta a diferentes clientes. Esto permite que cada aplicación ajuste el modo de operación del servidor a sus propios requerimientos.

Los modos definen qué sintaxis SQL debe soportar MySQL y que clase de chequeos de validación de datos debe realizar. Esto hace más fácil de usar MySQL en distintos entornos y usar MySQL junto con otros servidores de bases de datos.

Puede especificar el modo SQL por defecto arrancando `mysqld` con la opción `--sql-mode="modes"`. El valor puede dejarse en blanco (`--sql-mode=""`) si desea resetearlo.

En MySQL 5.0, también puede cambiar el modo SQL tras el tiempo de arranque cambiando la varialbe `sql_mode` usando el comando `SET [SESSION|GLOBAL] sql_mode='modes'` . Asignar la variable `GLOBAL` requiere el privilegio `SUPER` y afecta las operaciones de todos los clientes que conecten a partir de entonces. Asignar la variable `SESSION` afecta sólo al cliente actual. Cualquier cliente puede cambiar el valor de `sql_mode` en su sesión en cualquier momento.

`modes` es una lista de los diferentes modos separados por comas (`,`) . Puede consultar el modo actual mediante el comando `SELECT @@sql_mode` . El valor por defecto es vacío (sin modo seleccionado).

Los valores de los modos `sql_mode` más importantes probablemente son los siguientes:

- `ANSI`

  Cambia el comportamiento y la sintaxis para cumplir mejor el SQL.

- `STRICT_TRANS_TABLES`

  Si un valor no puede insertarse tal y como se da en una tabla transaccional, se aborta el comando. Para tablas no transaccionales, aborta el comando si el valor se encuentra en un comando que implique un sólo registro o el primer registro de un comando de varios registros. Más detalles a continuación en esta sección. (Implementado en MySQL 5.0.2)

- `TRADITIONAL`

  Hace que MySQL se comporte como un sistema de bases de datos SQL ``tradicional''. Una simple descripción de este modo es `` da un error en lugar de una alerta'' cuando se inserta un valor incorrecto en la columna. **Nota**: `INSERT`/`UPDATE` aborta así que se detecta un error. Puede que no sea lo que quiera si está usando un motor de almacenamiento no transaccional, ya que los cambios en los datos anteriores al error no se deshacen, resultando en una actualización ``parcial'' . (Añadido en MySQL 5.0.2)

Cuando este manual se refiere al ``modo estricto,'' implica un modo donde al menos `STRICT_TRANS_TABLES` o `STRICT_ALL_TABLES` está permitido.

La siguiente lista describe todos los modos soportados:

- `ALLOW_INVALID_DATES`

  No hace un chequeo total de los datos en modo estricto. Chequea sólo que los meses se encuentran en el rango de 1 a 12 y que los días están en el rango de 1 a 31. Esto es muy conveniente para aplicaciones Web donde obtiene un año, mes y día en tres campos distintos y quiere guardar exactamente lo que inserta el usuario (sin validación de datos). Este modo se aplica a columnas `DATE` y `DATETIME` . No se aplica a columnas `TIMESTAMP` , que siempre requieren una fecha válida.

  Este modo se implementó en MySQL 5.0.2. Antes de 5.0.2, este era el modo por defecto de MySQL para tratar datos. Desde 5.0.2, el permitir el modo estricto provoca que el servidor requiera que el mes y día se evalúen como valores legales y no simplemente en los rangos de 1 a 12 y de 1 a 31, respectivamente. Por ejemplo, `'2004-04-31'` es legal con el modo estricto desactivado, pero ilegal con el modo estricto activado. Para permitir tales fechas en modo estricto, habilite `ALLOW_INVALID_DATES` también.

- `ANSI_QUOTES`

  Trata '`"`' como un identificador delimitador de carácter (como '`` ` ``') y no como un delimitador de cadenas de caracteres. Puede usar '`` ` ``' para delimitar identificadores en modo ANSI. Con `ANSI_QUOTES` activado, puede usar doble delimitadores para delimitar una cadena de caracteres literales, ya que se interpreta como un identificador.

- `ERROR_FOR_DIVISION_BY_ZERO`

  Produce un error en modo estricto (de otra forma una advertencia) cuando encuentra una división por cero (o `MOD(X,0)`) durante un `INSERT` o `UPDATE`, o en cualquier expresión (por ejemplo, en una lista de select o cláusula `WHERE`) que implica datos de tablas y una divisón por cero. Si este modo no se da, MySQL retorna `NULL` para una división por cero. Si se usa `INSERT IGNORE` o `UPDATE IGNORE`, MySQL genera una advertencia de división por cero, pero el resultado de la operación es `NULL`. (Implmentado en MySQL 5.0.2)

- `HIGH_NOT_PRECEDENCE`

  Desde MySQL 5.0.2 , la precedencia del operador `NOT` se trata tal que expresiones como `NOT a BETWEEN b AND c` se parsean como `NOT (a BETWEEN b AND c)`. Antes de MySQL 5.0.2, la expresión se parseaba como `(NOT a) BETWEEN b AND c`. El antiguo comportamiento de mayor-precedencia puede obtenerse permitiendo el modo SQL `HIGH_NOT_PRECEDENCE` . (Añadido en MySQL 5.0.2)

  ```
  mysql> SET sql_mode = '';
  mysql> SELECT NOT 1 BETWEEN -5 AND 5;
          -> 0
  mysql> SET sql_mode = 'broken_not';
  mysql> SELECT NOT 1 BETWEEN -5 AND 5;
          -> 1
  ```

- `IGNORE_SPACE`

  Permite nombres entre el nombre de función y el carácter '`(`' . Esto fuerza que todos los nombres de función se traten como palabras reservadas. Como resultado, si quiere acceder a cualquier base de datos, tabla, o nombre de columna que sea una palabra reservada, debe delimitarla. Por ejemplo, y

como hay una función `USER()` , el nombre de la tabla `user` en la base de datos `mysql` y la columna `User` en esa table se reseva, así que debe delimitarla:

```
SELECT "User" FROM mysql."user";
```

- `NO_AUTO_CREATE_USER`

  Previene que `GRANT` cree automáticamente nuevos usuarios si de otra forma se haría, a no ser que se especifique un usuario. (Añadido en MySQL 5.0.2)

- `NO_AUTO_VALUE_ON_ZERO`

  `NO_AUTO_VALUE_ON_ZERO` afecta el tratamiento de las columnas `AUTO_INCREMENT` . Normalmente, genera el siguiente número de secuencia para la columna insertando `NULL` o `0` en ella. `NO_AUTO_VALUE_ON_ZERO` suprime este comportamiento para `0` de forma que sólo `NULL` genera el siguiente número de secuencia.

  Este modo puede ser útil si `0` se ha almacenado en una tabla con columnas `AUTO_INCREMENT` . (Esta no es una práctica recomendada, de todos modos.) Por ejemplo, si vuelca la tabla con `mysqldump` y posteriormente la recarga, normalmente MySQL genera un nuevo número de secuencia cuando encuentra los valores `0` , resultando en una tabla con distinto contenido que la que fue volcada. Activar `NO_AUTO_VALUE_ON_ZERO` antes de recargar el fichero con el volcado resuelve el problema. En MySQL 5.0, `mysqldump` incluye automáticamente en su salida un comando permitiendo `NO_AUTO_VALUE_ON_ZERO`.

- `NO_BACKSLASH_ESCAPES`

  Desactiva el uso del carácter de barra invertida ('`\`') como carácter de escape en cadenas de caracteres. Con este modo activado, la barra invertida se convierte en un carácter ordinario como cualquier otro. (Implementado en MySQL 5.0.1)

- `NO_DIR_IN_CREATE`

  Cuando crea una tabla, ignora todas las directivas `INDEX DIRECTORY` y `DATA DIRECTORY`. Este opción es útil en servidores de replicación esclavos.

- `NO_ENGINE_SUBSTITUTION`

  Evita la substitución automática de motor de almacenamiento cuando el motor deseado no está disponible o compilado.

- `NO_FIELD_OPTIONS`

  No muestra opciones específicas para columnas de MySQL en la salida de `SHOW CREATE TABLE`. Este modo se usa con `mysqldump` en modo de portabilidad.

- `NO_KEY_OPTIONS`

  No muestra opciones específicas para índices de MySQL en la salida de `SHOW CREATE TABLE`. Este modo se usa con `mysqldump` en modo de portabilidad.

- `NO_TABLE_OPTIONS`

  No muestra opciones específicas para tablas (tales como `ENGINE`) en la salida de `SHOW CREATE TABLE`. Este modo se usa con `mysqldump` en modo de portabilidad.

- `NO_UNSIGNED_SUBTRACTION`

En operaciones de resta, no marca el resultado como `UNSIGNED` si uno de los operandos no tiene signo. Note que esto hace que `UNSIGNED BIGINT` no sea 100% usable en todos los contextos. Consulte Sección 12.8, “Funciones y operadores de cast”.

- `NO_ZERO_DATE`

  En modo estricto, no permite `'0000-00-00'` como fecha válida. Puede insertar fechas 0 con la opción `IGNORE` . Cuando no está en modo estricto, la fecha se acepta pero se genera una advertencia. (Añadido en MySQL 5.0.2)

- `NO_ZERO_IN_DATE`

  En modo estricto, no acepta fechas la parte del mes o día es 0. Se usa con la opción `IGNORE` , inserta una fecha `'0000-00-00'` para cualquiera de estas fechas. Cuando no está en modo estricto, la fecha se acepta pero se genera una advertencia. (Añadido en MySQL 5.0.2)

- `ONLY_FULL_GROUP_BY`

  No permite consultas que en la parte del `GROUP BY` se refieran a una columna que no se seleccione.

- `PIPES_AS_CONCAT`

  Trata `||` como un concatenador de columnas de caracteres (lo mismo que `CONCAT()`) en lugar de como sinónimo de `OR`.

- `REAL_AS_FLOAT`

  Trata `REAL` como un sinónimo de `FLOAT` en lugar de sinónimo de `DOUBLE`.

- `STRICT_ALL_TABLES`

  Activa el modo estricto para todos los motores de almacenamiento. Rechaza los datos inválidos. Detalles adicionales a continuación. (Añadido en MySQL 5.0.2)

- `STRICT_TRANS_TABLES`

  Habilita el modo estricto para motores de almacenamiento transaccionales, y cuando sea posible también para los no transaccionales. Detalles adicionales a continuación. (Implementado en MySQL 5.0.2)

El modo estricto controla cómo MySQL trata los valores de entrada invalidos o no presentes. Un valor puede ser inválido por distintas razones. Por ejemplo, puede tener un tipo de datos incorrecto para la columna, o puede estar fuera de rango. Un valor no está presente cuando el registro a insertarse no tiene un valor para una columna que no tiene la cláusua `DEFAULT` explícita en su definición.

Para tablas transaccionales, se genera un error para valores inválidos o no presentes en un comando con los modos `STRICT_ALL_TABLES` o `STRICT_TRANS_TABLES` habilitados. El comando se aborta y deshace.

Para tablas no transaccionales, el comportamiento es el mismo para cualquier modo, si un valor incorrecto se encuentra en el primer registro a insertar o actualizar. El comando se aborta y la tabla continúa igual. Si el comando inserta o modifica varios registros y el valor incorrecto aparece en el segundo o posterior registro, el resultado depende de qué modo estricto esté habilitado:

- Para `STRICT_ALL_TABLES`, MySQL devuelve un error e ignora el resto de los registros. Sin embargo, en este caso, los primeros registros se insertan o actualizan. Esto significa que puede producirse una actualización parcial, que puede no ser lo que desea. Para evitarlo, es mejor usar comandos de un único registro ya que pueden abortarse sin cambiar la tabla.

- Para STRICT_TRANS_TABLES, MySQL convierte los valores inválidos en el valor válido más próximo para la columna e inserta el nuevo valor. Si un valor no está presente, MySQL inserta el valor por defecto implícito para el tipo de la columna. En ese caso, MySQL genera una advertencia en lugar de un error y continúa procesando el comando. Los valores implícitos se describen en Sección 13.1.5, “Sintaxis de CREATE TABLE”.

El modo estricto no permite fechas inválides como '2004-04-31'. Esto sigue permitiendo fechas con partes con ceros, como 2004-04-00' o fechas ``cero''. Para no permitirlas tampoco, active los modos SQL NO_ZERO_IN_DATE y NO_ZERO_DATE además del modo estricto.

Si no usa el modo estricto (esto es, ni STRICT_TRANS_TABLES ni STRICT_ALL_TABLES están activados), MySQL inserta valores ajustados para valores inválidos o no presentes y produce advertencias. En modo estricto, puede producir este comportamiento usando INSERT IGNORE o UPDATE IGNORE. Consulte Sección 13.5.4.22, “Sintaxis de SHOW WARNINGS”.

Los siguientes modos especiales se proporcionan como abreviaciones de combinaciones de modos de la lista precedente. Todos están disponibles en MySQL 5.0 empezando en la versión 5.0.0, excepto para TRADITIONAL, que se implementó en MySQL 5.0.2.

La descripción incluye todos los modos que están disponibles en la reción más reciente de MySQL. Para versiones anteriores, un modo de combinación no incluye todos los modos individuales que sólo están disponibles en las versiones más recientes.

- ANSI

  Equivalente a REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE. Antes de MySQL 5.0.3, ANSI también incluye ONLY_FULL_GROUP_BY. Consulte Sección 1.7.3, “Ejecutar MySQL en modo ANSI”.

- DB2

  Equivalente a PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MAXDB

  Equivalente a PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- MSSQL

  Equivalente a PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MYSQL323

  Equivalente a NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- MYSQL40

  Equivalente a NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- ORACLE

  Equivalente a PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- POSTGRESQL

Equivalente a PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- TRADITIONAL

  Equivalente a STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER.

5.3.3. Variables de sistema del servidor

El servidor mantiene varias variables de sistema que indican cómo está configurado. Todas ellas tienen valores por defecto. Pueden cambiarse al arrancar el servidor usando opciones en la línea de comandos o en ficheros de opciones. La mayoría de ellos pueden cambiarse en tiempo de ejecución usando el comando SET.

El servidor mysqld mantiene dos clases de variables. Las variables globales afectan las operaciones globales del servidor. Las variables de sesión afectan las operaciones para conexiones individuales de clientes.

Cuando el servidor arranca, inicializa todas las variables globales a sus valores por defecto. Estos valores pueden cambiarse con las opciones especificadas en los ficheros de opciones o en la línea de comandos. Una vez que el servidor arranca, aquellas variables globales que sean dinámicas pueden cambiarse conectando al servidor y ejectuando el comando SET GLOBAL *var_name*. Para cambiar una variable global, debe tener el privilegio SUPER.

El servidor mantiene un conjunto de variables de sesión para cada cliente que se conecta. Las variables de sesión del cliente se inicializan en tiempo de conexión usando los valores actuales de las correspondientes variables globales. Para aquellas variables de sesión que son dinámicas, el cliente puede cambiarlas mediante un comando SET SESSION *var_name* . Cambiar una variable de sesión no necesita privilegios especiales, pero un cliente puede cambiar sólo sus variables de sesión, no las de ningún otro cliente.

Un cambio de una variable global es visible para cualquier cliente que acceda a esa variable global. Sin embargo, esto afecta a las correspondientes variables de sesión que se inicializan por la variable global sólo para clientes que se conecten después del cambio. Esto no afecta las variables de sesión para cualquier cliente que ya esté conectado (tampoco para los clientes que ejecuten el comando SET GLOBAL).

Cuando se cambia una variable usando las opciones de arranque, los valores de la variable pueden darse con un sufijo K, M, o G para indicar kilobytes, megabytes, o gigabytes, respectivamente. Por ejemplo, el siguiente comando arranca el servidor con un tamaño de key buffer de 16 megabytes:

```
mysqld --key_buffer_size=16M
```

No importa que los sufijos se escriban en mayúscula o minúscula; 16M y 16m son equivalentes.

En tiempo de ejecución, use el comando SET para cambiar las variables de sistema. En este contexto, los sufijos no pueden usarse, pero el valor puede tomar la forma de una expresión:

```
mysql> SET sort_buffer_size = 10 * 1024 * 1024;
```

Para espeficicar explícitamente si desea cambiar la variable global o de sesión use la opción GLOBAL o SESSION:

```
mysql> SET GLOBAL sort_buffer_size = 10 * 1024 * 1024;
```

```
mysql> SET SESSION sort_buffer_size = 10 * 1024 * 1024;
```

Sin dicha opción, el comando actualiza la variable de sesión.

Las variables que pueden cambiarse en tiempo de ejecución se listan en Sección 5.3.3.1, “Variables de sistema dinámicas”.

Si desea restringir el valor máximo que puede tomar una variable de sistema con el comando `SET` , puede especificarlo con `--maximum-`*`var_name`* en el arranque del servidor. Por ejemplo, para evitar que el valor de `query_cache_size` se incremente por encima de 32MB en tiempo de ejecución, use la opción `--maximum-query_cache_size=32M`.

Puede consultar las variables de sistema y sus valores usando el comando `SHOW VARIABLES` . Consulte Sección 9.4, “Variables de sistema” para más información.

```
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                                                     |
| 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                                          |
| completion_type                 | 0                                                          |
| concurrent_insert               | 1                                                          |
| connect_timeout                 | 5                                                          |
| datadir                         | /var/lib/mysql/                                            |
| date_format                     | %Y-%m-%d                                                   |
| datetime_format                 | %Y-%m-%d %H:%i:%s                                          |
| div_precision_increment         | 4                                                          |
| default_week_format             | 0                                                          |
| delay_key_write                 | ON                                                         |
| delayed_insert_limit            | 100                                                        |
| delayed_insert_timeout          | 300                                                        |
| delayed_queue_size              | 1000                                                       |
| expire_logs_days                | 0                                                          |
| flush                           | OFF                                                        |
| flush_time                      | 0                                                          |
| ft_boolean_syntax               | + -><()~*:""&|                                             |
| ft_max_word_len                 | 84                                                         |
| ft_min_word_len                 | 4                                                          |
| ft_query_expansion_limit        | 20                                                         |
| ft_stopword_file                | (built-in)                                                 |
| group_concat_max_len            | 1024                                                       |
| have_archive                    | NO                                                         |
```

```
| have_bdb                        | YES                                                                |
| have_blackhole_engine           | YES                                                                |
| have_compress                   | YES                                                                |
| have_crypt                      | YES                                                                |
| have_csv                        | YES                                                                |
| have_example_engine             | YES                                                                |
| have_federated_engine           | YES                                                                |
| have_geometry                   | YES                                                                |
| have_innodb                     | YES                                                                |
| have_isam                       | NO                                                                 |
| have_ndbcluster                 | DISABLED                                                           |
| have_openssl                    | NO                                                                 |
| have_query_cache                | YES                                                                |
| have_raid                       | NO                                                                 |
| have_rtree_keys                 | YES                                                                |
| have_symlink                    | YES                                                                |
| init_connect                    |                                                                    |
| init_file                       |                                                                    |
| init_slave                      |                                                                    |
| 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_concurrency_tickets      | 500                                                                |
| innodb_data_file_path           | ibdata1:10M:autoextend                                             |
| innodb_data_home_dir            |                                                                    |
| innodb_doublewrite              | ON                                                                 |
| innodb_fast_shutdown            | 1                                                                  |
| innodb_file_io_threads          | 4                                                                  |
| innodb_file_per_table           | OFF                                                                |
| innodb_flush_log_at_trx_commit  | 1                                                                  |
| innodb_flush_method             |                                                                    |
| innodb_force_recovery           | 0                                                                  |
| innodb_lock_wait_timeout        | 50                                                                 |
| innodb_locks_unsafe_for_binlog  | OFF                                                                |
| innodb_log_arch_dir             |                                                                    |
| innodb_log_archive              | OFF                                                                |
| innodb_log_buffer_size          | 1048576                                                            |
| innodb_log_file_size            | 5242880                                                            |
| innodb_log_files_in_group       | 2                                                                  |
| innodb_log_group_home_dir       | ./                                                                 |
| innodb_max_dirty_pages_pct      | 90                                                                 |
| innodb_max_purge_lag            | 0                                                                  |
| innodb_mirrored_log_groups      | 1                                                                  |
| innodb_open_files               | 300                                                                |
| innodb_sync_spin_loops          | 20                                                                 |
| innodb_table_locks              | ON                                                                 |
| innodb_support_xa               | ON                                                                 |
| innodb_thread_concurrency       | 8                                                                  |
| innodb_thread_sleep_delay       | 10000                                                              |
| interactive_timeout             | 28800                                                              |
| join_buffer_size                | 131072                                                             |
| key_buffer_size                 | 8388600                                                            |
| key_cache_age_threshold         | 300                                                                |
| key_cache_block_size            | 1024                                                               |
| key_cache_division_limit        | 100                                                                |
| language                        | /usr/share/mysql/english/                                          |
| large_files_support             | ON                                                                 |
| large_pages                     | OFF                                                                |
| large_page_size                 | 0                                                                  |
| license                         | GPL                                                                |
| local_infile                    | ON                                                                 |
| locked_in_memory                | OFF                                                                |
| log                             | OFF                                                                |
| log_bin                         | OFF                                                                |
| log_bin_trust_routine_creators  | OFF                                                                |
```

```
| log_error                       |                                                           |
| log_slave_updates               | OFF                                                       |
| log_slow_queries                | OFF                                                       |
| log_warnings                    | 1                                                         |
| long_query_time                 | 10                                                        |
| low_priority_updates            | OFF                                                       |
| lower_case_file_system          | OFF                                                       |
| lower_case_table_names          | 0                                                         |
| max_allowed_packet              | 1048576                                                   |
| max_binlog_cache_size           | 4294967295                                                |
| max_binlog_size                 | 1073741824                                                |
| max_connect_errors              | 10                                                        |
| max_connections                 | 100                                                       |
| max_delayed_threads             | 20                                                        |
| max_error_count                 | 64                                                        |
| max_heap_table_size             | 16777216                                                  |
| max_insert_delayed_threads      | 20                                                        |
| max_join_size                   | 4294967295                                                |
| max_length_for_sort_data        | 1024                                                      |
| max_relay_log_size              | 0                                                         |
| max_seeks_for_key               | 4294967295                                                |
| max_sort_length                 | 1024                                                      |
| max_tmp_tables                  | 32                                                        |
| max_user_connections            | 0                                                         |
| max_write_lock_count            | 4294967295                                                |
| multi_range_count               | 256                                                       |
| myisam_data_pointer_size        | 6                                                         |
| myisam_max_sort_file_size       | 2147483647                                                |
| myisam_recover_options          | OFF                                                       |
| myisam_repair_threads           | 1                                                         |
| myisam_sort_buffer_size         | 8388608                                                   |
| engine_condition_pushdown       | OFF                                                       |
| ndb_autoincrement_prefetch_sz   | 32                                                        |
| ndb_force_send                  | ON                                                        |
| ndb_use_exact_count             | ON                                                        |
| ndb_use_transactions            | ON                                                        |
| ndb_cache_check_time            | 0                                                         |
| net_buffer_length               | 16384                                                     |
| net_read_timeout                | 30                                                        |
| net_retry_count                 | 10                                                        |
| net_write_timeout               | 60                                                        |
| new                             | OFF                                                       |
| old_passwords                   | OFF                                                       |
| open_files_limit                | 1024                                                      |
| optimizer_prune_level           | 1                                                         |
| optimizer_search_depth          | 62                                                        |
| pid_file                        | /var/lib/mysql/gigan.pid                                  |
| port                            | 3306                                                      |
| preload_buffer_size             | 32768                                                     |
| protocol_version                | 10                                                        |
| query_alloc_block_size          | 8192                                                      |
| query_cache_limit               | 1048576                                                   |
| query_cache_min_res_unit        | 4096                                                      |
| query_cache_size                | 0                                                         |
| query_cache_type                | ON                                                        |
| query_cache_wlock_invalidate    | OFF                                                       |
| query_prealloc_size             | 8192                                                      |
| range_alloc_block_size          | 2048                                                      |
| read_buffer_size                | 131072                                                    |
| read_only                       | OFF                                                       |
| read_rnd_buffer_size            | 262144                                                    |
| relay_log_purge                 | ON                                                        |
| relay_log_space_limit           | 0                                                         |
| rpl_recovery_rank               | 0                                                         |
| secure_auth                     | OFF                                                       |
| server_id                       | 0                                                         |
| skip_external_locking           | ON                                                        |
```

```
| skip_networking                 | OFF                                                        |
| skip_show_database              | OFF                                                        |
| slave_compressed_protocol       | OFF                                                        |
| slave_load_tmpdir               | /tmp/                                                      |
| slave_net_timeout               | 3600                                                       |
| slave_skip_errors               | OFF                                                        |
| slave_transaction_retries       | 10                                                         |
| slow_launch_time                | 2                                                          |
| socket                          | /var/lib/mysql/mysql.sock                                  |
| sort_buffer_size                | 2097144                                                    |
| sql_mode                        |                                                            |
| storage_engine                  | MyISAM                                                     |
| sql_notes                       | OFF                                                        |
| sql_warnings                    | OFF                                                        |
| sync_binlog                     | 0                                                          |
| sync_replication                | 0                                                          |
| sync_replication_slave_id       | 0                                                          |
| sync_replication_timeout        | 10                                                         |
| sync_frm                        | ON                                                         |
| system_time_zone                | EST                                                        |
| table_cache                     | 64                                                         |
| table_type                      | MyISAM                                                     |
| thread_cache_size               | 0                                                          |
| thread_stack                    | 196608                                                     |
| time_format                     | %H:%i:%s                                                   |
| 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                         | 5.0.7-beta-Max                                             |
| version_bdb                     | Sleepycat Software: Berkeley DB 4.1.24: (June 11, 2005)    |
| version_comment                 | MySQL Community Edition - Max (GPL)                        |
| version_compile_machine         | i686                                                       |
| version_compile_os              | pc-linux-gnu                                               |
| wait_timeout                    | 28800                                                      |
+---------------------------------+------------------------------------------------------------+
219 rows in set (0.00 sec)
```

La mayoría de variables de sistema se describen aquí. Las variables sin versión indican que están presentes en todas las versiones de MySQL 5.0. Para información histórica acerca de sus implementaciones, consulte *Manual de referencia de MySQL 4.1*. Las variables de sistema `InnoDB` están listadas en Sección 15.4, “Opciones de arranque de `InnoDB`”.

Los valores para tamaños de buffer, longitudes y tamaño de pila se dan en bytes a no ser que se especifique otra unidad.

Información sobre el ajuste de estas variables se encuentra en Sección 7.5.2, “Afinar parámetros del servidor”.

- `auto_increment_increment`

  `auto_increment_increment` y `auto_increment_offset` están pensados para usarse con replicación maestro-a-maestro, y puede usarse para controlar la operación de columnas `AUTO_INCREMENT` . Ambas variables pueden cambiarse global o localmente, y cada una de ellas puede tomar un valor entre 1 y 65.535 incluídos. Cambiar el valor de estas variables a 0 causa que su valor sea 1. Intentar cambiar el valor de cualquiera de estas variables a un entero mayor que 65,535 o menor a 0 causa que su valor sea 65,535. Intentar cambiar el valor de `auto_increment_increment` o de

`auto_increment_offset` con un valor no entero da un error, y el valor de la variable no se cambie en ese caso.

Estas dos variables afectan al comportamiento de la columnas `AUTO_INCREMENT` así:

- `auto_increment_increment` controla el intervalo en que se incrementa el valor de columna. Por ejemplo:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)

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)
```

(Note cómo se usa `SHOW VARIABLES` para obtener el valor actual de estas variables.)

- `auto_increment_offset` determina el punto de inicio para el valor de las columnas `AUTO_INCREMENT`. Considere lo siguiente, asumiendo que estos comandos se ejecutan durante la misma sesión que el ejemplo anterior:

```
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 |
+-----+
4 rows in set (0.02 sec)
```

Si el valor de `auto_increment_offset` es mayor que el de `auto_increment_increment`, entonces el valor `auto_increment_offset` se ignora.

Si una o ambas variables se cambian y los nuevos registros insertados en una tabla que contengan una columna `AUTO_INCREMENT` , el resultado puede ser no intuitivo, como los valores de las series de `AUTO_INCREMENT` se calculan sin tener en cuenta los valores ya existentes en la columna, y el siguiente valor insertado es el menor valor de la serie que es mayor al máximo valor existente en la columna `AUTO_INCREMENT` . En otras palabras, la serie se calcula así:

`auto_increment_offset + N * auto_increment_increment`

donde *N* es un entero positivo en la serie [1, 2, 3, ...]. Por ejemplo:

```
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)
```

Los valores mostrados por `auto_increment_increment` y `auto_increment_offset` generan las series 5 + *`N`* * 10, esto es, [5, 15, 25, 35, 45, ...]. El mayor valor presente en la columna `col` antes del `INSERT` es 31, y el siguiente valor disponible en la serie `AUTO_INCREMENT` es 35, así que los valores insertados para `col` empiezan en ese punto y los resultados se muestran para la consulta `SELECT`.

Es importante recordar que no es posible confinar los efectos de estas dos variables en una sola tabla, y por lo tanto no toman el lugar de la secuencia ofrecido por otros sistemas gestores de bases de datos; estas variables controlan el comportamiento de todas las columnas `AUTO_INCREMENT` en **todas** las tablas del servidor MySQL. Si una de estas variables se cambia globalmente, entonces sus efectos persisten hasta que el valor global se cambia o subyace al cambiarlo localmente, o hasta que se reinicia `mysqld` ; si se cambia localmente, entonces el nuevo valor afecta a las columnas `AUTO_INCREMENT` para todas las tablas en que se inserten nuevos registros por el usuario actual durante la sesión, hasta que los valores se cambien durante esa sesión.

La variable `auto_increment_increment` se añadió en MySQL 5.0.2. Su valor por defecto es 1. Consulte Capítulo 6, *Replicación en MySQL*.

- `auto_increment_offset`

  Esta variable se añadió en MySQL 5.0.2. Su valor por defecto es 1. Para más detalles, consulte la descripción de `auto_increment_increment`.

- `back_log`

  El numero de peticiones de conexión que puede tener MySQL. Esto se usa cuando el thread principal de MySQL recibe muchas peticiones de conexión en un pequeño lapso de tiempo. Necesita algo de tiempo (aunque muy poco) el thread principal para comprobar la conexión y empezar un nuevo thread. El valor de `back_log` indica cuántas peticiones pueden almacenarse duarnte este corto espacio de tiempo antes que MySQL temporalmente pare de responder a nuevas peticiones. Necesita incrementarlo sólo si espera un gran número de conexiones en un pequeño lapso de tiempo.

  En otras palabras, este valor es el tamaño de la cola que escucha conexiones TCP/IP entrantes. El sistema operativo tiene su propio límite en el tamaño de la cola. La página del manual para la llamada de sistema Unix `listen()` proporcion más detalles. Consulte la documentación del SO para el máximo valor de esta variable. Intentar cambiar el valor de `back_log` por encima del límite del sistema operativo no es efectivo.

- `basedir`

  El directorio de instalación de MySQL. Esta variable puede cambiarse con la opción `--basedir` .

- `bdb_cache_size`

  El tamaño del buffer que se utiliza para cachear índices y registros para tablas `BDB` . Si no usa tablas `BDB` debe arrancar `mysqld` con `--skip-bdb` para no gastar memoria para esta caché.

- `bdb_home`

El directorio base para las tablas BDB tables. Debe asignarse el mismo valor que para la variable datadir.

- bdb_log_buffer_size

  El tamaño del buffer que se utiliza para cachear índices y registros para tablas BDB . Si no usa tablas BDB , debe usar el valor 0 o arrancar mysqld con --skip-bdb para no gastar memoria con esta caché.

- bdb_logdir

  El directorio en que el motor de almacenamiento BDB escribe sus ficheros de log. Esta variable puede cambiarse con la opción --bdb-logdir .

- bdb_max_lock

  El máximo número de bloqueos que pueden tenerse activos en una tabla BDB (10,000 por defecto). Debe incrementarlo si errores como los siguientes ocurren cuando realiza transacciones grandes o cuando mysqld tiene que examinar muchos registros para calcular una consulta:

  ```
  bdb: Lock table is out of available locks
  Got error 12 from ...
  ```

- bdb_shared_data

  Está ON si usa --bdb-shared-data.

- bdb_tmpdir

  El valor de la opción --bdb-tmpdir .

- bdb_version

  Consulte la versión de version_bdb.

- binlog_cache_size

  El tamaño de la caché para tratar comandos SQL para el log binario durante una transacción. La cache del log binario se guarda para cada cliente si el servidor soporta diversos motores de almacenamiento transaccionales, empezando por MySQL 4.1.2, si el servidor tiene activados los logs binarios (opción --log-bin). Si suele usar transacciones grandes, con múltiples comandos, puede incrementar este valor para mejorar el rendimiento. Las variables de estado Binlog_cache_use y Binlog_cache_disk_use pueden ser útiles para ajustar el tamaño de estas variables. Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

- bulk_insert_buffer_size

  MyISAM usa una caché especial con forma de árbol para hacer las inserciones más rápidas para INSERT ... SELECT, INSERT ... VALUES (...), (...), ..., y LOAD DATA INFILE. Esta variable limita el tamaño del árbol de la caché en bytes por thread. Cambiarlo a 0 desactiva esta optimización. **Nota**: Esta caché se usa sólo al añadir datos en una tabla no vacía. El valor por defecto es 8MB.

- character_set_client

  El conjunto de caracteres para comandos que llegan del cliente.

- `character_set_connection`

  El conjunto de caracteres usado por los literales que no tienen un conjunto de caracteres introducido y para conversiones de números a cadenas de caracteres.

- `character_set_database`

  El conjunto de caracteres usado por la base de datos por defecto. El servidor actualiza esta variable cada vez que la base de datos por defecto cambia. Si no hay base de datos por defecto, la variable toma el mismo valor que `character_set_server`.

- `character_set_results`

  El conjunto de caracteres usado para retornar los resultados de las consultas al cliente.

- `character_set_server`

  El conjunto de caracteres por defecto del servidor.

- `character_set_system`

  El conjunto de caracteres usado por el servidor para almacenar identificadores. El valor es siempre `utf8`.

- `character_sets_dir`

  El directorio en el que están instalados los conjuntos de caracteres.

- `collation_connection`

  La colación del conjunto de caracteres de las conexiones.

- `collation_database`

  La colación usada por la base de datos por defecto. El servidor cambia esta variable cada vez que la base de datos por defecto cambia. Si no hay una base de datos por defecto, la variable toma el valor de `collation_server`.

- `collation_server`

  La colación por defecto del servidor.

- `concurrent_insert`

  Si `ON` (por defecto), MySQL permite comandos `INSERT` y `SELECT` que se ejecuten concurrentemente para tablas `MyISAM` que no tienen bloques libres en el medio. Puede desactivar esta opción arrancando `mysqld` con `--safe` or `--skip-new`.

  En MySQL 5.0.6 esta variable ha cambiado a un entero que toma 3 valores:

Valor	Descripción
0	Off
1	(Defecto) Permite inserciones concurrentes para tablas MyISAM que no tienen agujeros
2	Permite inserciones concurrentes para todas las tablas MyISAM. Si la tabla tiene un agujeroy está en uso por otro thread el nuevo registro se insertará al final de la tabla. Si la tabla no está en uso, MySQL hará un bloqueo de lectura normal e insertará el registro en el agujero.

- `connect_timeout`

  El número de segundos que el servidor `mysqld` espera para un paquete de conexión antes de responder con `Bad handshake`.

- `datadir`

  El directorio de datos de MySQL. Esta variable puede cambiarse con la opción `--datadir`.

- `default_week_format`

  El modo a usar por la función `WEEK()`.

- `delay_key_write`

  Esta opción se aplica sólo a tablas `MyISAM` . Puede tener uno de los siguientes valores que afectan la forma de tratar la opción `DELAY_KEY_WRITE` que puede usarse en los comandos `CREATE TABLE` .

Opción	Descrición
`OFF`	`DELAY_KEY_WRITE` se ignora.
`ON`	MySQL activa la opción `DELAY_KEY_WRITE` para `CREATE TABLE`. Este es el valor por defecto.
`ALL`	Todas las nuevas tablas abiertas se tratan como si se hubieran creado con la opción `DELAY_KEY_WRITE` activada.

  Si `DELAY_KEY_WRITE` está activo, significa que el key buffer para tablas con esta opción no se vuelcan en cada actualización del índice, sino que sólo cuando se cierra una tabla. Esto acelera las escrituras de claves, pero si usa esta característica, debe añadir chequeo automático de todas las tablas `MyISAM` arrancando el servidor con la opción `--myisam-recover` (por ejemplo `--myisam-recover=BACKUP,FORCE`). Consulte Sección 5.3.1, “Opciones del comando `mysqld`” y Sección 14.1.1, “Opciones de arranque de `MyISAM`”.

  Tenga en cuenta que `--external-locking` no ofrece ninguna protección contra corrupción de índices para tablas que usan escrituras de claves retardadas.

- `delayed_insert_limit`

  Tras insertar `delayed_insert_limit` registros retardados, el thread que se encarga del `INSERT DELAYED` comprueba si hay algún comando `SELECT` pendiente. Si es así, les permite que se ejecuten antes de continuar insertando registros retardados.

- `delayed_insert_timeout`

  Cuánto debe esperar el thread encargado de `INSERT DELAYED` para un comando `INSERT` antes de terminar.

- `delayed_queue_size`

  Este es el límite por tabla del número de registros a encolar cuando se tratan comandos `INSERT DELAYED` . Si la cola se llena, cualquier cliente que lance un comando `INSERT DELAYED` espera hasta que haya espacio en la cola de nuevo.

- `expire_logs_days`

El número de días para eliminar el log binario automáticamente. El valor por defecto es 0, lo que significa ``sin eliminación automática''. Las posibles eliminaciones se realizan al arranque y en la rotación del log binario.

- `flush`

  Está `ON` si ha arrancado `mysqld` con la opción `--flush`.

- `flush_time`

  Si se asigna un valor distinto a cero, todas las tablas se cierran cada `flush_time` segundos para liberar recursos y sincronizar datos no volcados en disco. Recomendamos que esta opción se use sólo en Windows 9x o Me, o en sistemas con recursos mínimos.

- `ft_boolean_syntax`

  La lista de operadores soportado por búsquedas full-text booleanas usando `IN BOOLEAN MODE`. Consulte Sección 12.7.1, “Búsquedas booleanas de texto completo (Full-Text)”.

  El valor por defecto de la variable es `'+ -><()~*:""&|'`. Las reglas para cambiar el valor son las siguientes:

  - La función del operador viene determinada por su posición en la cadena de caracteres.
  - El valor de reemplazo debe tener 14 caracteres.
  - Cada carácter debe ser ASCII y no alfanumérico.
  - El primer o segundo carácter debe ser un espacio.
  - No se permiten duplicados excepto los operadores delimitadores de frase en las posiciones 11 y 12. Estos dos caracteres no tienen porqué ser los mismos, pero son los dos únicos posibles.
  - Las posiciones 10, 13, y 14 (que por defecto son ':', '&', y '|') están reservados para extensiones futuras.

- `ft_max_word_len`

  La longitud máxima de la palabra incluída en un índice `FULLTEXT` .

  Nota: índices `FULLTEXT` deben reconstruirse al cambiar esta variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len`

  La longitud mínima de la palabra a incluirse en un índice `FULLTEXT` .

  Nota: índices `FULLTEXT` deben reconstruirse tras cambiar estas variables. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit`

  El número de mejores coincidencias a usar en búsquedas full-text realizadas usando `WITH QUERY EXPANSION`.

- `ft_stopword_file`

El fichero del que lee la lista de palabras de detención en búsquedas full-text. Todas las palabras del fichero se usan; los comentarios *no* se tienen en cuenta. Por defecto, se usa una lista de palabras de detención (como se define en el fichero `myisam/ft_static.c`). Actualizar esta variable con una cadena vacía (`''`) desactiva el filtrado de palabras de detención.

Nota: índices `FULLTEXT` deben reconstruirse tras cambiar esta variable o los contenidos del fichero de palabras de detención. Use `REPAIR TABLE tbl_name QUICK`.

- `group_concat_max_len`

  El valor máximo permitido para la longitud del resultado de la función `GROUP_CONCAT()`.

- `have_archive`

  `YES` si `mysqld` soporta tablas `ARCHIVE`, `NO` si no.

- `have_bdb`

  `YES` si `mysqld` soporta tablas `BDB`. `DISABLED` si se usa `--skip-bdb`.

- `have_compress`

  Si está disponible la biblioteca de compresión `zlib` en el servidor. Si no lo está, las funciones `COMPRESS()` y `UNCOMPRESS()` no pueden usarse.

- `have_crypt`

  Si la llamada de sistema `crypt()` está disponible en el servidor. Si no, la función `CRYPT()` no puede usarse.

- `have_csv`

  `YES` si `mysqld` soporta tablas `ARCHIVE`, `NO` si no.

- `have_example_engine`

  `YES` si `mysqld` soporta tablas `EXAMPLE` , `NO` si no.

  `have_federated_engine`

  `YES` si `mysqld` soporta tablas `FEDERATED`, `NO` si no. Esta variable se añadió en MySQL 5.0.3.

- `have_geometry`

  Si el servidor soporta tipos de datos espaciales.

- `have_innodb`

  `YES` si `mysqld` soporta tablas `InnoDB` . `DISABLED` si se usa `--skip-innodb`.

- `have_isam`

  En MySQL 5.0, esto aparece sólo por razones de compatibilidad con versiones anteriores, y siempre es `NO`, ya que no hay soporte para tablas `ISAM`.

- `have_ndbcluster`

  `YES` si `mysqld` soporta tablas `NDB Cluster` . `DISABLED` si se usa `--skip-ndbcluster` .

- have_openssl

  YES si mysqld soporta SSL (cifrado) del protocolo cliente/servidor.

- have_query_cache

  YES si mysqld soporta la cache de consultas.

- have_raid

  YES si mysqld soporta la opción RAID .

- have_rtree_keys

  Si los índices RTREE están disponibles. (Se usan para indexación espacial en tablas MyISAM.)

- have_symlink

  Si el soporte para enlaces simbólicos está activado. Es un requesito en Unix para soporte de las opciones de tabla DATA DIRECTORY y INDEX DIRECTORY .

- init_connect

  Una cadena de caracteres que ejecuta el servidor para cada cliente que se conecta. La cadena consiste en uno o más comandos SQL. Para especificar varios comandos, sepárelos con un punto y coma. Por ejemplo, cada cliente arranca por defecto con modo autocommit activado. No hay ninguna variable de servidor global para especificar que autocommit esté desactivado por defecto, pero puede usar init_connect para conseguir el mismo efecto:

  ```
  SET GLOBAL init_connect='SET AUTOCOMMIT=0';
  ```

  Esta variable puede cambiarse en la línea de comandos o en un fichero de opciones. Para cambiar la variable tal y como se ha mostrado usando un fichero de opciones, incluya las siguientes líneas:

  ```
  [mysqld]
  init_connect='SET AUTOCOMMIT=0'
  ```

  Tenga en cuenta que el contenido de init_connect no se ejecuta para usuarios con el privilegio SUPER; esto es para el caso que el contenido se haya introducido incorrectamente (contiene una consulta incorrecta, por ejemplo con un error de sintaxis), haciendo que todas las conexiones fallen. No ejecutarlo para usuarios SUPER permite que éstos abran una conexión y arreglen init_connect.

- init_file

  El nombre del fichero especificado con la opción --init-file cuando arranca el servidor. Este fichero contiene comandos SQL que se ejecutan al arrancar el servidor. Cada comando debe estar en una única línea y no debe incluir comentarios.

- init_slave

  Esta variable es similar a init_connect, pero es una cadena de caracteres que se ejecuta por parte de un servidor esclavo cada vez que el thread SQL arranca. El formato de la cadena de caracteres es el mismo que para la variable init_connect.

- innodb_*xxx*

Las variables `InnoDB` de sistema se listan en Sección 15.4, “Opciones de arranque de `InnoDB`”.

- `interactive_timeout`

  El número de segundos que espera el servidor para actividad en una conexión interactiva antes de cerrarla. Un cliente interactivo se define como un cliente que usa la opción `CLIENT_INTERACTIVE` para `mysql_real_connect()`. Consulte también `wait_timeout`.

- `join_buffer_size`

  El tamaño del buffer que se usa para full joins (joins que no usan índices). Normalmente la mejor forma de conseguir joins rápidos es añadir índices. Incrementar el valor de `join_buffer_size` para obtener un full join más rápido cuando se añaden índices no es posible. Un buffer para joins se reserva para cada full join entre dos tablas. Para un join complejo entre varias tablas en que sus índices no se usan, buffers para joins múltiples pueden ser necesarios.

- `key_buffer_size`

  Los bloques de índices para tablas `MyISAM` y `ISAM` se guardan en buffers y se comparten para todos los threads. `key_buffer_size` es el tamaño del buffer usado para los bloques de índices. El key buffer también se conoce como la key cache.

  El tamaño máximo permitido para `key_buffer_size` es 4GB. El máximo tamaño efectivo puede ser menor, dependiendo de la memoria RAM física y los límites por proceso de RAM impuestos por el sistema operativo o la plataforma hardware.

  Incremente el valor para obtener un mejor tratamiento de índices (para todas las lecturas y escrituras múltiples) al máximo que pueda permitirse. El uso de un valor que sea el 25% del total de memoria en una máquina que principalmente ejecute MySQL es bastante común. Sin embargo, si el valor es demasiado grande (por ejemplo, más del 50% del total de la memoria) el sistema puede empezar a paginar y ser extremadamente lento. MySQL confía en el sistema operativo para tratar el cacheo del sistema de ficheros para las lecturas de datos, así que debe dejar algún espacio para cachear el sistema de ficheros.

  Para más velocidad al escribir varios registros al mismo tiempo , use `LOCK TABLES`. Consulte Sección 13.4.5, “Sintaxis de `LOCK TABLES` y `UNLOCK TABLES`”.

  Puede chequear el rendimiento del key buffer con el comando `SHOW STATUS` y examinando las variables de estado `Key_read_requests`, `Key_reads`, `Key_write_requests`, y `Key_writes` . Consulte Sección 13.5.4, “Sintaxis de `SHOW`”.

  El ratio `Key_reads/Key_read_requests` normalmente debe ser menor a 0.01. El ratio `Key_writes/Key_write_requests` normalmente está cerca de 1 si usa mayoritariamente actualizaciones y borrados, pero puede ser mucho menor si tiende a hacer actualizaciones que afecten a muchos registros al mismo tiempo o si usa la opción de tabla `DELAY_KEY_WRITE` .

  La fracción del key buffer en uso puede determinarse mediante `key_buffer_size` conjuntamente con la variable de estado `Key_blocks_unused` y el tamaño de bloque del buffer. El tamaño de bloque del buffer está disponible en la variable de servidor `key_cache_block_size` . La fracción del buffer en uso es:

  ```
  1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
  ```

  Este valor es una aproximación ya que algún espacio del key buffer puede estar reservado internamente para estructuras administrativas.

En MySQL 5.0, es posible crear múltiples key caches para tablas MyISAM. El límite de tamaño de 4GB se aplica a cada caché individualmente, no como grupo. Consulte Sección 7.4.6, “La caché de claves de MyISAM”.

- `key_cache_age_threshold`

  Este valor controla la demotion de buffers desde la sub-cadena caliente de una key cache a la sub-cadena templada. Los valores más pequeño provocan que la demotion ocurra más rápidamenet. El valor mínimo es 100. El valor por defecto es 300. Consulte Sección 7.4.6, “La caché de claves de MyISAM”.

- `key_cache_block_size`

  El tamaño en bytes de los bloques de la key cache. El valor por defecto es 1024. Consulte Sección 7.4.6, “La caché de claves de MyISAM”.

- `key_cache_division_limit`

  El punto de división entre las sub-cadenas calientes y tempaladas de la cadena del buffer de la key cache. El valor es el porcentaje de la cadena del buffer para usar en la sub-cadena templada. El rango de los valores permitidos es de 1 a 100. El valor por defecto es 100. Consulte Sección 7.4.6, “La caché de claves de MyISAM”.

- `language`

  El idioma usado para los lenguajes de error.

- `large_file_support`

  Si `mysqld` está compilado con la opción de soporte para grandes ficheros.

- `large_pages`

  Indica si está activado el soporte para páginas grandes. Esta variable se añadió en MySQL 5.0.3.

- `license`

  El tipo de licencia que tiene el servidor.

- `local_infile`

  Si `LOCAL` está soportado para comandos `LOAD DATA INFILE`.

- `locked_in_memory`

  Si `mysqld` está bloqueado en memoria con `--memlock`.

- `log`

  Si el logueo de todas las consultas en el log general de consultas está activado. Consulte Sección 5.10.2, “El registro general de consultas”.

- `log_bin`

  Si el log binario está activado. Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

- log_bin_trust_routine_creators

Esta variable se aplica cuando el log binario está activado. Controla si los creadores de rutinas almacenadas son o no de confianza para crear rutinas almacenadas que escribirán eventos no seguros en el log binario. Si está a 0 (por defecto), los usuarios no tienen permiso para crear o cambiar rutinas almacenadas a no ser que tengan el privilegio `SUPER` además del privilegio `CREATE ROUTINE` o `ALTER ROUTINE` .

Asignar el valor 0 fuerza la restricción que una rutina debe declararse con la característica `DETERMINISTIC` , o con las características `READS SQL DATA` o `NO SQL` . Si la variable tiene el valor 1, MySQL no fuerza esta restricción en la creación de rutinas almacenadas.

Consulte Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

Esta variable se añadió en MySQL 5.0.6.

- `log_error`

  La localización del log de errores.

- `log_slave_updates`

  Si las actualizaciones recibidas por un servidor esclavo de un servidor maestro deben loguearse en el log binario del servidor esclavo. El logueo binario debe estar disponible en el esclavo para que tenga efecto. Consulte Sección 6.8, “Opciones de arranque de replicación”.

- `log_slow_queries`

  Si una consulta lenta debe loguearse. ``Lenta'' está determinado por el valor de la variable `long_query_time`. Consulte Sección 5.10.4, “El registro de consultas lentas (Slow Query Log)”.

- `log_warnings`

  Si produce mensajes de advertencia adicionales. Está activado por defecto. Las conexiones abortadas no se loguean en el log de error a no ser que el valor sea mayor a 1.

- `long_query_time`

  Si una consulta dura más que este valor en segundos, la variable de estado `Slow_queries` se incrementa. Si está usando la opción `--log-slow-queries` , la consulta se loguea en el fichero de log para consultas lentas. Este valor se mide en tiempo real, no tiempo de CPU, así que una consulta que esté bajo el umbral en un sistema con poca carga puede estar por encima del umbral en un sistema con mucha carga. Consulte Sección 5.10.4, “El registro de consultas lentas (Slow Query Log)”.

- `low_priority_updates`

  Con el valor `1`, todos los comandos `INSERT`, `UPDATE`, `DELETE`, y `LOCK TABLE WRITE` esperan hasta que no hayan pendientes `SELECT` o `LOCK TABLE READ` en la tabla afectada. Esta variable se llamaba préviamente `sql_low_priority_updates`.

- `lower_case_file_system`

  Esta variable indica si el sistema de ficheros donde el directorio de datos está localizado tiene nombres de ficheros no sensibles a mayúsculas y minúsculas. `ON` significa que los nombres del fichero no son sensibles a mayúsculas y minúsculas, `OFF` significa que son sensibles a mayúsculas y minúsculas.

- `lower_case_table_names`

Con el valor 1, los nombres de tablas se almacenan en minúsuculas en disco y las comparaciones de nombres de tablas no tienen en cuenta mayúsculas y minúsculas. Esta opción también se aplica a nombres de bases de datos y alias de tablas. Consulte Sección 9.2.2, “Sensibilidad a mayúsuclas y minúsculas de identificadores”.

Si usa tablas `InnoDB` , debe asignar a esta variable el valor 1 en todas las plataformas para forzar que los nombres se conviertan a minúsculas.

No debe asignar a esta variable el valor 0 si está ejecutando MySQL en un sistema que no tiene nombres de ficheros sensibles a mayúsculas y minúsculas (tales como Windows o Mac OS X). Si esta variable no tiene un valor al arrancar y el sistema de ficheros donde está localizado el directorio de datos no tiene nombres de ficheros sensibles a mayúsculas y minúsculas, MySQL automáticamente asigna a `lower_case_table_names` el valor 2.

- `max_allowed_packet`

  El tamaño máximo de un paquete o cualquier cadena de caracteres generada/intermedia.

  El buffer de paquetes de mensajes se inicializa a `net_buffer_length` bytes, pero puede crecer hasta `max_allowed_packet` bytes cuando sea necesario. Este valor por defecto es pequeño, para recibir paquetes grandes (posiblemente incorrectos).

  Debe incrementar este valor si está usando columnas grandes `BLOB` o cadenas de caracteres largas. Debe ser tan grande como el mayor `BLOB` que quiera usar. En MySQL 5.0, el límite del protocolo para `max_allowed_packet` es 1GB.

- `max_binlog_cache_size`

  Si una transacción con múltiples comandos requiere más que esta cantidad de memoria, obtiene el error `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage`.

- `max_binlog_size`

  Si una escritura en el log binario excede el valor dado, rota el log binario. No puede cambiar este valor a más de 1GB o a menos de 4096 bytes. El valor por defecto es 1GB.

  Nota en caso que use transacciones: Una transacción se escribe en un trozo del log binario, así que nunca se trocea entre varios logs binarios. Por lo tanto, si tiene transacciones grandes, puede ver logs binarios mayores que `max_binlog_size`.

  Si `max_relay_log_size` es 0, el valor de `max_binlog_size` se aplica a relay logs también.

- `max_connect_errors`

  Si hay más que este número de conexiones interrumpidas de un equipo, este equipo se bloquea para más conexiones. Puede desbloquear equipos bloqueados con el comando `FLUSH HOSTS`.

- `max_connections`

  El número de conexiones de cliente simultáneas permitidas. Incrementar este valor incrementa el número de descriptores de fichero que requiere `mysqld` . Consulte Sección 7.4.8, “Cómo abre y cierra tablas MySQL” para comentarios sobre límites en los descriptores de fichero. Consulte también Sección A.2.6, “`Demasiadas conexiones`”.

- `max_delayed_threads`

No arranque más que este número de threads para tratar comandos `INSERT DELAYED` . Si trata de insertar datos en una nueva tabla después de que todos los threads `INSERT DELAYED` estén en uso, el registro se inserta como si el atributo `DELAYED` no se hubiera especificado. Si cambia este valor a 0, MySQL nunca crea un thread para tratar registros `DELAYED` ; en efecto, esto desactiva `DELAYED` completamente.

- `max_error_count`

  El número máximo de mensajes de error, advertencia, y notas que se guardan para mostrar con `SHOW ERRORS` o `SHOW WARNINGS`.

- `max_heap_table_size`

  Esta variable cambia el máximo valor para el que las tablas `MEMORY` (`HEAP`) pueden crecer. El valor de la variable se usa para calcular el valor `MAX_ROWS` en tablas `MEMORY`. Cambiar esta variable no tiene efecto en ninguna tabla `MEMORY` existente, a no ser que la tabla se recree con un comando tal como `CREATE TABLE` o `TRUNCATE TABLE`, o cambiada con `ALTER TABLE`.

- `max_insert_delayed_threads`

  Esta variable es sinónimo de `max_delayed_threads`.

- `max_join_size`

  No permite los comandos `SELECT` que probablemente tengan que examinar más de `max_join_size` registros (para comandos en una misma tabla) o combinaciones de registros (para comandos con varias tablas) o que sea probable que hagan más de `max_join_size` búsquedas de disco. Al cambiar este valor, puede cazar los comandos `SELECT` donde las claves no se usan correctamente y que probablemente tardaran mucho tiempo. Cámbielo si sus usuarios tienden a realizar joins sin una cláusula `WHERE` , que tarden mucho tiempo, o que devuelvan millones de registros.

  Al cambiar esta variable a un valor que no sea `DEFAULT` resetea el valor de `SQL_BIG_SELECTS` a `0`. Si cambia el valor `SQL_BIG_SELECTS` de nuevo, la variable `max_join_size` se ignora.

  Si el resultado de una consulta está en la caché de consultas, no se hace el chequeo del tamaño del resultado, ya que el resultado se ha computado préviamente y no carga al servidor al enviarlo al cliente.

  Esta variable préviamente se llamaba `sql_max_join_size`.

- `max_length_for_sort_data`

  El límite en el tamaño del índice que determina qué algoritmo `filesort` se usa. Consulte Sección 7.2.10, “Cómo optimiza MySQL `ORDER BY`”.

- `max_relay_log_size`

  Si una escritura de un esclavo de replicación a su relay log excede el valor dado, rota el relay log. Esta variable le permite poner diferentes restricciones de tamaño en logs binarios y relay logs. Sin embargo, cambiar la variable a 0 hace que MySQL use `max_binlog_size` para el log binario y el relay log. Debe asignar un valor a `max_relay_log_size` entre 4096 bytes y 1GB (incluídos), o 0. El valor por defecto es 0. Consulte Sección 6.3, “Detalles de la implementación de la replicación”.

- `max_seeks_for_key`

  Limita el máximo número de búsquedas al buscar registros basados en una clave. El optimizador de MySQL asume que no se requieren más que este número de búsquedas de claves al buscar

registros en una tabla al escanear una clave, a pesar de la cardinalidad real de la clave (consulte Sección 13.5.4.11, “Sintaxis de SHOW INDEX”). Si se pone un valor pequeño (100?), puede forzar a MySQL a que prefiera claves en lugar de escaneos de tablas.

- `max_sort_length`

  Número de bytes que se usan al ordenar valores `BLOB` o `TEXT` . Sólo los primeros `max_sort_length` bytes cada valor se usan; el resto se ignoran.

- `max_tmp_tables`

  El número máximo de tablas temporales que un cliente puede mantener abiertas al mismo tiempo. (Esta opción todavía no hace nada.)

- `max_user_connections`

  El número máximo de conexiones simultáneas permitidas para cualquier cuenta MySQL dada. Un valor 0 significa ``sin límites.''

  Antes de MySQL 5.0.3, esta variable sólo tenía visibilidad global. A partir de MySQL 5.0.3, también tiene visibilidad para sesiones de sólo lectura. La variable de sesión tiene el mismo valor que la variable global a no ser que la cuenta actual tenga un valor diferente a 0 en el límite de recursos `MAX_USER_CONNECTIONS` . En este caso, el valor de sesión refleja el límite de la cuenta.

- `max_write_lock_count`

  Después de varios bloqueos de escritura, permite que algunos bloqueos de lectura se ejecuten en medio.

- `multi_read_range`

  Especifica el máximo número de rangos a enviar a un motor de almacenamiento durante la selección de rangos. El valor por defecto es 256. Enviar múltiples rangos a un motor es una característica que puede mejorar el rendimiento de ciertas selecciones dramáticamente, particularmente para `NDBCLUSTER`. Este motor necesita enviar la petición de rangos a todos los nodos, y enviar muchas de estas peticiones a la vez reduce el coste de comunicación significativamente. Esta variable se añadió en MySQL 5.0.3.

- `myisam_data_pointer_size`

  El tamaño por defecto para punteros en bytes, para ser usado con `CREATE TABLE` para tablas `MyISAM` cuando no se especifica la opción `MAX_ROWS` . Esta variable no puede ser menor que 2 ni mayor que 7. El valor por defecto es 4. Consulte Sección A.2.11, “`The table is full`”.

- (*OBSOLETO*) `myisam_max_extra_sort_file_size`

  Si el fichero temporal usado para índexación rápida `MyISAM` es mayor que usando la key cache con la cantidad especificada aquí, se utiliza preferentemente el método de la key cache. Esto se usa para forzar que las claves largas de caracteres en grandes tablas usen el método de de key cache (más lento) para crear el índice. Este valor se da en bytes.

  Nota: Esta variable se eliminó en MySQL 5.0.6.

- `myisam_max_sort_file_size`

  El tamaño máximo para tablas temporales que MySQL permite para recrear un índice `MyISAM` (durante `REPAIR TABLE`, `ALTER TABLE`, o `LOAD DATA INFILE`). Si el tamaño del fichero fuese mayor que este valor, el índice se crea usando la key cache en su lugar, lo cual es más lento. El valor se da en bytes.

- `myisam_recover_options`

  El valor para la opción `--myisam-recover`.

- `myisam_repair_threads`

  Si este valor es mayor que 1, los índices de tablas `MyISAM` se crean en paralelo (cada índice en su propio thread) durante el proceso `Repair by sorting` . El valor por defecto es 1. **Nota**: Reparación multi-threaded repair todavía es código con calidad *alpha*.

- `myisam_sort_buffer_size`

  El buffer que se reserva al ordenar índices `MyISAM` durante `REPAIR TABLE` o al crear índices con `CREATE INDEX` o `ALTER TABLE`.

- `named_pipe`

  (Sólo en Windows.) Indica si el servidor soporta conexiones sobre named pipes.

- `net_buffer_length`

  El buffer de comunicación se resetea a este tamaño entre consultas. Normalmente, debe cambiarse, pero si tiene poca memoria, puede inicializarse al tamaño esperado para los comandos SQL enviados por los clientes. Si los comandos exceden este tamaño, el buffer crece automáticamente, hasta `max_allowed_packet` bytes.

- `net_read_timeout`

  El número de segundos a esperar más datos de una conexión antes de abortar la lectura. Cuando el servidor está leyendo del cliente, `net_read_timeout` es el valor que controla el tiempo máximo para abortar. Cuando el servidor está escribiendo en el cliente, `net_write_timeout` es el valor que controla el tiempo máximo para abortar. Consulte también `slave_net_timeout`.

- `net_retry_count`

  Si una lectura en un puerto de comunicaciones se interrumpe, reintenta las veces especificadas antes de abandonar. Este valor debe inicializarse a un valor alto en FreeBSD, ya que las interrupciones internas se envían a todos los threads.

- `net_write_timeout`

  Número de segundos a esperar para que se escriba un bloque en una conexión antes de abortar la escritura. Consulte también `net_read_timeout`.

- `new`

  Esta variable se usaba en MySQL 4.0 para activar algunos comportamientos de la versión 4.1 y se mantiene para compatibilidad con versiones anteriores. En MySQL 5.0, el valor siempre es `OFF`.

- `old_passwords`

  Si el servidor debe usar contraseñas estilo anterior a la versión 4.1 para cuentas de usuario de MySQL. Consulte Sección A.2.3, “`Client does not support authentication protocol`”.

- `open_files_limit`

  Número de ficheros que el sistema operativo permite abrir `mysqld` . Este es el valor real permitido por el sistema y puede ser distinto del valor que se da a `mysqld` al arrancar. El valor es 0 en sistemas donde MySQL puede cambiar el número de ficheros abiertos.

- `optimizer_prune_level`

  Controla el heurístico aplicado durante la optimización de consultas para no hacer los planes parciales menos prometedores en el espacio de búsqueda del optimizador. Un valor de 0 desactiva el heurístico de forma que el optimizador realiza una búsqueda exhaustiva. Un valor de 1 provoca que el optimizador elimine planes basados en el número de registros calculados por planes intermedios. Esta variable se añadió en MySQL 5.0.1.

- `optimizer_search_depth`

  La máxima profundidad de búsqueda realizada por el optimizador de consultas. Valores mayores que el número de relaciones en una consulta dan como resultado en mejores planes de consulta, pero lleva más tiempo generar un plan de ejecución para una consulta. Los valores menores que el número de relaciones en una consulta retornan un plan de ejecución más rápido, pero el plan resultante puede estar lejos de ser el plan óptimo. Si se usa como valor 0, el sistema automáticamente escoge un valor razonable. Si se asigna como valor el máximo número de tablas usado en una consulta más 2, el optimizador cambia al algoritmo usado en MySQL 5.0.0 (y versiones anteriores) para mejorar el rendimiento de las búsquedas. Esta variable se añadió en MySQL 5.0.1.

- `pid_file`

  La ruta al fichero con el ID de proceso (PID). Esta variable puede cambiarse con la opción `--pid-file`.

- `port`

  El puerto en que escucha el servidor para conexiones TCP/IP. Esta variable puede cambiarse con la opción `--port` .

- `preload_buffer_size`

  El tamaño del buffer que se reserva al pre-cargar los índices.

- `protocol_version`

  La versión del protocolo cliente/servidor usado por el servidor MySQL.

- `query_alloc_block_size`

  El tamaño de bloques de memoria reservado para objetos creados durante el parseo y ejecución de consultas. Si tiene problemas con fragmentación de memoria, puede ayudar incrementar este valor un poco.

- `query_cache_limit`

  No cachea resultados mayores que este número de bytes. El valor por defecto es 1048576 (1MB).

- `query_cache_min_res_unit`

  Tamaño mínimo (en bytes) para bloques reservados en la caché de consultas. El valor por defecto es 4096 (4KB). Para más información acerca de optimizar esta variable consulte Sección 5.12.3, “Configuración de la caché de consultas”.

- `query_cache_size`

  La cantidad de memoria reservada para cachear resultados de consultas. El valor por defecto es 0, lo que desactiva la cache de consultas. Tenga en cuenta que la cantidad de memoria se reserva incluso

si `query_cache_type` tiene como valor 0. Consulte Sección 5.12.3, “Configuración de la caché de consultas” para más información.

- `query_cache_type`

  Cambia el tipo de la caché de consultas. Cambiando el valor `GLOBAL` se inicializa el tipo para todos los clientes que se conecten a partir de ese momento. Clientes individuales pueden cambiar el valor `SESSION` para afectar a su propio uso de la caché de consultas. Los posibles valores se muestran en la siguiente tabla:

Opción	Descripción
`0` o `OFF`	No cachea o retorna los resultados. Tenga en cuenta que esto no elimina el buffer caché para consultas. Para hacerlo, debe asignar 0 a `query_cache_size`.
`1` o `ON`	Cachea todos los resultados de consultas excepto los que empiecen con `SELECT SQL_NO_CACHE`.
`2` o `DEMAND`	Cachea los resultados sólo para consultas que comiencen con `SELECT SQL_CACHE`.

  En MySQL 5.0, el valor por defecto de esta variable es `ON`.

- `query_cache_wlock_invalidate`

  Normalmente, cuando un cliente adquiere un bloqueo `WRITE` en una tabla `MyISAM` , el resto de clientes no se bloquean para consultas cuyo resultado esté presente en la caché para consultas. Cambiando el valor de esta variable a 1 provoca que la adquisición de una tabla mediante un bloqueo `WRITE` invalida cualquier consulta en la caché de consultas que se refiera a la tabla. Esto fuerza a que otros clientes que traten de acceder a la tabla esperen mientras el bloqueo esté activo.

- `query_prealloc_size`

  El tamaño del buffer persistente usado para parsear y ejecutar consultas. Este buffer no se libera entre consultas. Si está ejecutando consultas complejas, un valor mayor de `query_prealloc_size` puede ser de utilidad para mejorar el rendimiento, ya que puede reducir la necesidad del servidor de realizar reserva de memoria durante las operaciones de ejecución de consultas.

- `range_alloc_block_size`

  El tamaño de los bloques que se reservan en la optimización de rango.

- `read_buffer_size`

  Cada thread que realiza un escaneo secuencial reserva un buffer de su tamaño (en bytes) para cada tabla que escanea. Si realiza muchos escaneos secuenciales, puede incrementar este valor, que por defecto es 131072.

- `read_only`

  Cuando una variable tiene el valor `ON` para un servidor esclavo de replicación, esto causa que el esclavo no permita actualizaciones excepto de threads de esclavos o de usuarios con el privilegio `SUPER`. Esto puede ser útil para asegurar que un servidor esclavo no acepte actualizaciones de los clientes.

- `relay_log_purge`

  Desactiva o activa el purgado automático de los relay logs tan pronto como no se necesitan. El valor por defecto es 1 (activado).

- `read_rnd_buffer_size`

  Cuando se leen registros ordenadamante tras una ordenación, los registros se leen a través de su buffer para evitar búsquedas en disco. Asignar a esta variable un valor mayor puede mejorar mucho el rendimiento de `ORDER BY`. Sin embargo, hay un buffer para cada cliente, así que no debe asignar un valor grande a la variable global. En lugar de ello, cambie la variable de sesión sólo en los clientes que necesiten ejecutar grandes consultas.

- `secure_auth`

  Si el servidor MySQL ha arrancado con la opción `--secure-auth` , bloquea conexiones de todas las cuentas que tengan las contraseñas almacenadas en el formato antiguo (anterior a 4.1). En ese caso, el valor de esta variable es `ON`, en el contrario es `OFF`.

  Debe activar esta opción si quiere evitar todo uso de contraseñas en el viejo formato (y por lo tanto comunicación insegura en la red).

  El arranque del servidor falla con un error si esta opción está activada y la tablas de privilegios están en formato anterior a 4.1. Consulte Sección A.2.3, “`Client does not support authentication protocol`”.

  Cuando se usa como una opción del lado del cliente, el cliente rehúsa conectar a un servidor si el servidor requiere contraseña en el viejo formato para la cuenta del cliente.

- `server_id`

  El valor de la opción `--server-id` . Se usa para servidores de replicación maestros y esclavos.

- `shared_memory`

  (Sólo en Windows.) Si el cliente permite conexiones a través de memoria compartida o no.

- `shared_memory_base_name`

  (Sólo en Windows.) Indica si el servidor permite conexiones a través de memoria compartida, e indica el identificador para memoria compartida. Esto es útil al ejecutar varias instancias de MySQL en una única máquina física.

- `skip_external_locking`

  Está `OFF` si `mysqld` usa bloqueo externo.

- `skip_networking`

  Está `ON` si el servidor permite sólo conexiones locales (no TCP/IP). En Unix, las conexiones locales usan un fichero socket de Unix. En Windows, las conexiones locales usan named pipes o memoria compartida. En NetWare, sólo se soportan conexiones TCP/IP, así que no asigne a esta variable el valor `ON`.

- `skip_show_database`

  Evita que se use el comando `SHOW DATABASES` sin tener el privilegio `SHOW DATABASES` . Esto puede mejorar la seguridad si le preocupa que los usuarios puedan ver las bases de datos pertenecientes a otros usuarios. En MySQL 5.0, su efecto depende del privilegio `SHOW DATABASES`: Si el valor de la variable es `ON`, el comando `SHOW DATABASES` está permitido sólo para usuarios que tengan el privilegio `SHOW DATABASES` , y el comando muestra todos los nombres de bases de datos. Si el valor es `OFF`,

`SHOW DATABASES` se permite para todos los usuarios, pero sólo muestra los nombres de bases de datos que el usuario tenga el permiso `SHOW DATABASES` u otros privilegios.

- `slave_compressed_protocol`

  Si usa compresión en el protocolo maestro/servidor si ambos lo soportan.

- `slave_load_tmpdir`

  El nombre del directorio donde el esclavo crea ficheros temporales para replicar el comando `LOAD DATA INFILE`.

- `slave_net_timeout`

  Número de segundos a esperar para más datos en una conexión maestro/ servidor antes de abortar la lectura.

- `slave_skip_errors`

  Los errores de replicación que el esclavo debe dejar pasar (ignorar).

- `slow_launch_time`

  Si la creación de un thread toma más de los segundos especificados por esta variable, el servidor incrementa la variable de estado `Slow_launch_threads` .

- `socket`

  Plataformas Unix: El fichero socket usado para conexiones clientes locales. Por defecto `/var/lib/mysql/mysql.sock`.

  Windows: El nombre del named pipe usado para conexiones cliente locales. Por defecto `mysql`.

- `sort_buffer_size`

  Cada thread que necesita una ordenación reserva un buffer de su tamaño. El incremento de este valor permite acelerar las operaciones `ORDER BY` o `GROUP BY` . Consulte Sección A.4.4, “Dónde almacena MySQL los archivos temporales”.

- `sql_mode`

  El modo del servidor SQL, que en MySQL 5.0 puede cambiarse dinámicamente. Consulte Sección 5.3.2, “El modo SQL del servidor”.

- `sql_slave_skip_counter`

  El número de eventos del maestro que el servidor esclavo debe ignorar.

- `storage_engine`

  Esta variable es un sinónimo de `table_type`. En MySQL 5.0, `storage_engine` es el nombre preferido.

- `sync_binlog`

  Si es positivo, el servidor MySQL sincroniza su log binario a disco (`fdatasync()`) después de cada escritura `sync_binlog` en su log binario. Tenga en cuenta que sólo hay una escritura en el log binario por comando si está en modo autocommit, y de otra forma una escritura por transacción. El valor por defecto es 0 lo que no sincroniza con disco. Un valor de 1 es la elección más segura, ya que en caso

de error se pierden como mucho un comando/transacción del log binario; de todas formas, también es el valor más bajo (a no ser que el disco tenga una caché de batería, lo que hace la sincronización muy rápida).

- `sync_frm`

  Si esta variable se asigna a 1, cuando se crea una tabla no temporal su fichero `.frm` se sincroniza a disco (`fdatasync()`); esto es lo más lento pero más seguro en caso de un error. El valor por defecto es 1.

- `system_time_zone`

  La zona horaria del servidor. Cuando el servidor comienza la ejecución, hereda unos valores de zona horaria de los valores por defecto de la máquina, posiblemente modificados por el entorno de la cuenta usado para ejecutar el servidor o el script de arranque. El valor se usa para asignar `system_time_zone`. Normalmente la zona horaria se especifica con la variable de entorno `TZ`. También puede especificarse usando la opción `--timezone` del script `mysqld_safe` .

- `table_cache`

  El número de tablas abiertas por todos los threads. Incrementar este valor incrementa el número de descriptores de ficheros que requiere `mysqld` . Puede chequear si necesita incrementar la caché de la tabla chequeando la variable de estado `Opened_tables` . Consulte Sección 5.3.4, “Variables de estado del servidor”. Si el valor de `Opened_tables` es grande y no quiere hacer muchos `FLUSH TABLES` (lo que fuerza a cerrar y reabrir todas las tablas), entonces debe incrementar el valor de la variable `table_cache` .

  Para más información sobre la cache de la tabla, consulte Sección 7.4.8, “Cómo abre y cierra tablas MySQL”.

- `table_type`

  El tipo de tabla por defecto (motor de almacenamiento). Para cambiar el tipo de tabla en el arranque del servidor, use la opción `--default-table-type`. Consulte Sección 5.3.1, “Opciones del comando `mysqld`”.

- `thread_cache_size`

  El número de threads que el servidor debe cachear para reusar. Cuando un cliente desconecta, los threads de clientes se ponen en la caché si hay menos de `thread_cache_size` threads. Peticiones de threads se sirven reusando threads tomados de la caché cuando es posible. Esta variable puede incrementarse para mejorar el rendimiento si tiene muchas nuevas conexiones. (Normalmente esto no da una mejora notable de rendimiento si tiene una buena implementación de threads.) Mediante el estudio de las diferencias entre las variables de estado `Connections` y `Threads_created` (consulte Sección 5.3.4, “Variables de estado del servidor” para más detalles) puede ver lo eficiente que es la caché de threads.

- `thread_concurrency`

  En Solaris, `mysqld` llama a `thr_setconcurrency()` con este valor. Esta función permite a las aplicaciones dar al sistema de threads una piesta sobre el número deseado de threads que deben ejecutarse simultáneamente.

- `thread_stack`

El tamaño de la pila para cada thread. Muchos de los límites detectados por el test `crash-me` dependen de este valor. El valor por defecto es lo suficientemente grande para un funcionamiento normal. Consulte Sección 7.1.4, “El paquete de pruebas de rendimiento (benchmarks) de MySQL”.

- `time_zone`

  La zona horaria. El valor inical de este `'SYSTEMA'` (usa el valor de `system_time_zone`), pero puede especificarse explícitamente al arrancar el servidor con la opción `--default-time-zone` .

- `tmp_table_size`

  Si una tabla temporal en memoria excede este tamaño, MySQL la convierte automáticamente en una tabla en disco de tipo `MyISAM`. Incremente el valor de `tmp_table_size` si realiza muchas consultas avanzadas `GROUP BY` y tiene mucha memoria disponible.

- `tmpdir`

  El directorio usado para ficheros y tablas temporales. Esta variable puede tomar una lista de diferentes paths usados con una política round-robin. Estos paths pueden separarse mediante dos puntos (`':'`) en Unix y punto y coma (`';'`) en Windows, NetWare, y OS/2.

  Esta característica puede usarse para repartir la carga entre varios discos físicos. Si el servidor MySQL actúa como un esclavo de replicación, no debe asignar `tmpdir` un valor que apunte a un directorio en un sistema de ficheros en memoria o a un directorio que se borre cada vez que el servidor reinicie. Un esclavo de replicación necesita que algunos de sus ficheros temporales sobrevivan a un reinicio de servidor de forma que pueeda replicar tablas temporales u operaciones `LOAD DATA INFILE` . Si los ficheros en un directorio de ficheros temporales se pierden cuando reinicia el servidor, la replicación falla.

- `transaction_alloc_block_size`

  El tamaño reservado (en bytes) de bloques de memoria que se reservan para almacenar consultas que son partes de una transacción que debe ser guardad en el log binario al hacer un commit.

- `transaction_prealloc_size`

  Tamaño en bytes del buffer persistente para `transaction_alloc_blocks` que no se libera entre consultas. Haciéndolo lo suficientemente grande para guardar todas las consultas dentro de una única transacción, puede evitar varias llamadas `malloc()` .

- `tx_isolation`

  El nivel de aislamiento de transacción por defecto. Por defecto es `REPEATABLE-READ`.

- `updatable_views_with_limit`

  Esta variable controla si las actualizaciones pueden hacerse usando una vista que no contenga una clave primária en la tabla subyacente, si la actualización contiene una cláusula `LIMIT` . (Tales actualizaciones normalmente las generan herramientas GUI.) Una actualización es un comando `UPDATE` o `DELETE`.Clave primaria se refiere a un índice `PRIMARY KEY`, o `UNIQUE` en el que ninguna columna puede contener `NULL`.

  La variable puede tener dos valores:

- `1` o `YES`: Muestra sólo una advertencia (no un mensaje de error). Este es el valor por defecto.
- `0` o `NO`: Prohibe la actualización.

Esta variable se añadió en MySQL 5.0.2.

- `version`

  El número de versión para el servidor.

- `version_bdb`

  La versión del motor de almacenamiento `BDB`.

- `version_comment`

  El script `configure` tiene una opción `--with-comment` que permite especificar un comentario al compilar MySQL. Esta variable contiene el valor de dicho comentario.

- `version_compile_machine`

  El tipo de máquina o arquitectura en el que se compiló MySQL.

- `version_compile_os`

  El tipo de sistema operativo en el que se compiló MySQL.

- `wait_timeout`

  Número de segundos que el servidor espera para recibir actividad en una conexión no interactiva antes de cerrarla.

  En el arranque de un thread, el valor de la variable de sesión `wait_timeout` se inicializa de la variable global `wait_timeout` o de `interactive_timeout` , dependiendo del tipo de cliente (como se define en la opción de conexión `CLIENT_INTERACTIVE` de `mysql_real_connect()`). Consulte también `interactive_timeout`.

5.3.3.1. Variables de sistema dinámicas

Varias variables de sistema del servidor son dinámicas y pueden cambiarse en tiempo de ejecución mediante `SET GLOBAL` o `SET SESSION`. También puede obtener sus valores usando `SELECT`. Consulte Sección 9.4, “Variables de sistema”.

La siguiente tabla muestra la lista completa de todas las variables dinámicas de sistema. La última columna indica para cada variable si son `GLOBAL` o `SESSION` (o ambas).

Nombre de Variable	Tipo de Valor	Tipo
`autocommit`	booleana	`SESSION`
`big_tables`	booleana	`SESSION`
`binlog_cache_size`	numérica	`GLOBAL`
`bulk_insert_buffer_size`	numérica	`GLOBAL` | `SESSION`
`character_set_client`	cadena de caracteres	`GLOBAL` | `SESSION`
`character_set_connection`	cadena de caracteres	`GLOBAL` | `SESSION`
`character_set_results`	cadena de caracteres	`GLOBAL` | `SESSION`

character_set_server	cadena de caracteres	GLOBAL | SESSION
collation_connection	cadena de caracteres	GLOBAL | SESSION
collation_server	cadena de caracteres	GLOBAL | SESSION
concurrent_insert	booleana	GLOBAL
connect_timeout	numérica	GLOBAL
convert_character_set	cadena de caracteres	GLOBAL | SESSION
default_week_format	numérica	GLOBAL | SESSION
delay_key_write	OFF | ON | ALL	GLOBAL
delayed_insert_limit	numérica	GLOBAL
delayed_insert_timeout	numérica	GLOBAL
delayed_queue_size	numérica	GLOBAL
error_count	numérica	SESSION
expire_logs_days	numérica	GLOBAL
flush	booleana	GLOBAL
flush_time	numérica	GLOBAL
foreign_key_checks	booleana	SESSION
ft_boolean_syntax	numérica	GLOBAL
group_concat_max_len	numérica	GLOBAL | SESSION
identity	numérica	SESSION
innodb_autoextend_increment	numérica	GLOBAL
innodb_concurrency_tickets	numérica	GLOBAL
innodb_max_dirty_pages_pct	numérica	GLOBAL
innodb_max_purge_lag	numérica	GLOBAL
innodb_sync_spin_loops	numérica	GLOBAL
innodb_table_locks	booleana	GLOBAL | SESSION
innodb_thread_concurrency	numérica GLOBAL	
innodb_thread_sleep_delay	numérica GLOBAL	
insert_id	booleana	SESSION
interactive_timeout	numérica	GLOBAL | SESSION
join_buffer_size	numérica	GLOBAL | SESSION
key_buffer_size	numérica	GLOBAL
last_insert_id	numérica	SESSION
local_infile	booleana	GLOBAL
log_warnings	numérica	GLOBAL
long_query_time	numeric	GLOBAL | SESSION
low_priority_updates	booleana	GLOBAL | SESSION
max_allowed_packet	numérica	GLOBAL | SESSION
max_binlog_cache_size	numérica	GLOBAL
max_binlog_size	numérica	GLOBAL

max_connect_errors	numérica	GLOBAL
max_connections	numérica	GLOBAL
max_delayed_threads	numérica	GLOBAL
max_error_count	numérica	GLOBAL | SESSION
max_heap_table_size	numérica	GLOBAL | SESSION
max_insert_delayed_threads	numérica	GLOBAL
max_join_size	numérica	GLOBAL | SESSION
max_relay_log_size	numérica	GLOBAL
max_seeks_for_key	numérica	GLOBAL | SESSION
max_sort_length	numérica	GLOBAL | SESSION
max_tmp_tables	numérica	GLOBAL | SESSION
max_user_connections	numérica	GLOBAL
max_write_lock_count	numérica	GLOBAL
multi_read_range	numérica	GLOBAL | SESSION
myisam_data_pointer_size	numérica	GLOBAL
log_bin_trust_routine_creators	booleana	GLOBAL
myisam_max_sort_file_size	numérica	GLOBAL | SESSION
myisam_repair_threads	numérica	GLOBAL | SESSION
myisam_sort_buffer_size	numérica	GLOBAL | SESSION
net_buffer_length	numérica	GLOBAL | SESSION
net_read_timeout	numérica	GLOBAL | SESSION
net_retry_count	numérica	GLOBAL | SESSION
net_write_timeout	numérica	GLOBAL | SESSION
old_passwords	numérica	GLOBAL | SESSION
optimizer_prune_level	numérica	GLOBAL | SESSION
optimizer_search_depth	numérica	GLOBAL | SESSION
preload_buffer_size	numérica	GLOBAL | SESSION
query_alloc_block_size	numérica	GLOBAL | SESSION
query_cache_limit	numérica	GLOBAL
query_cache_size	numérica	GLOBAL
query_cache_type	enumeración	GLOBAL | SESSION
query_cache_wlock_invalidate	booleana	GLOBAL | SESSION
query_prealloc_size	numérica	GLOBAL | SESSION
range_alloc_block_size	numérica	GLOBAL | SESSION
read_buffer_size	numérica	GLOBAL | SESSION
read_only	numérica	GLOBAL
read_rnd_buffer_size	numérica	GLOBAL | SESSION
rpl_recovery_rank	numérica	GLOBAL
safe_show_database	booleana	GLOBAL

secure_auth	booleana	GLOBAL
server_id	numérica	GLOBAL
slave_compressed_protocol	booleana	GLOBAL
slave_net_timeout	numérica	GLOBAL
slave_transaction_retries	numérica	GLOBAL
slow_launch_time	numérica	GLOBAL
sort_buffer_size	numérica	GLOBAL | SESSION
sql_auto_is_null	booleana	SESSION
sql_big_selects	booleana	SESSION
sql_big_tables	booleana	SESSION
sql_buffer_result	booleana	SESSION
sql_log_bin	booleana	SESSION
sql_log_off	booleana	SESSION
sql_log_update	booleana	SESSION
sql_low_priority_updates	booleana	GLOBAL | SESSION
sql_max_join_size	numérica	GLOBAL | SESSION
sql_mode	enumeración	GLOBAL | SESSION
sql_notes	booleana	SESSION
sql_quote_show_create	booleana	SESSION
sql_safe_updates	booleana	SESSION
sql_select_limit	numérica	SESSION
sql_slave_skip_counter	numérica	GLOBAL
updatable_views_with_limit	enumeración	GLOBAL | SESSION
sql_warnings	booleana	SESSION
sync_binlog	numérica	GLOBAL
sync_frm	booleana	GLOBAL
storage_engine	enumeración	GLOBAL | SESSION
table_cache	numérica	GLOBAL
table_type	enumeración	GLOBAL | SESSION
thread_cache_size	numérica	GLOBAL
time_zone	cadena de caracteres	GLOBAL | SESSION
timestamp	booleana	SESSION
tmp_table_size	enumeración	GLOBAL | SESSION
transaction_alloc_block_size	numérica	GLOBAL | SESSION
transaction_prealloc_size	numérica	GLOBAL | SESSION
tx_isolation	enumeración	GLOBAL | SESSION
unique_checks	booleana	SESSION
wait_timeout	numérica	GLOBAL | SESSION
warning_count	numérica	SESSION

Variables marcadas como **cadena de caracteres** toman un valor de cadena de caracteres. Variables marcadas como **numérica** toman un valor numérico. Variables marcadas como **boolean** toman como valor 0, 1, `ON` o `OFF`. Variables marcadas como **enumeración** normalmente deben tomar uno de los valores disponibles para la variable, pero también pueden tomar como valor el número correspondiente al valor deseado de la enumeración. Para variables de sistema enumeradas, el primer valor de la enumeración es 0. Esto difiere de las columnas `ENUM` , en que el primer valor de la enumeración es 1.

5.3.4. Variables de estado del servidor

El servidor mantiene muchas variables de estado que proveen de información sobre sus operaciones. Puede ver estas variables y sus valores utilizando la sentencia `SHOW STATUS`:

```
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_files                 | 3          |
| Created_tmp_tables                | 2          |

…

| Threads_created                   | 217        |
| Threads_running                   | 88         |
| Uptime                            | 1389872    |
+-----------------------------------+------------+
```

Muchas variables de estado son inicializadas a 0 por la sentencia `FLUSH STATUS`.

Las variables de estado tienen los siguientes significados. Las variables en las que no se indica versión están presentes en MySQL 5.0. Para información a propósito de su historial de implementación, consulte *Manual de referencia de MySQL 4.1*.

- `Aborted_clients`

  El número de conexiones que han sido abortadas debido a que el cliente murió sin cerrar la conexión apropiadamente. Consulte Sección A.2.10, “Errores de comunicación y conexiones abortadas”.

- `Aborted_connects`

  El número de intentos de conexión al servidor MySQL que han fallado. Consulte Sección A.2.10, “Errores de comunicación y conexiones abortadas”.

- `Binlog_cache_disk_use`

  El número de transacciones que han utilizado la cache temporal del registro binario pero han excedido el valor de `binlog_cache_size` y utilizado un archivo temporal para almacenar las sentencias de la transacción.

- `Binlog_cache_use`

  El número de transacciones que han utilizado la cache temporal del registro binario.

- `Bytes_received`

  El número de bytes recibidos desde todos los clientes.

- `Bytes_sent`

  El número de bytes enviados hacia todos los clientes.

- `Com_xxx`

  Las variables del contador de sentencias `Com_xxx` indica el número de veces que cada sentencia *`xxx`* ha sido ejecutada. Existe una variable de estado por cada tipo de sentencia. Por ejemplo `Com_delete` y `Com_insert` cuentan sentencias `DELETE` and `INSERT`, respectivamente.

  Las variables de estado `Com_stmt_xxx` fueron añadidas en 5.0.8:

  - `Com_stmt_prepare`
  - `Com_stmt_execute`
  - `Com_stmt_fetch`
  - `Com_stmt_send_long_data`
  - `Com_stmt_reset`
  - `Com_stmt_close`

  Estas variables representan comandos de sentencias preparadas. Sus nombres se refieren a los comandos `COM_xxx` utilizados en la capa de red; en otras palabras: Sus valores son incrementados siempre que las llamadas de la API de sentencias preparadas como `mysql_stmt_prepare()`, `mysql_stmt_execute()`, etc. son ejecutadas. Asímismo,`Com_stmt_prepare`, `Com_stmt_execute` y `Com_stmt_close` se incrementan también cuando alguien ejecuta las siguientes sentencias SQL: `PREPARE`, `EXECUTE`, or `DEALLOCATE` respectivamente. `Com_stmt_fetch` representa el número total de comunicaciones de red ejecutadas al recibir datos de los cursores

  Todas las variables `Com_stmt_xxx` son incrementadas aunque el argumento de una sentencia preparada sea desconocido u ocurra un error durante la ejecución. Es decir, sus valores corresponden al número de peticiones recibidas, no al número de peticiones completadas con éxito.

- `Connections`

  El número de instentos de conexión (con éxito o no) al servidor MySQL.

- `Created_tmp_disk_tables`

  El número de tablas temporales en disco creadas automáticamente por el servidor mientras ejecutaba sentencias.

- `Created_tmp_files`

  Número de arhcivos temporales que `mysqld` ha creado.

- `Created_tmp_tables`

  El número de tablas temporales en memoria creadas automáticamente por el servidor mientras ejecuta sentencias. Si `Created_tmp_disk_tables` es grande, quizá querría aumentar el valor de `tmp_table_size` para causar que las tablas sean basadas en memoria en vez de basadas en disco.

- `Delayed_errors`

  El número de registros escritos con `INSERT DELAYED` en los que algún error ocurrió (probablemente `duplicate key`).

- `Delayed_insert_threads`

  El número de hilos gestores de `INSERT DELAYED` en uso.

- `Delayed_writes`

  El número de registros `INSERT DELAYED` escritos

- `Flush_commands`

  El número de comandos `FLUSH` ejecutados.

- `Handler_commit`

  El número de sentencias `COMMIT` internas.

- `Handler_discover`

  El servidor MySQL puede preguntar al motor de almacenamiento `NDB Cluster` si conoce una tabla con un nombre dado. Esto se llama descubrimiento (discovery). `Handler_discover` indica el número de veces que se han descubierto tablas mediante este mecanismo.

- `Handler_delete`

  El número de veces que se han borrado registros de tablas.

- `Handler_read_first`

  El número de veces que se lee la primera entrada de un índice. Si este valor es alto, indica que el servidor está haciendo muchos escaneos "full index"; por ejemplo, `SELECT col1 FROM foo`, suponiendo que `col1` está indexada.

- `Handler_read_key`

  El número de peticiones para leer un registro basadas en una clave. Si este número es alto, es una buena indicación de que las consultas y tablas están indexadas adecuadamente.

- `Handler_read_next`

  El número de peticiones para leer el siguiente registro en el orden de la clave. Esto se incrementa cada vez que se está consultando una columna indexada con una restricción de rango, o si se está haciendo un escaneo de índice.

- `Handler_read_prev`

  El número de peticiones para leer el registro previo en el orden de la clave. Este método de lectura es utilizado principalmente para optimizar `ORDER BY ...DESC`.

- `Handler_read_rnd`

  El número de peticiones para leer un registro basándose en una posición fija. Es alto si se están haciendo muchas consultas que requieren ordenación del resultado. Probablemente tenga muchas consultas que requieran que MySQL escanee tablas completas o tiene joins que no utilizan claves adecuadamente.

- `Handler_read_rnd_next`

  El número de peticiones para leer el siguiente registro en el archivo de datos. Éste es alto si está haciendo muchos escaneos de tablas. Generalmente esto sugiere que las tablas no están indexadas adecuadamente o que las consultas no están escritas para obetner ventaja de los índices que se tienen.

- `Handler_rollback`

  El número de sentencias `ROLLBACK` internas.

- `Handler_update`

  El número de peticiones de modificación de un registro en una tabla.

- `Handler_write`

  El número de peticiones de inserción de un registro en una tabla.

- `Innodb_buffer_pool_pages_data`

  El número de páginas que contienen datos (procesados o no). Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_pages_dirty`

  El número de páginas sin procesar. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_pages_flushed`

  El número de páginas del buffer sobre las que se ha hecho una petición de volcado. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_pages_free`

  El número de páginas libres. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_pages_latched`

  El número de páginas bloqueadas en el buffer `InnoDB`. Estas son las páginas que están siendo actualmente leidas o escritas o que no pueden ser volcadas o borradas por alguna otra razón. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_pages_misc`

  El número de páginas ocupadas porque han sido seleccionadas para procesos administrativos, como bloqueos de registro o el índice de hash adaptativo. Este valor puede ser también calculado como `Innodb_buffer_pool_pages_total` - `Innodb_buffer_pool_pages_free` - `Innodb_buffer_pool_pages_data`. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_pages_total`

  Tamaño total del buffer, en páginas. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_read_ahead_rnd`

  El número de lecturas avanzadas ``aleatorias'' que `InnoDB` ha iniciado. Esto pasa cuando una consulta necesita escanear una gran poción de una tabla pero en orden aleatorio. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_read_ahead_seq`

El número de lecturas avanzadas secuenciales que `InnoDB` ha iniciado. Esto pasa cuando `InnoDB` realiza un escaneo secuencial completo de una tabla. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_read_requests`

  El número de peticiones lógicas de lectura que `InnoDB` ha hecho. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_reads`

  El número de lecturas lógicas del buffer que `InnoDB` no pudo satisfacer y tuvo que hacer una lectura de una única página. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_wait_free`

  Normalmente, las escrituras al buffer de `InnoDB` se llevan acabo en segundo plano. Aún así, si es necesario leer o crear una página y no existe ninguna página vacía disponible, entonces es también necesario esperar a que las páginas sean volcadas previamente. Este contador cuenta las instancias de estas esperas. Si el tamaño del buffer ha sido establecido correctamente, este valor debería ser pequeño. Añadido en MySQL 5.0.2.

- `Innodb_buffer_pool_write_requests`

  El número de escrituras hechas al buffer de `InnoDB`. Añadido en MySQL 5.0.2.

- `Innodb_data_fsyncs`

  El número de operaciones `fsync()` total. Añadido en MySQL 5.0.2.

- `Innodb_data_pending_fsyncs`

  El número de operaciones `fsync()` pendientes. Añadido en MySQL 5.0.2.

- `Innodb_data_pending_reads`

  El número actual de lecturas pendientes. Añadido en MySQL 5.0.2.

- `Innodb_data_pending_writes`

  El número actual de escrituras pendientes. Añadido en MySQL 5.0.2.

- `Innodb_data_read`

  El total de datos leidos, en bytes. Añadido en MySQL 5.0.2.

- `Innodb_data_reads`

  El número total de lecturas de datos. Añadido en MySQL 5.0.2.

- `Innodb_data_writes`

  El número total de escrituras de datos. Añadido en MySQL 5.0.2.

- `Innodb_data_written`

  El total de datos escritos, en bytes. Añadido en MySQL 5.0.2.

- `Innodb_dblwr_writes` , `Innodb_dblwr_pages_written`

El número de operaciones doublewrite que se han ejecutado y el número de páginas que se han escrito para ello. Añadido en MySQL 5.0.2. Consulte Sección 15.14.1, “E/S de disco (Disk I/O)”.

- `Innodb_log_waits`

  El número de esperas debidas a que el registro del buffer era demasiado pequeño y se tuvo que esperar a que fuese volcado antes de continuar. Añadido en MySQL 5.0.2.

- `Innodb_log_write_requests`

  El número de peticiones de escritura al registro. Añadido en MySQL 5.0.2.

- `Innodb_log_writes`

  El número de escrituras físicas al archivo de registro. Añadido en MySQL 5.0.2.

- `Innodb_os_log_fsyncs`

  El número de escrituras `fsync()` realizadas al archivo de registro. Añadido en MySQL 5.0.2.

- `Innodb_os_log_pending_fsyncs`

  El número de operaciones `fsync()` del archivo de registro. Añadido en MySQL 5.0.2.

- `Innodb_os_log_pending_writes`

  Escrituras del archivo de registro pendientes. Añadido en MySQL 5.0.2.

- `Innodb_os_log_written`

  El número de bytes escritos al archivo de registro. Añadido en MySQL 5.0.2.

- `Innodb_page_size`

  El tamaño de página con que se compiló `InnoDB` (16KB por defecto). Muchos valores son contados en páginas; el tamaño de página permite convertirlos fácilmente a bytes. Añadido en MySQL 5.0.2.

- `Innodb_pages_created`

  El número de páginas creadas. Añadido en MySQL 5.0.2.

- `Innodb_pages_read`

  El número de páginas leidas. Añadido en MySQL 5.0.2.

- `Innodb_pages_written`

  El número de páginas escritas. Añadido en MySQL 5.0.2.

- `Innodb_row_lock_current_waits`

  El número de bloqueos de registro por el que se está esperando. Añadido en MySQL 5.0.3.

- `Innodb_row_lock_time`

  El tiempo total gastado en adquirir bloqueos de registro, en milisegundos. Añadido en MySQL 5.0.3.

- `Innodb_row_lock_time_avg`

  El tiempo medio gastado en adquirir un bloqueo de registro, en milisegundos. Añadido en MySQL 5.0.3.

- `Innodb_row_lock_time_max`

  El tiempo máximo gastado en adquirir un bloqueo de registro, en milisegundos. Añadido en MySQL 5.0.3.

- `Innodb_row_lock_waits`

  El número de veces que se ha tenido que esperar por un bloqueo de registro. Añadido en MySQL 5.0.3.

- `Innodb_rows_deleted`

  El número de registros borrados de tablas `InnoDB`. Añadido en MySQL 5.0.2.

- `Innodb_rows_inserted`

  El número de registros insertados en tablas `InnoDB`. Añadido en MySQL 5.0.2.

- `Innodb_rows_read`

  El número de registros leidos desde tablas `InnoDB`. Añadido en MySQL 5.0.2.

- `Innodb_rows_updated`

  El número de registros actualizados en tablas `InnoDB`. Añadido en MySQL 5.0.2.

- `Key_blocks_not_flushed`

  El número de bloques de claves en la cache de claves que han cambiado pero todavía no han sido volcados a disco.

- `Key_blocks_unused`

  El número de bloques sin utilizar en la cache de claves. Puede utilizar este valor para determinar qué tamaño de la cache de claves está en uso; consulte la discusión de `key_buffer_size` en Sección 5.3.3, “Variables de sistema del servidor”.

- `Key_blocks_used`

  El número de bloques utilizados en la cache de claves. Este valor es una marca de máximo, que indica el número máximo de bloques que han sido nunca utilizados al mismo tiempo.

- `Key_read_requests`

  El número de peticiones para leer un bloque de claves de la cache.

- `Key_reads`

  El número de lecturas físicas de un bloque de claves desde disco. Si `Key_reads` es grande, entonces el valor de `key_buffer_size` es, probablemente, demasiado pequeño. La tasa de fallos de la cache puede ser calculada como `Key_reads`/`Key_read_requests`.

- `Key_write_requests`

  El número de peticiones de escritura de un bloque de claves a la cache.

- `Key_writes`

  El número de escrituras físicas de un bloque de claves a disco.

- `Last_query_cost`

El coste total de la última consulta compilada tal como ha sido computada por el optimizador de consultas. Es útil para comparar el coste de diferentes planes de ejecución para la misma consulta. El valor por defecto de 0 significa que no se ha compilado ninguna consulta todavía. Esta variable fue añadida en MySQL 5.0.1 con un valor por defecto de -1. En MySQL 5.0.7 el valor por defecto ha cambiado a 0; también en la versión 5.0.7, el ámbito de `Last_query_cost` ha cambiado a sesión en vez de global.

- `Max_used_connections`

  El número máximo de conexiones que han sido utilizadas simultáneamente desde que el servidor ha sido iniciado.

- `Not_flushed_delayed_rows`

  El número de registros esperando a ser escritos en colas de `INSERT DELAY`.

- `Open_files`

  El número de archivos que están abiertos.

- `Open_streams`

  El número de flujos de datos (strems) que están abiertos (utilizados principalmente para el registro).

- `Open_tables`

  El número de tablas que están actualmente abiertas.

- `Opened_tables`

  El número de tablas que han sido abiertas. Si `Opened_tables` es grande, probablemente el valor de `table_cache` es demasiado pequeño.

- `Qcache_free_blocks`

  El número de bloques de memoria libres en la cache de consultas.

- `Qcache_free_memory`

  El total de memoria libre en la cache de consultas.

- `Qcache_hits`

  El número de éxitos de la cache.

- `Qcache_inserts`

  El número de consultas añadidas a la cache.

- `Qcache_lowmem_prunes`

  El número de consultas que fueron borradas de la cache de consultas debido a baja memoria.

- `Qcache_not_cached`

  El número de consultas que no han entrado en la cache (por no ser "cacheables", o debido al parámetro `query_cache_type`).

- `Qcache_queries_in_cache`

El número de consultas registradas en la cache.

- `Qcache_total_blocks`

  El número total de bloques en la cache de consultas.

- `Questions`

  El número de consultas que han sido enviadas al servidor.

- `Rpl_status`

  El estado de la replicación a prueba de fallos (todavía no implementado).

- `Select_full_join`

  El número de joins que no utilizan índices. Si este valor no es 0, debería comprobar cuidadosamente los índices de sus tablas.

- `Select_full_range_join`

  El número de joins que han utilizado una búsqueda por rango en una tabla de referencia.

- `Select_range`

  El número de joins que han usado rangos en la primera table. Normalmente no es algo crítico aunque pueda ser bastante grande.

- `Select_range_check`

  El número de joins sin claves que comprueban la utilización de claves después de cada registro. Si esto no es cero, debería comprobar cuidadosamente los índices de sus tablas.

- `Select_scan`

  El número de joins que han hecho un escaneo total de la primera tabla.

- `Slave_open_temp_tables`

  El número de tablas temporales abiertas actualmente por el subproceso SQL esclavo.

- `Slave_running`

  Este valor es `ON` si el servidor es un esclavo que está conectado a un servidor maestro.

- `Slave_retried_transactions`

  Número total de veces desde el inicio que el subproceso SQL esclavo de replicación ha intentando alguna transacción. Para la serie de MySQL 5.0, esta variable fue añadida en la versión 5.0.4.

- `Slow_launch_threads`

  El número de subprocesos que han tardado en crearse más de `slow_launch_time` segundos.

- `Slow_queries`

  El número de consultas que han tardado más de `long_query_time` segundos. Consulte Sección 5.10.4, “El registro de consultas lentas (Slow Query Log)”.

- `Sort_merge_passes`

  El número de pasadas que el algoritmo de ordenación ha tenido que hacer. Si este valor es grande, debería considerar incrementar el valor de la variable de sistema`sort_buffer_size`

- `Sort_range`

  El número de ordenaciones que fueron realizadas utilizando rangos.

- `Sort_rows`

  El número de registros ordenados.

- `Sort_scan`

  El número de ordenaciones que fueron hechas escaneando la tabla.

- `Ssl_`*`xxx`*

  Variables utilizadas para conexiones SSL.

- `Table_locks_immediate`

  El número de veces que un bloque de tabla ha sido adquirido inmediatamente.

- `Table_locks_waited`

  El número de veces que un bloque de tabla no se ha podido adquirir inmediatamente y se ha necesitado una espera. Si esto es alto, y tiene problemas de rendimiento, debería primero optimizar sus consultas y después, o bien utilizar replicación, o dividir sus tablas.

- `Threads_cached`

  El número de subprocesos en la cache de subprocesos.

- `Threads_connected`

  El número de conexiones abiertas actualmente.

- `Threads_created`

  El número de subprocesos creados para gestionar conexiones. Si `Threads_created` es grande, debería incrementar el valor de `thread_cache_size`. La tasa de éxitos de la cache puede ser calculada como `Threads_created`/`Connections`.

- `Threads_running`

  El número de subprocesos que no están durmiendo.

- `Uptime`

  El número de segundos que el servidor ha estado funcionando ininterrumpidamente.

5.4. El proceso de cierre del servidor MySQL

Los pasos del proceso de apagado del servidor son:

1. Comienza el proceso de apagado

2. El servidor crea un subproceso de apagado si es necesario
3. El servidor deja de aceptar nuevas conexiones
4. El servidor acaba con su tarea actual
5. Se apagan o cierran los motores de almacenamiento
6. El servidor se cierra

Seguidamente, una descripción más detallada del proceso:

1. Comienza el proceso de apagado.

   El apagado del servidor puede iniciarse de diferentes maneras. Por ejemplo, un usuario con el privilegio `SHUTDOWN` puede ejecutar la orden `mysqladmin shutdown`. `mysqladmin` puede ser utilizado en cualquier plataforma soportada por MySQL. También son posibles otros métodos de encendido y apagado específicos de cada sistema operativo: El servidor se apaga en Unix cuando recibe una señal `SIGTERM`. Un servidor ejecutándose como servicio en Windows se apaga cuando el administrador de servicios se lo indica.

2. El servidor crea un subproceso de apagado si es necesario.

   Dependiendo de como se ha iniciado el apagado, el servidor puede crear un subproceso para llevar a cabo el proceso de apagado. Si el apagado fue demandado por un cliente, se crea un subproceso de apagado. Si el apagado es debido a la recepción de una señal `SIGTERM`, el subproceso de la señal podría llevar a cabo el apagado él mismo, o podría crear un subproceso separado para hacerlo. Si el servidor intenta crear un subproceso de apagado y no puede hacerlo (por ejemplo, porque no hay memoria suficiente disponible), crea un mensaje de diagnóstico que aparece en el registro de errores:

   ```
   Error: Can't create thread to kill server
   ```

3. El servidor deja de aceptar nuevas conexiones.

   Para prevenir que comiencen nuevas actividades durante el apagado, el servidor deja de aceptar nuevas conexiones de clientes. Esto lo consigue cerrando las conexiones de red a las que normalmente escucha: el puerto TCP/IP, el archivo socket Unix, la "named pipe" de Windows, y la memoria compartida de Windows.

4. El servidor acaba con su tarea actual.

   En cada subproceso asociado a una conexión de un cliente, se rompe la conexión al cliente, y dicho subproceso es marcado como muerto. Los subprocesos mueren cuando se dan cuenta de que han sido marcados de esa manera. Los subprocesos de conexiones inactivas mueren rápidamente. Los que están actualmente procesando sentencias, consultan periodicamente su estado y tardan un poco más en morir. Para encontrar información adicional sobre la terminación de subprocesos, consulte Sección 13.5.5.3, “Sintaxis de `KILL`”, en particular las instrucciones sobre las operaciones `REPAIR TABLE` o `OPTIMIZE TABLE` que han sido matadas en tablas `MyISAM`.

   En los subprocesos que tienen abierta una transacción, la transacción se cancela. Nótese que si un subproceso está actualizando una tabla no transaccional, una operación como un `UPDATE` o `INSERT` de múltiples registros podría dejar la tabla parcialmente actualizada, porque la operación puede terminar antes de ser completada.

   Si el servidor es un servidor maestro de replicación, los subprocesos asociados a esclavos todavía conectados son tratados como subprocesos de cualquier otro cliente. Es decir, cada uno es marcado como muerto y se cierra la próxima vez que éste comprueba su estado.

Si el servidor es un servidor esclavo de replicación, los subprocesos de Entrada/Salida y SQL se paran, si están activos, antes de que los subprocesos de cliente sean marcados como muertos. Se permite al subproceso SQL finalizar su sentencia actual (para evitar causar problemas de replicación), y entonces se para. Si el subproceso SQL estaba en medio de una transacción en ese momento, la transacción se cancela.

5. Se apagan o cierran los motores de almacenamiento.

   En este punto, la cache de tabla se escribe a disco y se cierran todas las tablas abiertas.

   Cada motor de almacenamiento realiza todas las acciones necesarias para las tablas que controla. Por ejemplo, MyISAM realiza cualquier escritura de índices pendiente de una tabla. InnoDB escribe sus buffers a disco (desde la versión 5.0.5: a menos que `innodb_fast_shutdown` valga 2), escribe el número de secuencia de registro (LSN - Log Sequence Number) en el espacio de tablas, y cierra sus propios subprocesos internos.

6. El servidor se cierra.

5.5. Cuestiones de seguridad general

Esta sección describe algunos temas generales de seguridad que hay que tener en cuenta, y qué se puede hacer para aumentar la seguridad de la instalación MySQL contra ataques o errores de uso. Para encontrar información específica sobre el sistema de control de accesos que MySQL utiliza para crear cuentas de usuarios y comprobar el acceso a las bases de datos, consulte Sección 5.6, “El sistema de privilegios de acceso de MySQL”.

5.5.1. Guía de seguridad general

Cualquiera que utilice MySQL en un ordenador conectado a Internet debería leer esta sección para evitar los errores de seguridad más comunes.

Al tratar el tema de la seguridad, hacemos hincapié en la necesidad de proteger totalmente la máquina completa (no únicamente el servidor MySQL) contra todos los tipos de ataques posibles; intercepción pasiva de paquetes, alteración, reproducción de comandos (playback), y denegación de servicio. Aquí no tratamos todos los aspectos de disponibilidad y tolerancia a fallos.

Para todas las conexiones, consultas, y otras operaciones que los usuarios pueden intentar realizar, MySQL utiliza seguridad basada en Listas de Control de Acceso (ACLs). También hay algún soporte para conexiones cifradas mediante SSL entre clientes y servidores MySQL. Muchos de los conceptos que aquí se exponen no son específicos de MySQL; las mismas ideas generales se pueden aplicar a cualquier aplicación.

Al ejecutar MySQL, siga siempre que sea posible estas recomendaciones:

- **¡No de nunca a nadie (excepto a la cuenta `root` de MySQL acceso a la tabla `user` en la base de datos `mysql`!** Esto es crítico. **La clave cifrada es la verdadera clave en MySQL.** Cualquiera que sepa cual es la clave que hay en la tabla `user` y tenga acceso a la máquina host de la cuenta registrada **puede acceder fácilmente como ese usuario**.

- Estudie el sistema de privilegios de acceso de MySQL. Las sentencias `GRANT` y `REVOKE` se utilizan para controlar el acceso a MySQL. No otorgue más privilegios de los necesarios. Nunca otorgue privilegios a un mismo usuario sin tener en cuenta el equipo desde el que se conecta.

  Lista de comprobaciones:

  - Pruebe el comando `mysql -u root`. Si es capaz de conectar al servidor sin la necesidad de introducir una clave, tiene problemas. ¡Cualquiera puede conectar a su servidor MySQL como el usuario `root` de MySQL con privilegios totales! Revise las instrucciones de instalación de MySQL, prestando atención en concreto a la información sobre establecer una clave para el usuario `root`. Consulte Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”.

  - Utilice la sentencia `SHOW GRANTS` y compruebe quién tiene acceso a qué. Después utilice la sentencia `REVOKE` para denegar los privilegios que no son necesarios.

- No almacene ninguna clave sin cifrar en su base de datos. Si alguien tuviera acceso a su ordenador, el intruso podría obtener la lista completa de claves y utilizarlas. En vez de eso, utilice `MD5()`, `SHA1()`, o cualquier otra función de hashing de un sentido.

- No elija claves que puedan aparecer en un diccionario. Existen programas especiales para romperlas. Incluso claves como ``xperro98'' son muy malas. Es mucho mejor ``oweei98'', que contiene la misma palabra ``perro'' pero escrita desplazándose una tecla a la izquierda en un teclado QWERTY convencional. Otro método es usar ``Mtupc'', que ha sido tomada de las primeras letras de cada palabra de la frase ``María tuvo un pequeño corderito.'' Así es fácil de recordar y escribir, pero difícil de adivinar para cualquiera que no la conozca.

- Invierta en un firewall. Le protegerá de al menos el 50% de todos los tipos de vulnerabilidades de cualquier software. Ponga MySQL tras el firewall o en una zona desmilitarizada (DMZ).

  Lista de comprobaciones:

  - Intente escanear sus puertos desde Internet utilizando una herramienta como `nmap`. MySQL utiliza el puerto 3306 por defecto. Este puerto no debería ser accesible desde lugares no confiables. Otra manera simple de probar si el puerte MySQL está abierto o no es intentar el siguiente comando desde alguna máquina remota, donde `server_host` es la máquina en la que su servidor MySQL se está ejecutando:

    ```
    shell> telnet server_host 3306
    ```

    Si consigue conectar y algunos caracteres extraños, el puerto está abierto, y debería cerrarlo en su firewall o router, a menos que tenga una buena razón para mantenerlo abierto. Si el comando `telnet` no consigue conectar o la conexión es rechazada, entonces el puerto se encuentra bloqueado, que es como queremos que esté.

- No confíe en ningún dato enviado por los usuarios de sus aplicaciones. Pueden intentar engañar a su código introduciendo secuencias de caracteres especiales en formularios webs, URLs, o cualquier aplicación que haya desarrollado. Asegúrese de que su aplicación permance segura si un usuario introduce algo como ``; DROP DATABASE mysql;''. Este es un ejemplo algo extremo, pero los mayores agujeros de seguridad y pérdidas de datos pueden ocurrir como resultado de hackers utilizando técnicas similares, si no se está preparado para ellas.

  Un error común es proteger únicamente valores de tipo cadena de caracteres. Recuerde comprobar los datos numéricos también. Si una aplicación genera una consulta como `SELECT * FROM table WHERE ID=234` cuando un usuario introduce el valor `234`, el usuario podría introducir el valor `234 OR 1=1` para provocar que la aplicación genere la consulta `SELECT * FROM table WHERE ID=234 OR 1=1`. Como resultado, el servidor extraerá todos los registros en la tabla. Esto, además de exponer cada registro, causa una carga excesiva en el servidor. La manera más simple de protegerse frente a este tipo de ataque es utilizar comillas simples alrededor de las constantes numéricas: `SELECT * FROM table WHERE ID='234'`. Si el usuario entrase información extra, todo sería parte de la cadena de

caracteres. En un contexto numérico, MySQL automáticamente convierte esta cadena en un número, y elimina cualquier carácter no númerico del final que la cadena pueda contener.

A veces la gente piensa que si una base de datos contiene sólo datos de dominio público, no tiene por qué ser protegida. Esto es incorrecto. Aunque sea admitible mostrar cualquier registro de la base de datos, siempre se debería proteger contra ataques de tipo denegación de servicio (por ejemplo, aquellos que se basan en la técnica del párrafo precedente, que causan que el servidor malgaste recursos). Si no, el servidor podría quedar inservible para sus usuarios legítimos.

Lista de comprobaciones:

- Intente introducir comillas simples y dobles (`'` y `"`) en todos sus formularios web. Si obtiene cualquier clase de error MySQL, investigue el problema sin demora.
- Intente modificar las URLs dinámicas añadiendo las cadenas `%22` (`"`), `%23` (`#`), y `%27` (`'`).
- Intente modificar los tipos de datos en las URLs dinámicas de tipos numéricos a alfanuméricos, usando los caracteres mostrados en los ejemplos previos. Su aplicaicón debería ser segura contra estos y otros ataques similares.
- Intente introducir letras, espacios, y símbolos especiales en vez de números en los campos numeicos. Su aplicación debería eliminarlos antes de pasarlos a MySQL, o en todo caso generar un error. ¡Pasar valores sin comprobar a MySQL es muy peligroso!
- Compruebe el tamaño de los datos antes de pasárselos a MySQL.
- Haga que su aplicación se conecte a la base de datos utilizando un nombre de usuario diferente del que utiliza para tareas administrativas. No dé a sus aplicaciones ningún acceso que no necesiten.

- Muchas interfaces de programación de aplicaciones proveen alguna manera de preceder con caracteres de escape los caracteres especiales en sus datos. Usados adecuadamente, esto previene que los usuarios de las aplicaciones introduzcan valores que provoquen que la aplicación genere sentencias con efectos diferentes a los que usted pretendía:
  - API MySQL de C: Utilice la función `mysql_real_escape_string()`.
  - MySQL++: Utilice los modificadores `escape` y `quote` para streams
  - PHP: Utilice la función `mysql_escape_string()`, que está basada en la función del mismo nombre de la API MySQL de C. (Con versiones anteriores a PHP 4.0.3, utilice `addslashes()` en cambio.) En PHP 5, puede utilizar la extensión `mysqli`, que soporta los protocolo de autentificación y clave de acceso mejorados de MySQL, así como las sentencias preparadas con placeholders.
  - DBI de Perl: Utilice el método `quote()` o utilice placeholders.
  - JDBC de Java: Utilice un objeto `PreparedStatement` y placeholders.

  Otras interfaces de programación deberían tener capacidades similares.
- No transmita datos sin cifrar por Internet. Esta información es accesible para cualquiera que tenga el tiempo y la habilidad para interceptarla y utilizarla para sus propios propósitos. En vez de eso, utilice un protocolo de cifrado como SSL o SSH. MySQL soporta conexiones SSL internas desde la versión 4.0.0. El redireccionamiento de puertos de SSH se puede utilizar para crear un tunel cifrado (y comprimido) para la comunicación.
- Aprenda a utilizar las herramientas `tcpdump` y `strings`. En la mayoría de los casos, usted puede comprobar si los flujos de datos de MySQL están cifrados ejecutando un comando como el siguiente:

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

(Esto funciona en Linux, y debería funcionar, con pequeñas modificaciones en otros sistemas.) Atención: Si no ve los datos en formato de texto, esto no siempre quiere decir que la información esté realmente cifrada. Si necesita un alto nivel de seguridad, debería consultar a un experto en la materia.

5.5.2. Hacer que MySQL sea seguro contra ataques

Cuando se conecta a un servidor MySQL, debería utilizar una clave. La clave no se transmite en texto llano a través de la conexión. El tratamiento de las claves durante la conexión de un cliente ha sido mejorado en MySQL 4.1.1 para ser muy seguro. Si todavía está utilizando claves del tipo anterior a 4.1.1, el algoritmo de cifrado no es tan potente como el nuevo algoritmo; con un poco de esfuerzo un atacante inteligente que pueda interceptar el tráfico entre el cliente y el servidor podría romper la clave. (Consulte Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1” para una explicación sobre los diferentes métodos de tratamiento de claves.) Si la conexión entre el cliente y el servidor pasa a través de una red no segura, debería utilizar un tunel SSH para cifrar la comunicación.

Toda la demás información se transmite como texto, y puede ser leida por cualquiera que pueda observar la conexión. Si esto le preocupa, utilice el protocolo comprimido para hacer que el tráfico sea mucho más difícil de descifrar. Para hacer la conexión aún más segura, debería utilizar SSH para conseguir una conexión TCP/IP cifrada entre el servidor MySQL y el cliente MySQL. Puede encontrar un cliente SSH Open Source en http://www.openssh.org/, y un cliente comercial en http://www.ssh.com/.

En MySQL 5.0, puede utilizar también el soporte interno de OpenSSL. Consulte Sección 5.7.7, “Usar conexiones seguras”.

Para convertir un sistema MySQL en seguro, debería considerar seriamente las siguientes sugerencias:

- Utilice claves para todos los usuarios MySQL. Un programa cliente no conoce necesariamente la identidad de la persona utilizándolo. Es común en las aplicaciones cliente/servidor que el usuario pueda especificar cualquier nombre de usuario al programa cliente. Por ejemplo, cualquiera puede utilizar el programa `mysql` para conectarse como cualquier otra persona, simplemente invocándolo de la siguiente manera: `mysql -u otro_usuario nombre_bd` cuando `otro_usuario` no tiene clave. Si todos los usuarios tienen una clave, conectarse utilizando la cuenta de otro usuario se vuelve mucho más difícil.

  Para cambiar la clave de un usuario, utilice la sentencia `SET PASSWORD`. También es posible alterar la tabla `user` en la base de datos `mysql` directamente. Por ejemplo, para cambiar la clave de todas las cuentas MySQL que tienen por nombre de usuario `root`, haga lo siguiente:

  ```
  shell> mysql -u root
  mysql> UPDATE mysql.user SET
  Password=PASSWORD('newpwd')
      -> WHERE User='root';
  mysql> FLUSH PRIVILEGES;
  ```

- Nunca ejecute el servidor MySQL con el usuario `root` de Unix. Esto es extremadamente peligroso porque cualquier usuario con el privilegio `FILE` es capaz de crar ficheros como `root` (por ejemplo, `~root/.bashrc`). Para prevenir esto, `mysqld` rechaza ejecutarse como `root` a menos que se utilice explícitamente la opción `--user=root`.

  En vez de eso, `mysqld` puede (y debe) ser ejecutado mediante un usuario normal sin privilegios. Puede crear una cuenta de Unix específica llamada `mysql` para hacelo todo aún más seguro. Utilice esta cuenta tan solo para administrar MySQL. Para ejecutar `mysqld` mediante un usuario de Unix diferente,

añada la opción `user` que especifica el nombre de usuario al grupo `[mysqld]` del fichero de opciones `/etc/my.cnf` o al fichero de opciones `my.cnf` en el directorio de datos del servidor. Por ejemplo:

```
[mysqld]
user=mysql
```

Esto provoca que el servidor se inicie mediante el usuario designado, lo ejecute usted manualmente o mediante `mysqld_safe` o `mysql.server`. Para más detalles, consulte Sección A.3.2, “Cómo ejecutar MySQL como usuario normal”.

Ejecutar `mysqld` como un usuario Unix diferente de `root` no significa que necesite cambiar el usuario `root` de la tabla `user`. Los usuarios de las cuentas MySQL no tienen nada que ver con los usuarios de las cuentas Unix.

- No permita el uso de enlaces simbólicos a tablas. (Esto puede desactivarse con la opción `--skip-symbolic-links`.) Esto es especialmente importante si ejecuta `mysqld` como `root`, porque cualquiera que tenga acceso de escritura al directorio de datos del servidor ¡podría entonces borrar cualquier fichero en el sistema!. Consulte Sección 7.6.1.2, “Utilización de enlaces simbólicos para tablas en Unix”.

- Asegúrese de que el único usuario Unix con permisos de lectura o escritura en los directorios de la base de datos es el usuario que ejecuta `mysqld`.

- No otorgue los privilegios `PROCESS` o `SUPER` a usuarios no-administrativos. La salida del de `mysqladmin processlist` muestra el texto de cualquier sentencia que se esté ejecutando, así que cualquier usuario al que se permita ejecutar ese comando puede ser capaz de ver si otro usuario ejecuta una sentencia `UPDATE user SET password=PASSWORD('not_secure')`.

  `mysqld` reserva una conexión extra para usuarios que tengan el privlegio `SUPER`, así que un usuario `root` puede conectarse y comprobar la actividad del servidor aún cuando todas las conexiones normales estén en uso.

  El privilegio `SUPER` puede utilizarse para cerrar conexiones de cliente, cambiar el funcionamiento del servidor modificando el valor de variables del sistema, y controlar servidores de replicación.

- No otorgue el privilegio `FILE` a usuarios no-administrativos. Cualquier usuario que posea este privilegio puede escribir un archivo en cualquier de lugar del sistema de ficheros con los privilegios del demonio `mysqld`. Para hacer esto un poco más seguro, los archivos generados con `SELECT ... INTO OUTFILE` no sobreescriben archivos existentes, y pueden ser escritos por cualquiera.

  El privilegio `FILE` puede también ser utilizado para leer cualquier archivos que sea legible por cualquiera o accesible para el usuario Unix que ejecuta el servidor. Con este privilegio, podría por ejemplo leer cualquier fichero e insertarlo en una tabla de la base de datos. Esto podría utilizarse, por ejemplo, utilizando `LOAD DATA` para cargar `/etc/passwd` en una tabla, que podría ser mostrada después con un `SELECT`.

- Si no confía en sus DNS, podría utilizar números IP en vez de nombres en las tablas de permisos (tablas grant). En cualquier caso, debería ser muy cuidadoso en crear registros en las tablas de permiso utilizando nombres que contengan caracteres comodín.

- Si quiere restringir el número de conexiones permitidas para una misma cuenta, puede hacerlo estableciando la variable `max_user_connections` de `mysqld`. La sentencia `GRANT` también soporta opciones de control de recursos para limitar la extensión de uso de servidor permitido a una cuenta. Consulte Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”.

5.5.3. Opciones de arranque para mysqld relacionadas con la seguridad

Las siguientes opciones de mysqld afectan a la seguridad:

- --allow-suspicious-udfs

  Esta opción cotrola si las funciones definidas por el usuario que sólo tienen un símbolo xxx para la función principal se pueden cargar. Por defecto, la opción está desactivada y sólo UDFs que tengan al menos un símbolo auxiliar pueden cargarse. Esto previene intentos de cargar funciones desde ficheros objeto compartidos que no contegan UDFs legítimos. Para MySQL 5.0, esta opción se añadió en MySQL 5.0.3. Consulte Sección 27.2.3.6, “Precauciones de seguridad en funciones definidas por usuarios”.

- --local-infile[={0|1}]

  Si arranca el servidor con --local-infile=0, los clientes no pueden usar LOCAL en comandos LOAD DATA . Consulte Sección 5.5.4, “Cuestiones relacionadas con la seguridad y LOAD DATA LOCAL”.

- --old-passwords

  Fuerza al servidor a generar hashes de contraseñas cortos (pre-4.1) para las nuevas contraseñas. Esto es útil para compatibilidad cuando el servidor debe soportar antiguos programas cliente. Consulte Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1”.

- (*OBSOLETO*) --safe-show-database

  En versiones previas de MySQL, esta opción provoca que el comando SHOW DATABASES muestre los nombers de sólo aquellas bases de datos para las que el usuario tiene algún tipo de privilegio. En MySQL 5.0, esta opción no está disponible ya que es el comportamiento por defecto, y hay un privilegio SHOW DATABASES que puede usarse para controlar el acceso a los nombres de las bases de datos para cada cuenta. Consulte Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”.

- --safe-user-create

  Si está activada, un usuario no puede crear nuevos usuarios con el comando GRANT a no ser que el usuario tenga el privilegio INSERT para la tabla mysql.user . Si desea que un usuario tenga la habilidad de crear nuevos usuarios con los privilegios que el usuario tiene derecho a otorgar, debe otorgar al usuario el siguiente privilegio:

  ```
  mysql> GRANT INSERT(user) ON mysql.user TO 'nombre_usuario'@'nombre_host';
  ```

  Esto asegura que el usuario no pueda cambiar ninguna columna de privilegios directamente, pero debe usar el comando GRANT para dar privilegios a otros usuarios.

- --secure-auth

  Desactiva autenticación para cuentas que usen antiguas contraseñas (pre-4.1)

- --skip-grant-tables

  Esta opción hace que el servidor no use el sistema de privilegios en absoluto. Esto da a todo el mundo *acceso total* a todas las bases de datos! (Puede decirle a un servidor en ejecución que arranque usando las tablas de permisos de nuevo usando mysqladmin flush-privileges o el comando mysqladmin reload, o mediante el comando FLUSH PRIVILEGES.)

- --skip-name-resolve

Los nombres de equipo no se resuelven. Todo valor en la columna `Host` en la tabla de permisos deben ser números IP o `localhost`.

- `--skip-networking`

  No permite conexiones TCP/IP a través de la red. Todas las conexiones a `mysqld` se realizan mediante ficheros socket de Unix.

- `--skip-show-database`

  Con esta opción, el comando `SHOW DATABASES` se permite sólo a usuarios que tengan el privilegio `SHOW DATABASES` , y el comando muestra todos los nombres de bases de datos. Sin esta opción , `SHOW DATABASES` está permitido a todos los usuarios, pero muestra cada nombre de base de datos sólo si el usuario tiene el privilegio `SHOW DATABASES` o algún privilegio para la base de datos.

5.5.4. Cuestiones relacionadas con la seguridad y LOAD DATA LOCAL

El comando `LOAD DATA` puede cargar un fichero que esté localizado en el equipo servidor, o puede cargar un fichero localizado en el equipo cliente cuando se especifica la palabra clave `LOCAL`.

Hay dos aspectos de seguridad potenciales al soportar la versión `LOCAL` de los comandos `LOAD DATA`:

- La transferencia del fichero desde el equipo cliente al equipo servidor se inicia mediante el servidor MySQL. En teoría, puede construirse un servidor modificado de forma que le diga al programa cliente que transfiera un fichero elegido por el servidor en lugar de el fichero especificado por el cliente en el comando `LOAD DATA` . Tal servidor podría acceder a cualquier fichero en el equipo cliente al que el usuario cliente tuviese acceso de lectura.

- En un entorno Web en el que los clientes se conecten mediante un servidor Web, un usuario podría usar `LOAD DATA LOCAL` para leer cualquier fichero al que el servidor Web tuviese acceso de lectura (asumiendo que el usuario puediese ejecutar cualquier comando contra el servidor SQL). En este entorno, el cliente respecto al servidor MySQL es el servidor Web, no el programa ejectuado por el usuario para conectar al servidor Web.

Para tratar estos problemas, hemos cambiado el funcionamiento de `LOAD DATA LOCAL` en MySQL 3.23.49 y MySQL 4.0.2 (4.0.13 en Windows):

- Por defecto, todos los clientes MySQL y bibliotecas en distribuciones binarias se compilan con la opción `--enable-local-infile` para ser compatible con la versión MySQL 3.23.48 y anteriores.

- Si compila MySQL de los ficheros fuentes pero no usa la opción `--enable-local-infile` para `configure`, `LOAD DATA LOCAL` no puede usarse por ningún cliente a no ser que se escriba explícitamente para invocar `mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0)`. Consulte Sección 24.2.3.44, “`mysql_options()`”.

- Puede desactivar todos los comandos `LOAD DATA LOCAL` desde el lado del servidor arrancando `mysqld` con la opción `--local-infile=0`.

- Para el cliente de línea de comando `mysql`, `LOAD DATA LOCAL` puede activarse especificando la opción `--local-infile[=1]` , o deshabilitarse con la opción `--local-infile=0` . De forma similar, para `mysqlimport`, las opciones `--local` o `-L` permite la carga de datos locales. En cualquier caso, el uso exitoso de una operación de carga local requiere que el servidor lo permita.

- Si usa `LOAD DATA LOCAL` en scripts de Perl scripts u otros programsa que lean del grupo `[client]` en los ficheros de opciones, puede añadir la opción `local-infile=1` a ese grupo. De todos modos,

para evitar que esto cause problemas en programas que no entiendan `local-infile`, especifíquelo usando el prefijo `loose-` :

```
[client]
loose-local-infile=1
```

- Si `LOAD DATA LOCAL INFILE` está desactivado, tanto en el servidor o el cliente, un cliente que trate de ejecutar dicho comando recibe el siguiente mensaje de error:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

5.6. El sistema de privilegios de acceso de MySQL

MySQL tiene un sistema avanzado pero no de seguridad y privilegios. Esta sección describe su funcionamiento.

5.6.1. Qué hace el sistema de privilegios

La función primaria del sistema de privilegios de MySQL es autenticar un usuario conectándose desde un equipo dado, y asociar dicho usuario con privilegios en una base de datos tales como `SELECT`, `INSERT`, `UPDATE`, y `DELETE`.

Funcionalidad adicional incluye la habilidad de tener usuarios anónimos y de dar privilegios para funciones específicas de MySQL tales como `LOAD DATA INFILE` y operaciones administrativas.

5.6.2. Cómo funciona el sistema de privilegios

El sistema de privilegios de MySQL asegura que todos los usuarios pueden ejecutar sólo la operación permitida a los mismos. Como usuario, cuando conecta a un servidor MySQL, su identidad se determina mediante *el equipo desde el que se conecta* y *el nombre de usuario que especifique*. Cuando efectúe peticiones tras conectar, el sistema le otorga privilegios acorde a su identidad y *lo que quiera hacer*.

MySQL considera tanto su nombre de usuario y su equipo a la hora de identificarle, ya que no hay razón para asumir que un nombre de usuario pertenece a la misma persona en cualquier sitio de Internet. Por ejemplo, el usuario `joe` que conecta desde `office.com` no tiene porqué ser la misma persona que el usuario `joe` que conecta desde `elsewhere.com`. MySQL trata esto permitiendole distinguir usuarios en diferentes equipos que tienen el mismo nombre. Puede otorgar un conjunto de privilegios para conexiones de `joe` desde `office.com`, y un conjunto distinto para conexiones de `joe` desde `elsewhere.com`.

El control de acceso de MySQL implica dos etapas:

- Etapa 1: El servidor comprueba si debe permitirle conectarse.
- Etapa 2: Asumiendo que se conecta, el servidor comprueba cada comando que ejecuta para ver si tiene suficientes permisos para hacerlo. Por ejemplo, si intenta seleccionar registros de una tabla en una base de datos o eliminar una tabla de la base de datos, el servidor verifica que tenga el permiso `SELECT` para la tabla o el permiso `DROP` para la base de datos.

Si sus permisos cambian (por usted mismo o alguien distinto) mientras está conectado, estos cambios no tienen porqué tener efecto inmediatamente para el siguiente comando que ejecute. Consulte Sección 5.6.7, “Cuándo tienen efecto los camios de privilegios” para más detalles.

El servidor guarda información de privilegios en las tablas de permisos de la base de datos `mysql` (esto es, en la base de datos llamada `mysql`). El servidor MySQL lee el contenido de dichas tablas en memoria

cuando arranca y las vuelve a leer bajo las circunstancias indicadas en Sección 5.6.7, “Cuándo tienen efecto los camios de privilegios”. Las decisiones acerca de control de acceso se basan en las copias en memoria de las tablas de permisos.

Normalmente, manipula los contenidos de las tablas de permisos indirectamente usando los comandos `GRANT` y `REVOKE` para configurar cuentas y controlar los privilegios disponibles para cada una. Consulte Sección 13.5.1.3, “Sintaxis de `GRANT` y `REVOKE`”. La discusión aquí describe la estructura subyacente de las tablas de permisos y cómo el servidor usa sus contenidos cuando interactúa con clientes.

El servidor usa las tablas `user`, `db`, y `host` en la base de datos `mysql` en ambas etapas de control de acceso. Las columnas en estas tablas de permisos se muestran a continuación:

Nombre tabla	**user**	**db**	**host**
Alcance columnas	`Host`	`Host`	`Host`
	`User`	`Db`	`Db`
	`Password`	`User`	
Columnas privilegios	`Select_priv`	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`	`Create_priv`
	`Drop_priv`	`Drop_priv`	`Drop_priv`
	`Grant_priv`	`Grant_priv`	`Grant_priv`
	`Create_view_priv`	`Create_view_priv`	`Create_view_priv`
	`Show_view_priv`	`Show_view_priv`	`Show_view_priv`
	`Create_routine_priv`	`Create_routine_priv`	
	`Alter_routine_priv`	`Alter_routine_priv`	
	`References_priv`	`References_priv`	`References_priv`
	`Reload_priv`		
	`Shutdown_priv`		
	`Process_priv`		
	`File_priv`		
	`Show_db_priv`		
	`Super_priv`		
	`Create_tmp_table_priv`	`Create_tmp_table_priv`	`Create_tmp_table_priv`
	`Lock_tables_priv`	`Lock_tables_priv`	`Lock_tables_priv`
	`Execute_priv`		
	`Repl_slave_priv`		
	`Repl_client_priv`		
Columnas seguridad	`ssl_type`		
	`ssl_cipher`		

	x509_issuer		
	x509_subject		
Columnas recursos control	max_questions		
	max_updates		
	max_connections		
	max_user_connections		

Execute_priv se presentó en MySQL 5.0.0, pero no fue operacional hasta MySQL 5.0.3.

Las columnas Create_view_priv y Show_view_priv se añadieron en MySQL 5.0.1.

Las columnas Create_routine_priv, Alter_routine_priv, y max_user_connections se añadieron en MySQL 5.0.3.

Durante la segunda etapa de control de acceso, el servidor efectúa una verificación de petición para asegurar que cada cliente tiene suficientes privilegios para cada petición que recibe. Adicionalmente las tablas de permisos user, db, y host , el servidor puede consultar las tablas tables_priv y columns_priv para peticiones que impliquen tablas. Las tablas tables_priv y columns_priv proveen de un control de privilegios más fino a nivel de tabla y columna. Tienen las siguientes columnas:

Nombre tabla	**tables_priv**	**columns_priv**
Alcance de columnas	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Columnas privilegios	Table_priv	Column_priv
	Column_priv	
Otras columnas	Timestamp	Timestamp
	Grantor	

Las columnas Timestamp y Grantor actualmente no se usan y no se discuten más en esta sección.

Para verificación de peticiones que impliquen rutinas almacenadas, el servidor puede consultar la tabla procs_priv. Esta tabla tiene las siguientes columnas:

Nombre tabla	**procs_priv**
Alcance de columnas	Host
	Db
	User
	Routine_name
	Routine_type
Columnas privilegios	Proc_priv
Otras columnas	Timestamp

	`Grantor`

La tabla `procs_priv` existe desde MySQL 5.0.3. La columna `Routine_type` se añadió en MySQL 5.0.6. Hay una columna `ENUM` con valores de `'FUNCTION'` o `'PROCEDURE'` para indicar el tipo de rutina a que se refiere el registro. Esta columna permite que los privilegios se otorgen separadamente para una función y para un procedimiento con el mismo nombre.

Las columnas `Timestamp` y `Grantor` no se usan actualmente y no se discuten más aquí.

Cada tabla de permisos contiene columnas de alcance y columnas de privilegios:

- Las columnas de alcance determinan el alcance de cada entrada (registro) en las tablas; esto es, el contexto en que el registro se aplica. Por ejemplo, un registro de la tabla `user` con los valores `Host` y `User` de `'thomas.loc.gov'` y `'bob'` se usarían para autenticar conexiones hechas al servidor desde el equipo `thomas.loc.gov` por un cliente que especifique un nombre de usuario de `bob`. De forma similar, un registro de la tabla `db` con las columnas `Host`, `User`, y `Db` con valores `'thomas.loc.gov'`, `'bob'` y `'reports'` se usaría cuando `bob` conectase desde el equipo `thomas.loc.gov` para acceder a la base de datos `reports` . Las tablas `tables_priv` y `columns_priv` contienen columnas de alcance indicando tablas o combinaciones de tabla/columna para las que cada registro se aplica. La columna de alcance `procs_priv` indica la rutina de almacenamiento que se aplica a cada registro.
- Las columnas de privilegios indican qué privilegios se otorgan a un registro de la tabla; esto es, qué operaciones pueden ejecutarse. El servidor combina la información de diversas tablas de permisos para tener una descripción completa de los permisos de un usuario. Las reglas usadas para ello se describen en Sección 5.6.6, “Control de acceso, nivel 2: comprobación de solicitudes”.

Las columnas de alcance contienen cadenas de caracteres. Se declaran tal y como se muestra a continuación; el valor por defecto es la cadena de caracteres vacía:

Nombre de columna	Tipo
`Host`	`CHAR(60)`
`User`	`CHAR(16)`
`Password`	`CHAR(16)`
`Db`	`CHAR(64)`
`Table_name`	`CHAR(64)`
`Column_name`	`CHAR(64)`
`Routine_name`	`CHAR(64)`

Con propósito de chequeos de acceso, las comparaciones de los valores de `Host` no tienen en cuenta mayúsculas y minúsculas. Los valores de `User`, `Password`, `Db`, y `Table_name` son sensibles a mayúsculas y minúsuclas. Los valores de `Column_name` no son sensibles a mayúsculas y minúsculas.

En las tablas `user`, `db`, y `host` , cada privilegio se lista en una columna separada que se declara como `ENUM('N','Y') DEFAULT 'N'`. En otras palabras, cada privilegio puede estar desactivado o activado, estando desactivados por defecto.

En las tablas `tables_priv`, `columns_priv`, and `procs_priv` , las columnas de privilegios se declaran como columnas de tipo `SET`. Los valores en estas columnas pueden contener cualquier combinación de los privilegios controlados por la tabla:

Nombre de tabla	Nombre de columna	Posible conjunto de elementos
`tables_priv`	`Table_priv`	`'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'`
`tables_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`columns_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`procs_priv`	`Proc_priv`	`'Execute', 'Alter Routine', 'Grant'`

Brevemente, el servidor usa las tablas de permisos como sigue:

- Las columnas de alcance de la tabla `user` determinan si se rechazan o permiten conexiones entrantes. Para conexiones permitidas, cualquier privilegio otorgado en la tabla `user` indica los privilegios globales del usuario (superusuario). Estos privilegios se aplican a *todas* las bases de datos en el servidor.

- Las columnas de alcance de la tabla `db` determinan qué usuarios pueden acceder a qué bases de datos desde qué equipo. La columna de privilegios determina qué operaciones se permiten. Un privilegio otorgado a nivel de base de datos se aplica a la base de datos y a todas sus tablas.

- La tabla `host` se usa en conjunción con la tabla `db` cuando desea que un registro de la tabla `db` se aplique a varios equipos. Por ejemplo, si queiere que un usuario sea capaz de usar una base de datos desde varios equipos en su red, deje el valor `Host` vacío en el registro de usuario de la tabla`db` , luego rellene la tabla `host` con un registro para cada uno de estos equipos. Este mecanismo se describe con mayor detalle en Sección 5.6.6, “Control de acceso, nivel 2: comprobación de solicitudes”.

  Nota: La tabla `host` no se ve afectada por los comandos `GRANT` ni `REVOKE` . La mayoría de instalaciones MySQSL no necesitan usar esta tabla en absoluto.

- Las tablas `tables_priv` y `columns_priv` son similares a la tabla `db` , pero son más detalladas: se aplican a nivel de tabla y de columna en lugar de a nivel de base de datos. Un privilegio otorgado a nivel de tabla se aplica a la tabla y a todas sus columnas. Un privilegio otorgado a nivel de columna se aplica sólo a la columna especificada.

- La tabla `procs_priv` se aplica a rutinas almacenadas. Un privilegio otorgado a nivel de rutina se aplica sólo a una única rutina.

Permisos administrativos (tales como `RELOAD` o `SHUTDOWN`) se especifican sólo en la tabla `user` . Esto es debido a que las operaciones administrativas son operaciones del propio servidor y no específicas de bases de datos, así que no hay ninguna razón para listar estos privilegios en las otras tablas de permisos. De hecho, para determinar si puede realizar una operación administrativa, el servidor sólo necesita consultar la tabla `user`.

El privilegio `FILE` también se especifica sólo en la tabla `user` . No es un privilegio administrativo como tal, pero la habilidad de leer o escribir archivos en el equipo servidor es independiente de las bases de datos a las que acceda.

El servidor `mysqld` lee los contenidos de las tablas de permisos en memoria cuando arranca. Puede decirle que las vuelva a leer mediante el comando `FLUSH PRIVILEGES` o ejecutando los comandos `mysqladmin flush-privileges` o `mysqladmin reload` . Los cambios en las tablas de permisos tienen efecto como se indica en Sección 5.6.7, “Cuándo tienen efecto los camios de privilegios”.

Cuando modifica los contenidos de las tablas de permisos, es una buena idea asegurarse que sus cambios configuran permisos tal y como desea. Para consultar los permisos de una cuenta dada, use el comando `SHOW GRANTS` . Por ejemplo, para determinar los permisos que se otorgan a una cuenta con valores `Host` y `User` de `pc84.example.com` y `bob`, use este comando:

```
mysql> SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

Una herramienta de diagnóstico útil es el script `mysqlaccess` , proporcionado por Yves Carlier para la distribución MySQL. Invoque `mysqlaccess` con la opción `--help` para ver cómo funciona. Note que `mysqlaccess` chequea acceso usando sólo las tablas `user`, `db`, y `host` . No chequea los privilegios de tabla, columna, o rutina especificados en las tablas `tables_priv`, `columns_priv`, o `procs_priv`.

Para ayuda adicional en problemas relacionados con el diagnóstico de permisos, consulte Sección 5.6.8, “Causas de errores `Access denied`”. Para consejos generales sobre seguridad , consulte Sección 5.5, “Cuestiones de seguridad general”.

5.6.3. Privilegios de los que provee MySQL

La información sobre los privilegios de las cuentas está almacenada en las tablas `user`, `db`, `host`, `tables_priv`, `columns_priv`, y `procs_priv` de la base de datos `mysql`. El servidor MySQL lee el contenido de estas tablas y lo almacena en memoria cuando se inicia, y lo relee bajo ciertas circunstancias indicadas en Sección 5.6.7, “Cuándo tienen efecto los camios de privilegios”. Las decisiones de control de acceso se basan en la copia en memoria de las tablas grant.

Los nombres utilizados en las sentencias `GRANT` y `REVOKE` para referirse a privilegios se muestran en la siguiente tabla, junto al nombre de columna asociado con cada privilegio en las tablas grant y el contexto en que el privilegio se aplica. Más información sobre el significado de cada privilegio se puede encontrar en Sección 13.5.1.3, “Sintaxis de `GRANT` y `REVOKE`”.

Privilegio	Columna	Contexto
`CREATE`	`Create_priv`	bases de datos, tablas, o índices
`DROP`	`Drop_priv`	bases de datos o tablas
`GRANT OPTION`	`Grant_priv`	bases de datos, tablas, o procedimientos almacenados
`REFERENCES`	`References_priv`	bases de datos o tables
`ALTER`	`Alter_priv`	tablas
`DELETE`	`Delete_priv`	tablas
`INDEX`	`Index_priv`	tablas
`INSERT`	`Insert_priv`	tablas
`SELECT`	`Select_priv`	tablas
`UPDATE`	`Update_priv`	tablas
`CREATE VIEW`	`Create_view_priv`	vistas
`SHOW VIEW`	`Show_view_priv`	vistas
`ALTER ROUTINE`	`Alter_routine_priv`	procedimientos almacenados
`CREATE ROUTINE`	`Create_routine_priv`	procedimientos almacenados
`EXECUTE`	`Execute_priv`	procedimientos almacenados
`FILE`	`File_priv`	acceso a archivos en la máquina del servidor
`CREATE TEMPORARY TABLES`	`Create_tmp_table_priv`	administración del servidor
`LOCK TABLES`	`Lock_tables_priv`	administración del servidor
`CREATE USER`	`Create_user_priv`	administración del servidor

PROCESS	Process_priv	administración del servidor
RELOAD	Reload_priv	administración del servidor
REPLICATION CLIENT	Repl_client_priv	administración del servidor
REPLICATION SLAVE	Repl_slave_priv	administración del servidor
SHOW DATABASES	Show_db_priv	administración del servidor
SHUTDOWN	Shutdown_priv	administración del servidor
SUPER	Super_priv	administración del servidor

CREATE VIEW y SHOW VIEW fueron añadidos en MySQL 5.0.1. CREATE USER, CREATE ROUTINE, y ALTER ROUTINE fueron añaidos en MySQL 5.0.3. Aunque EXECUTE ya estaba presente en MySQL 5.0.0, no se activó hasta MySQL 5.0.3. Para utilizar estos privilegios cuando se actualice desde una versión más antigua de MySQL que no los tiene, debe actualizar sus tablas grant utilizando el script mysql_fix_privilege_tables proporcionado con la distribución MySQL. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Para crear o modificar procedimientos almacenados cuando el registro binario está activado, debe también tener el privilegio SUPER, tal y como se explica en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

Los privilegios CREATE y DROP permiten crear nuevas bases de datos y tablas, o eliminar las existentes. *Si otorga el privilegio DROP para la base de datos mysql a un usuario, ese usuario puede eliminar la base de datos en la que MySQL almacena los privilegios de acceso.*

Los privilegios SELECT, INSERT, UPDATE, and DELETE permiten realizar operaciones en registros de tablas existentes en una base de datos.

Las sentencias SELECT requieren el privilegio SELECT sólo si realmente extraen registros de una tabla. Algunas sentencias SELECT no acceden a tablas, y pueden ser ejecutados por tanto sin permiso para ninguna base de datos. Por ejemplo, podría utilizar el cliente mysql como una simple calculadora para evaluar expresiones que no hacen referencia a tablas:

```
mysql> SELECT 1+1;
mysql> SELECT PI()*2;
```

El privilegio INDEX permite crear o eliminar índices. INDEX es aplicable a tablas existentes. Si tiene el privilegio CREATE para una tabla, entonces puede incluir definiciones de índices en la sentencia CREATE TABLE.

El privilegio ALTER permite utilizar ALTER TABLE para cambiar la estructura de o renombrar tablas.

El privilegio CREATE ROUTINE es necesario para crear procedimientos almacenados (funciones y procedimientos). El privilegio ALTER ROUTINE se necesita para modificar o eliminar procedimientos almacenados, y EXECUTE es necesario para ejecutarlos.

El privilegio GRANT permite dar a otros usuarios las privilegios que uno mismo posee. Puede ser utilizado para bases de datos, tablas, y procedimientos almacenados.

El privilegio FILE otorga permiso para leer y escribir archivos en la máquina del servidor utilizando las sentencias LOAD DATA INFILE y SELECT ... INTO OUTFILE. Un usuario que tiene el privilegio FILE puede leer cualquier archivo de la máquina del servidor que sea legible por cualquiera o por el usuario que ejecuta el servidor MySQL. (Esto implica que el usuario puede leer cualquier archivo en el directorio de datos, porque el servidor puede acceder a cualquiera de estos archivos.) El privilegio FILE también

permite al usuario crear archivos nuevos en cualquier directorio en que el servidor MySQL tiene acceso de escritura. Los archivos existentes no pueden ser sobreescritos.

Los privilegios restantes son utilizados para operaciones administrativas. Muchas de ellas puede ser realizadas utilizando el programa `mysqladmin` o mediante sentencias SQL. La siguiente tabla muestra qué comandos de `mysqladmin` permite ejecutar cada privilegio administrativo.

Privilegio	Comandos permitidos a los poseedores del privilegio Holders
`RELOAD`	`flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh`, `reload`
`SHUTDOWN`	`shutdown`
`PROCESS`	`processlist`
`SUPER`	`kill`

El comando `reload` comunica al servidor que debe releer las tablas grant a memoria. `flush-privileges` es un sinónimo de `reload`. El comando `reload` cierra y vuelve a abrir los archivos de registro y vuelca todas las tablas. Los otros comandos `flush-`*`xxx`* realizan funciones similares a `refresh`, pero son más específicas y pueden ser preferibles en algunos casos. Por ejemplo, si quiere tan solo volcar los archivos de registro, `flush-logs` es mejor opción que `refresh`.

El comando `shutdown` apaga el servidor. Este comando puede ejecutarse únicamente desde `mysqladmin`. No hay sentencia SQL equivalente.

El comando `processlist` muestra información sobre los subprocesos que se están ejecutando en el servidor (es decir, sobre las sentencias que se están ejecutando por parte de clientes asociados con otras cuentas). El comando `kill` mata los subprocesos del servidor. Siempre puede mostrar información sobre sus propios subprocesos, o matarlos, pero necesita el privilegio `PROCESS` para ver subprocesos iniciados por otros usuarios, y el privilegio `SUPER` para matarlos. Consulte Sección 13.5.5.3, “Sintaxis de KILL”.

El privilegio `CREATE TEMPORARY TABLES` permite la utilización de la palabra clave `TEMPORARY` en sentencias `CREATE TABLE`.

El privilegio `LOCK TABLES` permite la utilización de sentencias `LOCK TABLES` explícitas para bloquear tablas para las que tiene el privilegio `SELECT`. Esto incluye el uso de bloqueos de escritura, que evita que cualquier otra persona lea la tabla bloqueada.

El privilegio `REPLICATION CLIENT` permite la utilización de las sentencias `SHOW MASTER STATUS` y `SHOW SLAVE STATUS`.

El privilegio `REPLICATION SLAVE` debería otorgarse a cuentas que son utilizadas por servidores esclavos para conectarse al servidor actual como su maestro. Sin este privilegio, la cuenta esclava no puede pedir actualizaciones que se hayan hecho a las bases de datos del servidor maestro.

El privilegio `SHOW DATABASES` permite a la cuenta ver los nombres de las bases de datos mediante la ejecución de la sentencia `SHOW DATABASE`. Cuentas que no tengan este privilegio solo pueden ver las bases de datos para las que tienen privilegios, y no pueden utilizar la sentencia de ninguna manera si el servidor ha sido iniciado con la opción `--skip-show-database`.

En general, es una buena idea garantizar a una cuenta solo aquellos privilegios que necesita. Se debe tener especial cuidado en seguir esta regla con los privilegios administrativos y `FILE`:

- El privilegio `FILE` puede utilizarse inadecuadamente para introducir en una tabla de la base de datos cualquier archivo que el servidor MySQL sea capaz de leer en la máquina del servidor. Esto incluye todos los archivos que sean legibles para cualquiera, además de los archivos almacenados en el

directorio de datos del servidor. Esta tabla puede entonces ser accedida utilizando una sentencia `SELECT` para transferir sus contenidos a la máquina cliente.

- El privilegio `GRANT` permite a los usuarios otorgar sus mismos privilegios a otros usuarios. Dos usuarios con diferentes privilegios y con el privilegio `GRANT` pueden combinar sus privilegios.
- El privilegio `ALTER` puede ser utilizado inadecuadamente para sabotear el sistema de privilegios mediante el renombrado de tablas.
- El privilegio `SHUTDOWN` puede utilizarse inadecuadamente para denegar el servicio a otros usuarios de manera total, cerrando el servidor.
- El privilegio `PROCESS` puede utilizarse para ver el texto de las consultas que se estén ejecutando actualmente, incluyendo consultas que establecen o modifican passwords.
- El privilegio `SUPER` puede utilizarse para cerrar la conexión a otros clientes o cambiar como el servidor funciona.
- Los privilegios otorgados para la propia base de datos `mysql` pueden utilizarse para cambiar passwords y otra información de privilegios de acceso. Las passwords se almacenan cifradas, así que un usuario malicioso no puede simplemente leerlas para conocer la password. Aún así, el usuario con privlegio de escritura a la columna `Password` de la tabla `user` puede cambiar la password de una cuenta, y seguidamente conectarse al servidor MySQL utilizando esa cuenta.

Hay algunas cosas que no se pueden hacer con el sistema de privilegios de MySQL:

- No se puede especificar explícitamente que a un usuario se le deba denegar el acceso.
- No se puede especificar que un usuario tenga privilegios para crear o eliminar tablas en una base de datos, pero que no pueda crear o eliminar la propia base de datos.

5.6.4. Conectarse al servidor MySQL

Los programas cliente de MySQL espera por lo general que usted especifique los parámetros de conexión cuando quiere acceder a un servidor MySQL:

- El nombre de la máquina donde se está ejecutando el servidor MySQL
- Su nombre de usuario
- Su password

Por ejemplo, el cliente `mysql` puede ejecutarse desde un prompt de línea de comandos (indicado aquí por `shell>`) de la siguiente manera:

```
shell> mysql -h nombre_host -u nombre_usuario -psu_clave
```

Las sintaxis alternativas de las opciones `-h`, `-u`, y `-p` son `--host=nombre_host`, `--user=nombre_usuario`, y `--password=su_clave`. Nótese que *no hay espacios* entre `-p` o `--password=` y la clave que le sigue.

Si utiliza una opción `-p` o `--password` pero no especifica un valor para la clave, el programa cliente le pedirá que introduzca la clave. La clave no se mostrará mientras la introduce. Esto es más seguro que especificar la clave en la línea de comandos. Cualquier usuario de su sistema podría ser capaz de ver la clave especificada en la línea de comandos ejecutando un comando como `ps auxww`. Consulte Sección 5.7.6, “Guardar una contraseña de forma segura”.

Los programas clientes de MySQL utilizan valores por defecto para cualquier parámetro que no se especifique:

- El nombre de servidor por defecto es `localhost`.
- El nombre de usuario por defecto es `ODBC` en Windows y su nombre de usuario de Unix en Unix.
- No se aplica ninguna clave si `-p` no está especificado.

De esta manera, para un usuario de Unix con un nombre de usuario de `jose`, todos los siguientes comandos son equivalentes:

```
shell> mysql -h localhost -u jose
shell> mysql -h localhost
shell> mysql -u jose
shell> mysql
```

Otros clientes MySQL se comportan de manera similar.

Puede especificar valores diferentes para que se utilicen cuando se realiza una conexión de manera que no tenga que introducirlos en la línea de comandos cada vez que invoca un programa cliente. Esto puede llevarse a cabo de diversas maneras:

- Puede especificar los parámetros de conexión en la sección `[client]` de un archivo de opciones. La sección relevante del archivo debería tener el siguiente aspecto:

  ```
  [client]
  host=nombre_servidor
  user=nombre_usuario
  password=su_clave
  ```

  Los archivos de opciones son explicados en profundidad en Sección 4.3.2, “Usar ficheros de opciones”.

- Puede especificar algunos parámetros de conexión utilizando variables de ambiente. El nombre del servidor para `mysql` puede ser especificado utilizando `MYSQL_HOST`. El nombre de usuario MySQL puede especificarse mediante `USER` (esto es para Windows y Netware únicamente). La clave se puede especificar utilizando `MYSQL_PWD`, aunque esto es inseguro; consulte Sección 5.7.6, “Guardar una contraseña de forma segura”. Para ver la lista de variables, consulte Apéndice E, *Variables de entorno*.

5.6.5. Control de acceso, nivel 1: Comprobación de la conexión

Cuando intente conectar a un servidor MySQL, el servidor aceptará o rechazará la conexión basándose en su identidad y si usted puede identificar su identidad proporcionando la clave correcta. Si no es así, el servidor le denegará el acceso completamente. En caso contrario, el servidor acepta la conexión, y entra en el Estado 2, y espera peticiones.

Su identidad se basa en dos elementos de información:

- El nombre de máquina cliente (o ip) desde donde usted se conecta
- Su nombre de usuario MySQL

La comprobación de la identidad se realiza utilizando las tres columnas de la tabla `user` (`Host`, `User`, y `Password`). El servidor sólo acepta la conexión si las columnas `Host` y `User` de alguna de las tablas `user` es coincidente con el nombre de máquina y usuario del cliente, y además el cliente proporciona la clave especificada en ese registro.

Los valores de `Host` en la tabla `user` pueden ser especificados de las siguientes maneras:

- Un valor de `Host` debe ser un nombre de máquina o un número IP, o `'localhost'` para indicar la máquina local.
- Puede utilizar los caracteres comodín '`%`' y '`_`' en los valores de las columnas `Host`. Estos tienen el mismo significado que en las operaciones de búsqueda de patrones realizadas mediante el operador `LIKE`. Por ejemplo, un valor de `Host` igual a `'%'` retorna cualquier nombre de máquina, así como un valor de `'%.mysql.com'` retorna cualquier nombre de máquina en el dominio `mysql.com`.
- Para valores de `Host` especificados como números IP, puede especificar una máscara de red indicando cuantos bits de la dirección utilizar para el número de red. Por ejemplo:

  ```
  mysql> GRANT ALL PRIVILEGES ON db.*
      -> TO david@'192.58.197.0/255.255.255.0';
  ```

  Esto permite a `david` conectarse desde cualquier cliente que tenga un número IP `client_ip` para el que la siguiente condición sea cierta:

  ```
  client_ip & netmask = host_ip
  ```

  Es decir, para la sentencia `GRANT` recientemente mostrada: That is, for the `GRANT` statement just shown:

  ```
  client_ip & 255.255.255.0 = 192.58.197.0
  ```

  Los números IP que satisfagan esta condición y pueden conectar al servidor MySQL son lo que están en el rango desde `192.58.197.0` hasta `192.58.197.255`.

- Nota: La máscara de red solo puede ser utilizada para decirle al servidor que use 8, 16, 24 o 32 bits para la dirección, por ejemplo:

  ```
  192.0.0.0/255.0.0.0 (cualquier dirección de la red clase A 192)
  192.168.0.0/255.255.0.0 (cualquier dirección de la red clase B 192.168)
  192.168.1.0/255.255.255.0 (cualquier dirección de la red clase C 192.168.1)
  192.168.1.1 (solo esta IP específica)
  ```

  La siguiente máscara de red (28 bits) no funcionará:

  ```
  192.168.0.1/255.255.255.240
  ```

- Un valor vacío de `Host` en un registro de la tabla `db` significa que los privilegios de dicho registro deben ser combinados con aquellos que se encuentren en el registro de la tabla `host` que concuerde con el nombre del cliente. Los privilegios se combinan utilizando operaciones AND (intersección), no OR (union). Puede encontrar más información sobre la tabla `host` en Sección 5.6.6, “Control de acceso, nivel 2: comprobación de solicitudes”.

  Un valor de `Host` en blanco en las otras tablas grant lo mismo que `'%'`.

Debido a que puede usar comodines en los valores IP de la columna `Host` (por ejemplo, `'144.155.166.%'` para conseguir cualquier IP en una subred), alguien podría intentar explotar esta capacidad poniéndole a su cliente el nombre `144.155.166.cualquierhost.com`. Para evitar estos intentos, MySQL no permite que los comodines sean utilizados para los nombres de cliente que empiezan con dígitos y un punto. Así que si tiene un cliente con nombre similar a `1.2.cualquierhost.com`, su nombre nunca

concordará con la columna `Host` de las tablas grant. Un comodín de IP solo puede concordar con números IP, no nombres de cliente.

En la columna `User`, los caracteres comodín no están permitidos, pero puede especificar un valor en blanco, que será válido para cualquier nombre. Si el registro de la tabla `user` que concuerda con una conexión entrante tiene un valor vacio, el usuario es considerado anónimo, sin nombre de usuario, no un usuario con el nombre que el cliente especificó realmente. Esto significa que el nombre de usuario vacío es utilizado para todas las comprobaciones de acceso posteriores durante la duración de la conexión (es decir, durante el Estado 2).

La columna `Password` puede estar vacía. Esto no es un comodín que permite que cualquier clave sea permitida. Significa que el usuario debe conectarse sin especificar una clave.

Los valores que no están vacíos de `Password` en la tabla `user` representan claves cifradas. MySQL no almacena las claves en forma de texto llano para que cualquiera pueda verlo. En vez de esto, la clave suministrada por un usuario que se está intentando conetar es cifrada (utilizando la función `PASSWORD()`). La clave cifrada se utiliza entonces durante el proceso de conexión en el momento de comprobar si es correcta. (Esto se realiza sin que la clave cifrada viaje nunca sobre la conexión.) Desde el punto de vista de MySQL, la clave cifrada es la clave REAL, así que no debería darse acceso a ella a nadie. En concreto, no de acceso de lectura a las tablas de la base de datos `mysql` a usuarios no-administrativos.

MySQL 5.0 utiliza el método de autenticación más fuerte (implementado primeramente en MySQL 4.1) y que tiene una mejor protección de la clave durante el proceso de conexión que versiones previas. Es seguro aún cuando los paquetes TCP/IP fuesen interceptados o la base de datos `mysql` capturada. El cifrado de claves es comentado en mayor profundida en Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1”.

Los siguientes ejemplos nos enseñan como se aplicarían a las conexiones entrantes diferentes combinaciones de los valores de las columnas `Host` y `User` de la tabla `user`.

`Cliente` **Valor**	`Usuario` **Valor**	**Conexiones que concuerdan con la entrada Entry**
`'thomas.loc.gov'`	`'fred'`	`fred`, conectando desde `thomas.loc.gov`
`'thomas.loc.gov'`	`''`	Cualquier usuario, conectando desde `thomas.loc.gov`
`'%'`	`'fred'`	`fred`, conectando desde cualquier cliente
`'%'`	`''`	Cualquier usuario conectando desde cualquier cliente
`'%.loc.gov'`	`'fred'`	`fred`, conectando desde cualquier cliente en el dominio `loc.gov`
`'x.y.%'`	`'fred'`	`fred`, conectando desde `x.y.net`, `x.y.com`, `x.y.edu`, etc. (esto, probablemente, no es útil)
`'144.155.166.177'`	`'fred'`	`fred`, conectando desde el cliente con dirección IP `144.155.166.177`
`'144.155.166.%'`	`'fred'`	`fred`, conectando desde cualquier cliente en la subred de clase C `144.155.166`
`'144.155.166.0/255.255.255.0"`	`'fred'`	Idéntico al ejemplo anterior

Es posible que el nobre del cliente y del usuario de una conexión entrante concuerde con más de un registro en la tabla `user`. El conjunto de ejemplos precedentes demuestra esto: Algunas de las entradas mostradas concuerdan con una conexión de `fred` desde `thomas.loc.gov`.

Cuando hay la posibilidad de múltiples concordancias, el servidor debe determinar cual de ellas utilizar. El problema se resuelve de la siguiente manera:

- Siempre que el servidor lee la tabla `user` a memoria, ordena los registros.
- Cuando un cliente intenta conectar, el servidor mira a través de los registros en el orden establecido.
- El servidor utiliza el primer registro que concuerda con el nombre y usuario del cliente.

Para ver como esto ocurre, supongamos que la tabla `user` es como esta:

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| %         | root     | …
| %         | jeffrey  | …
| localhost | root     | …
| localhost |          | …
+-----------+----------+-
```

Cuando el servidor lee la tabla, ordena las entradas con los valores de `Host` más específicos primero. Los nombres de cliente y números de IP son los más específicos. El comodín `'%'` significa ``cualquier cliente'' y es menos específico. Registros con el mismo valor de `Host` se ordenan con el valor de `User` más específico (un valor de `User` en blanco significa ``cualquier usuario'' y es menos específico). En la tabla `user` recién mostrada, el resultado después de ordenar sería el siguiente:

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| localhost | root     | … ...
| localhost |          | … ...
| %         | jeffrey  | … ...
| %         | root     | … ...
+-----------+----------+-
```

Cuando un cliente intenta conectar, el servidor mira los registros ordenados y utiliza la primera concordancia. Para una conexión desde `localhost` por `jeffrey`, dos de las entradas de la tabla concuerdan: la primera con los valores `'localhost'` y `''` de `Host` y `User` respectivamente, y el registro con los valores `'%'` y `'jeffrey'`. El registro con valor `'localhost'` aparece primero en la tabla ordenada, así que es el que el servidor utiliza.

Aquí hay otro ejemplo. Supongamos que la tabla `user` tiene el siguiente aspecto:

```
+----------------+----------+-
| Host           | User     | …
+----------------+----------+-
| %              | jeffrey  | …
| thomas.loc.gov |          | …
+----------------+----------+-
```

La tabla ordenada sería:

```
+----------------+----------+-
| Host           | User     | …
+----------------+----------+-
| thomas.loc.gov |          | …
| %              | jeffrey  | …
```

```
+---------------+----------+-
```

Una conexión de `jeffrey` desde `thomas.loc.gov` concuerda con el primer registro, mientras que una conexión de `jeffrey` desde `whitehouse.gov` concuerda con el segundo.

Es una confusión común el pensar que, para un nombre de usuario dado, todos los registros que nombran explícitamente con a ese usuario son utilizadas primero cuando el servidor intenta encontrar una concordancia para una conexión. Esto es sencillamente falso. El ejemplo anterior ilustra esto, donde una conexión de `jeffrey` desde `thomas.loc.gov` no concuerda primero con el registro que contiene `'jeffrey'` como valor en la columna `User`, sino que ha concordado con el registro que no contiene nombre de usuario. Como resultado, `jeffrey` es tratado como un usuario anónimo aunque ha especificado un nombre de usuario al conectarse.

Si puede conectar al servidor, pero sus privilegios no son los que espera, probablemente está siendo identificado como algún otro usuario. Para averiguar qué cuenta de usuario utilizó el servidor para identificarle, use la función `CURRENT_USER()`. Devuelve un valor en formato *`user_name@host_name`* que indica los valores de `User` y `Host` del registro concordante en la tabla `user`. Suponga que `jeffrey` conecta y ejecuta la siguiente sentencia:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

El resultado mostrado indica que el registro que concuerda en la tabla `user` tiene un valor vacío en la columna `User`. En otras palabras, el servidor trata a `jeffrey` como a un usuario anónimo.

Otra cosa que puede hacer para localizar problemas de autentificación es imprimir por pantalla la tabla `user` y ordenarla a mano para ver donde está la primera concordancia. Consulte Sección 12.9.3, “Funciones de información”.

5.6.6. Control de acceso, nivel 2: comprobación de solicitudes

Una vez establecida una conexión, el servidor entra en el estado 2 del control de acceso. Por cada petición que viene en la conexión, el servidor determina que operación realizar, y entonces comprueba si tiene suficientes privilegios para hacerlo. Aquí es donde las columnas de privilegios de las tablas grant entran en juego. Esots privilegios puede venir de cualquiera de las tablas `user`, `db`, `host`, `tables_priv`, o `columns_priv`.(Puede encontrar útil consultar Sección 5.6.2, “Cómo funciona el sistema de privilegios”, que enumera las columnas presentes en cada una de las tablas grant.)

La tabla `user` otorga privilegios que se asignan de manera global, y que se aplican sin importar sobre qué base de datos trabajamos. Por ejemplo, si la tabla `user` le otorga el privilegio `DELETE`, usted podrá borrar registros de cualquier tabla en cualquier base de datos en todo el servidor. En otras palabras, los privilegios de la tabla `user` son privilegios de superusuario. Es aconsejable otorgar privilegios en la tabla `user` sólo a superusuarios tales como administradores de base de datos. Para otros usuarios, debería dejar los privilegios de la tabla `user` con el valor `'N'` y otorgar los privilegios únicamente a niveles más específicos. Puede otorgar privilegios para bases de datos, tablas o columnas particulares.

Las tablas `db` y `host` otorgan privilegios específicos para una base de datos. Los valores en las columnas de estas tablas pueden tener los siguientes formatos:

- Los caracteres comodín '`%`' y '`_`' pueden utilizarse en las columnas `Db` de cualquiera de las tablas. Estos tienen el mismo significado que en las operaciones de reconocimiento de patrones efectuadas con el

operador `LIKE`. Si quiere utilizar cualquiera de estos caracteres literales al otorgar privilegios, debe incluirlos en una secuencia de escape con una barra invertida. Por ejemplo, para incluir el carácter '_' como parte del nombre de una base de datos, especifíquelo como '_' en la sentencia `GRANT`.

- Un valor de `'%'` en la columna `Host` de la tabla `db` significa ``cualquier host.'' Un valor vacío en la columna `Host` de la tabla `db` significa ``consulta la tabla `host` para más información'' (un proceso que se describe más adelante en esta sección).
- Un valor de `'%'` o en blanco en la columna `Host` de la tabla `host` significa ``cualquier host.''
- Un valor de `'%'` o en blanco de la columna `Db` en cualquiera de las dos tablas, significa ``cualquier base de datos.''
- Un valor en blanco de la columna `User` en cualquiera de las tablas concuerda con el usuario anónimo.

El servidor lee y ordena las tablas `db` y `host` al mismo tiempo que lee la tabla `user`. El servidor ordena la tabla `db` basándose en el rango de las columnas `Host`, `Db` y `User`, y ordena la tabla `host` basándose en el rango de las columnas `Host` y `Db`. Igual que con la tabla `user`, la ordenación coloca los valores menos específicos en última posición, y cuando el servidor busca correspondencias, utiliza la primera que encuentra.

Las tablas `tables_priv` y `columns_priv` otorgan privilegios específicos para tablas y columnas respectivamente. Los valores en las columnas de rango de estas tablas pueden tener los siguientes formatos:

- Los caracteres comodín '`%`' y '`_`' pueden ser utilizados en la columna `Host` de cualquiera de las tablas. Estos comodines tienen el mismo significado que en las operaciones de búsqueda de patrones realizadas con el operador `LIKE`.
- Un valor de `'%'` o vacío en la columna `Host` de cualquiera de las tablas significa ``cualquier host.''
- Las columnas `Db`, `Table_name` y `Column_name` no pueden contener caracteres comodín ni estar en blanco en ninguna de las tablas.

El servidor ordena las tablas `tables_priv` y `columns_priv` basándose en las columnas `Host`, `Db`, y `User`. Esto es similar a la ordenación de la tabla `db`, pero más simple, porque únicamente la columna `Host` puede contener comodines.

El proceso de verificación de peticiones se describe aquí. (Si usted está familirizado con el código fuente de control de acceso, se dará cuenta de que la descripción aquí contenida difiere ligeramente de el algoritmo utilizado en el código. La descripción es equivalente a lo que el código hace realmente; solo difiere para hacer la explicación más simple.)

Para peticiones que requieran privilegios de administrador, como `SHUTDOWN` o `RELOAD`, el servidor comprueba únicamente el registro de la tabla `user` porque es la única tabla que especifica los privilegios administrativos. El acceso se otorga si el registro permita la operación demandada, y es denegado en caso contrario. Por ejemplo, si usted quisiera ejecutar `mysqladmin shutdown`, pero su registro de la tabla `user` no le otorga el privilegio `SHUTDOWN`, el servidor deniega el acceso sin ni siquiera codnsultar las tablas `db` o `host`. (No contienen ninguna columna `Shutdown_priv`, así que no hay necesidad de hacerlo.)

Para peticiones sobre bases de datos (`INSERT`, `UPDATE`, etc.), el servidor primero comprueba los privilegios globales del usuario (superuser) mirando el registro de la tabla `user`. Si el registro permite la operación demandada, se otorga el acceso. Si los privilegios globales de la tabla `user` son insuficientes, el servidor determina los privilegios específicos sobre la base de datos comprobando las tablas `db` y `host`:

1. El servidor busca en la tabla `db` una concordancia en las columnas `Host`, `Db` y `User`. Las columnas `Host` y `User` se hacen concordar con el nombre de host y de usuario MySQL. La columna `Db` se hace concordar con la base de datos a la que el usuario quiere acceder. Si no hay ningún registro para `Host` y `User`, se deniega el acceso.

2. Si hay un registro que concuerda en el registro de la tabla `db` y su columna `Host` no está vacía, ese registro define los privilegios específicos del usuario en la base de datos.

3. Si la columna `Host` del registro concordante de la tabla `db` se encuentra vacía, significa que la tabla `hosts` enumera qué hosts pueden tener acceso a la base de datos. En este caso, una comprobación más se realiza en la tabla `host` para encontrar una concordancia en las columnas `Host` y `Db`. Si ningún registro de la tabla `host` concuerda, se deniega el acceso. Si hay una concordancia, los privilegios específicios sobre la base de datos del usuario son calculados como la intersección (¡*no* unión!) de los privilegios en los registros de las tablas `db` y `host`; es decir, los privilegios que tienen valor `'Y'` en ambos registros. (De esta manera puede otorgar privilegios en el registro de la tabla `db` y entonces restringir selectivamente host a host utilizando los registros de la tabla `hosts`.)

Tras determinar los privilegios específicos de la base de datos otorgados por los registros de las tablas `db` y `host`, el servidor los añade a los privilegios globales otorgados por la tabla `user`. Si el resultado permite la operación demandada, se otorga el acceso. En caso contrario, el servidor comprueba sucesivamente la tabla del usuario y los privilegios de las columnas en las tablas `tables_priv` y `columns_priv`, los añade a los privilegios del usuario, y permite o deniega el acceso basándose en el resultado.

Expresado en términos booleanos, la descripción precedente de como se calculan los privilegios de un usuario, se puede resumir en:

```
privilegios globales
O (privilegios de base de datos Y privilegios de host)
O privilegios de tabla
O privilegios de columna
```

Puede no ser evidente por qué, si los privilegios globales del registro en la tabla `user` no han sido inicialmente suficientes para la operación demandada, el servidor añado estos privilegios a los de base de datos, tabla y columna más tarde. La razón es que una petición puede requerir más de un tipo de privilegio. Por ejemplo, si usted ejecuta una sentencia `INSERT INTO ... SELECT`, necesita tanto el privilegio `INSERT` como el privilegio `SELECT`. Sus privilegios podrían estar configurados de manera que la tabla `user` otorgue un privilegio, y la tabla `db` otorgue el otro. En este caso, necesita ambos privilegios para realizar la sentencia, pero el servidor no puede saberlo desde cada una de las tablas únicamente; los privilegios otorgados por los registros de ambas tablas deben ser combinados.

La tabla `host` no es afectada por sentencias `GRANT` o `REVOKE`, así que en la mayoría de las instalaciones MySQL queda sin utilizar. Si usted la modifica directamente, puede utilizarla para algunos propósitos específicos, como mantener una lista de servidores seguros. Por ejemplo, en TcX, la tabla `host` contiene una lista de todas las máquinas en la red local. Éstas tienen otorgados todos los privilegios.

También puede utilizar la tabla `host` para indicar hosts que *no* son seguros. Supongamos que tiene una máquina `public.su.dominio` que está situada en un lugar público que no considera seguro. Puede permitir el acceso a todos los hosts de su red excepto a esa máquina utilizando registros de la tabla `host` como este:

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

Nauturalmente, usted debe siempre comprobar sus entradas en las tablas grant (por ejemplo, utilizando `SHOW GRANTS` o `mysqlaccess`) para estar seguro de que sus privilegios de acceso son realmente los que piensa que son.

5.6.7. Cuándo tienen efecto los camios de privilegios

Cuando `mysqld` se inicia, todos los contenidos de las tablas grant se leen a memoria y se hacen efectivas para el control de acceso en ese punto.

Cuando el servidor recarga las tablas grant, los privilegios para los conexiones de clientes existentes se ven afectadas de la siguiente manera:

- Los cambios en los privilegios de tabla y columna toman efecto en la siguiente petición del cliente.
- Los cambios en privilegio sde base de datos toman efecto en la siguiente sentencia `USE db_name`.
- Los cambios a los privilegios globales y las claves de acceso toman efecto la próxima vez que el cliente se conecte.

Si usted modifica las tablas grant utilizando `GRANT`, `REVOKE`, o `SET PASSWORD`,el servidor se da cuenta de estos cambios y recarga las tablas grant en la memoria inmediatamente.

Si usted modifica las tablas grant directamente utilizando sentencias como `INSERT`, `UPDATE`, o `DELETE`, los cambios no tendrán efecto en la comprobación de privilegios hasta que se reinicie el servidor, o bien se le comunique a éste que debe recargar las tablas. Para recargar las tablas manualmente, ejecute la sentencia `FLUSH PRIVILEGES` o los comandos `mysqladmin flush-privileges` o `mysqladmin reload`.

Si usted cambia las tablas grant directamnete pero olvida recargarlas, sus cambios *no tienen efecto* hasta que reinicie el servidor. Esto podría confundirle intentando averiguar por qué sus cambios no parecen tener efecto.

5.6.8. Causas de errores `Access denied`

Si usted se encuentra problemas cuando intenta conectar al servidor MySQL, los siguientes elementos explican algunas medidas que se pueden tomar para corregir el problema.

- Asegúrese de que el servidor se está ejecutando. Si no se está ejecutando, no puede conectarse a él. Por ejemplo, si intenta conectarse a el servidor y ve un mensaje como cualquiera de los siguientes, podría ser que el servidor no se esté ejecutando:

```
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)
```

  También podría ser que el servidor se esté ejecutando, pero usted se esté intentando conectar utilizando un puerto TCP/IP, named pipe, o archivo socket de Unix diferentes de aquellos a los que el servidor atiende. Para corregir esto cuando invoca a un programa cliente, especifique la opción `--port` para indicar el puerto adecuado, o la opción `--socket` para indicar la named pipi o el archivo socket de Unix apropiados. Para averiguar dónde se encuentra el archivo socket, puede ejecutar:

```
shell> netstat -ln | grep mysql
```

- Las tablas grant deben estar correctamente configuradas para que el servidor pueda utilizarlas en el control de acceso. Para algunos tipos de distribución (como las distribuciones binarias de Windows,

o las distribuciones RPM de Linux), el proceso de instalación inicializa la base de datos mysql que contiene las tablas grant. Para distribuciones que no hacen esto, usted debe inicializar las tablas grant manualmente ejecutando el script mysql_install_db. Para más detalles, consulte Sección 2.9.2, “Pasos a seguir después de la instalación en Unix”.

Una manera de determinar si debe inicializar las tablas grant es buscar un directorio mysql bajo el directorio de datos. (El directorio de datos normalmente se llama data o var y se encuentra bajo su directorio de instalación de MySQL.) Asegúrese de que tiene un archivo llamado user.MYD en el directorio mysql de la base de datos. Si no, ejecute el script mysql_install_db. Tras ejecutar este script e iniciar el servidor, compruebe los privilegio siniciales ejecutando este comando:

```
shell> mysql -u root test
```

El servidor debería dejarle conectar sin error.

- Tras una nueva instalación, usted debería conectarse al servidor y configurar sus usuarios y sus permisos de acceso:

  ```
  shell> mysql -u root mysql
  ```

  El servidor debería dejarle conectar porque el usuario MySQL root no tiene clave de acceso inicialmente. Esto, además, es un riesgo de seguridad, así que asignar la clave de acceso para las cuentas root es algo que debe hacer mientras se configuran el resto de usuarios. Puede consultar las instrucciones para asignar las claves iniciales aquí Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”.

- Si usted ha actualizado una instalación MySQL existente a una nueva versión, ¿ejecutó el script mysql_fix_privilege_tables? En caso negativo, hágalo. La estructura de las tablas grant cambia ocasionalmente cuando se añaden nuevas características, así que tras una actualización debería siempre asegurarse de que sus tablas tienen la estructura actual. Para más instrucciones, consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

- Si un programa cliente recibe el siguiente mensaje de error cuando intenta conectar, significa que el servidor está esperando las claves de acceso en un formato más nuevo del que el cliente es capaz de generar:

  ```
  shell> mysql
  Client does not support authentication protocol requested
  by server; consider upgrading MySQL client
  ```

  Para más información sobre atajar este problema, consulte Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1” y Sección A.2.3, “Client does not support authentication protocol”.

- Si intenta conectarse como root y obtiene el siguiente error, significa que usted no tiene un registro en la tabla user con un valor en la columna User de 'root' y que mysqld no puede resolver el nombre de host para su cliente:

  ```
  Access denied for user ''@'unknown' to database mysql
  ```

  En este caso, debería reiniciar el servidor con la opción --skip-grant-tables y editar su archivo /etc/hosts o \windows\hosts para añadir una entrada para su host.

- Recuerde que los programas cliente utilizan parámetros de conexión especificados en archivos de opciones o variables de entorno. Si un programa cliente parece estar enviando parámetros de conexión por defecto incorrectos cuando no los especifica en línea de comandos, compruebe su entorno y

cualquier archivo de opciones implicado. Por ejemplo, si obtiene Access denied cuando intenta ejecutar un cliente sin ninguna opción, asegúrese de que no ha especificado una clave de acceso antigua en sus archivos de opciones.

Puede suprimir el uso de archivos de opciones por parte de un programa cliente invocándolo con la opción --no-default. Por ejemplo:

```
shell> mysqladmin --no-defaults -u root version
```

Los archivos de opciones que los clientes utilizan están enumerados en Sección 4.3.2, “Usar ficheros de opciones”. Las variables de entorno se enumeran en Apéndice E, *Variables de entorno*.

- Si obtiene el siguiente error, significa que está utilizando una clave de root incorrecta:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

Si este error precedente ocurre aún cuando usted no ha especificado ninguna clave de acceso, significa que tiene una clave incorrecta en algún archivo de opciones. Intente utilizar la opción --no-defaults como se explica en el punto anterior.

Para más información sobre el cambio de claves, consulte Sección 5.7.5, “Asignar contraseñas a cuentas”.

Si usted ha perdido u olvidado la clave de root, puede reiniciar mysqld con --skip-grant-tables para cambiar la clave. Consulte Sección A.4.1, “Cómo reiniciar la contraseña de root”.

- Si usted cambia la clave utilizando SET PASSWORD, INSERT, o UPDATE, debe cifrar la clave utilizando la función PASSWORD(). Si no utiliza PASSWORD() para estas sentencias, la clave no funcionará. Por ejemplo, la siguiente sentencia asigna una clave, pero no la cifra, así que el usuario no es capaz de conectar tras ella:

```
mysql> SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

En vez de esto, debe establecer la clave así:

```
mysql> SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

La función PASSWORD() no es necesaria cuando se especifica la clave utilizando sentencias GRANT o (a partir de MySQL 5.0.2) CREATE USER, o también con el comando mysqladmin password, los cuales utilizan automáticamente PASSWORD() para cifrar la clave. Consulte Sección 5.7.5, “Asignar contraseñas a cuentas” y Sección 13.5.1.1, “Sintaxis de CREATE USER”.

- localhost es un sinónimo para su nombre de máquina local, y también es la máquina por defecto al que los clientes se intentan conectar si no se especifica explícitamente.

  Para evitar este problema en sistemas como ese, puede utilizar la opción --host=127.0.0.1 para mencionar la máquina t explícitamente. Esto crea una conexión TCP/IP al servidor mysqld local. También puede utilizar TCP/IP especificando una opción --host que utilice el nombre real de la máquina local. En este caso, el nombre de host debe ser especificado en una fila de la tabla user del servidor, aun cuando el cliente esté corriendo en la misma máquina que el servidor.

- Si obtiene un error Access denied cuando intenta conectarse a la base de datos con mysql- u user_name, puede que tenga un problema con la tabla user. Compruebe esto ejecutando mysql -u root mysql e introduciendo esta sentencia SQL:

```
mysql> SELECT * FROM user;
```

El resultado debería incluir una fila cuyas columnas Host y User coincidan con el nombre de su máquina y su nombre de usuario MySQL.

- El mensaje de error Access denied indica con qué nombre de usuario se está intentando entrar al sistema, la máquina cliente desde la que se está intentando conectar, y si se está utilizando clave de acceso o no. Normalmente, debería tener una línea en la tabla user que concuerde exactamente con el nombre de máquina y el nombre de usuario que se ha obtenido en el mensaje de error. Por ejemplo, si obtiene un mensaje de error que contiene using password: NO, significa que se ha intentado entrar sin utilizar una clave de acceso.

- Si el siguiente error aparece cuando se intenta conectar desde una máquina diferente a la que está ejecutando el servidor MySQL, significa que no hay ninguna fila en la tabla user con un valor Host que concuerde con la máquina cliente:

```
Host ... is not allowed to connect to this MySQL server
```

Puede corregir esto estableciendo una cuenta para la combinación de nombre de máquina y usuario que está utilizando cuando se intenta conectar.

Si usted no conoce el número de IP o el nombre de máquina del ordenador desde el que se está conectando, debería poner una fila con un valor de '%' en la columna Host de la tabla user, y reiniciar mysqld en el servidor con la opción --log. Tras intentar conectar desde la máquina cliente, la información en el log de MySQL indica desde donde se conectó realmente. (Entonces cambie el valor '%' en la tabla user para introducir el nombre real de la máquina que se muestra en el log. De otra manera obtendría un sistema inseguro, porque permitiría conexiones desde cualquier máquina para ese usuario dado.)

En Linux, hay otra razón por la que este error puede ocurrir, y es que esté utilizando una versión binaria de MySQL que haya sido compilada con una versión diferente de la librería glibc de la que usted está utilizando. En este caso, usted debería actualizar su sistema operativo o glibc, o descargarse una distribución MySQL en código fuente y compilarla usted mismo. Un paquete RPM de código fuente es, normalmente, trivial de compilar e instalar, así que esto no es un gran problema.

- Si especifica un nombre de máquina cuando se intenta conectar, pero obtiene un mensaje de error donde el nombre de máquina no se muestra, o es un número IP, significa que el servidor MySQL obtuvo un error al intentar resolver el número IP de el cliente a un nombre:

```
shell> mysqladmin -u root -pxxxx -h some-hostname ver
Access denied for user 'root'@'' (using password: YES)
```

Esto indica que existe un problema con DNS. Para arreglarlo, ejecute mysqladmin flush-hosts para reestablecer la cache interna de DNS. Consulte Sección 7.5.6, “Cómo usa MySQL las DNS”.

Estas son algunas soluciones permanentes:

- Intente encontrar qué le ocurre a su servidor DNS y arréglelo.
- Especifique números IP en vez de nombres de máquina en las tablas grant de MySQL
- Introduzca una línea para la máquina cliente en /etc/hosts.
- Ejecute mysqld con la opción --skip-name-resolve.

  - Ejecute mysqld con la opción --skip-host-cache.
  - En Unix, si está ejecutando el servidor y el cliente en la misma máquina, conéctese a localhost. Las conexiones a localhost en Unix utilizan un archivo socket en vez de TCP/IP.
  - En Windows, si está ejecutando cliente y servidor en la misma máquina, y el servidor tiene activada la característica de conexión mediante named pipe, conéctese a . (punto). Las conexiones a . utilizan una named pipe en vez de TCP/IP.

- Si el comando mysql -u root test funciona pero mysql -h *your_hostname* -u root test termina con un Access denied (donde *your_hostname* es el nombre real de su máquina local), puede que no haya especificado correctamente el nombre de su máquina en la tabla user. Un problema común es que el valor de Host en la fila de la tabla user especifique un nombre de máquina incompleto, pero las rutinas de resolución de nombres de dominio de su sistema retornen un nombre de dominio completo (o viceversa). Por ejemplo, si tiene una entrada con valor 'tcx' en Host de la tabla user, pero sus DNS dicen que su nombre de máquina es 'tcx.subnet.se', esta entrada no funcionará. Intente añadir una entrada a la tabla user con un valor de Host que contenga un comodín; por ejemplo 'tcx.%'. ¡En cualquier caso, el uso de nombres que acaben con '%' es *inseguro* y *no* recomendado!

- Si mysql -u *user_name* test funciona pero mysql -u *user_name other_db_name* no funcionado, entonces es que no tiene concedidos los permisos para acceder a la base de datos *other_db_name* para ese usuario dado.

- Si mysql -u *user_name* funciona cuando se ejecuta en la máquina del servidor, pero mysql -h *host_name* -u *user_name* no lo hace cuando se ejecuta en una máquina cliente remota, entonces no tiene activado el acceso a el servidor para el usuario dado en la máquina remota.

- Si no es capaz de averiguar por qué obtiene el error de Access denied, borre de la tabla user todos los valores que contengan comodines (que contengan caracteres '%' o '_'). Un error muy común es insertar un nuevo registro con Host='%' y User='*some_user*', pensando que esto permite especificar que localhost se conecte desde la misma máquina. La razón por la que esto no funciona es que los privilegios por defecto incluyen un registro con Host='localhost' y User=''. Debido a que ese registro es más específico que '%', se utiliza preferentemente frente al nuevo registro cuando conecta desde localhost. El procedimiento correcto es insertar una segunda entrada con Host='localhost' y User='*some_user*', o borrar la entrada con Host='localhost' y User=''. Tras borrar la entrada, recuerde ejecutar una sentencia FLUSH PRIVILEGES para recargar las tablas grant.

- Si obtiene el siguiente error, puede ser que tenga un problema con las tablas db o host:

```
Access to database denied
```

  Si la entrada seleccionada de la tabla db contiene un valor vacio en la columna Host, asegúrese de que hay una o más entradas correspondientes en la tabla host especificando a qué máquinas se aplica la entrada de la tabla db.

- Si usted puede conectar al servidor MySQL, pero obtiene un mensaje Access denied siempre que ejecuta una sentencia SELECT ... INTO OUTFILE o LOAD DATA INFILE, su entrada en la tabla user no tiene el privilegio FILE activado.

- Si cambia las tablas grant directamente (por ejemplo, mediante sentencias INSERT, UPDATE, o DELETE) y sus cambios parecen que son ignorados, recuerde que debe ejecutar la sentencia FLUSH PRIVILEGES statement or el comando mysqladmin flush-privileges para que el servidor relea las tablas de privilegios. En otro caso, los cambios no tendrán efecto hasta la próxima vez que

el servidor sea reiniciado. Recuerde que tras cambiar la clave de `root` con un comando `UPDATE`, no necesita especificar la nueva clave hasta que se haga la relectura de privilegios, porque el servidor no sabrá que la clave ha cambiado hasta ese momento.

- Si sus privilegios parecen haber cambiado en el medio de una sesión, podría ser que un administrador de MySQL los haya cambiado. La recarga de las tablas grant afecta a las nuevas conexiones de cliente, pero también a las ya existentes tal y como se indica en Sección 5.6.7, “Cuándo tienen efecto los camios de privilegios”.

- Si tiene problemas de acceso con un programa Perl, PHP, Python u ODBC, intente conectar al servidor con `mysql -u user_name db_name` o `mysql -u user_name -pyour_pass db_name`. Si puede conectar utilizando el programa cliente `mysql`, el problema está en su programa, no en los privilegios de acceso. (No hay ningún espacio entre `-p` y la clave; también puede utilizar la sintaxis `--password=your_pass` para especificar la clave. Si utiliza la opción `-p` sola, MySQL le preguntará por la clave.)

- Para hacer comprobaciones, inicie el servidor `mysqld` con la opción `--skip-grant-tables`. Puede cambiar las tablas grant y utilizar el script `mysqlaccess` para comprobar que sus modificaciones tienen el efecto deseado. Cuando esté satisfecho con los cambios, ejecute `mysqladmin flush-privileges` para decirle al servidor `mysqld` que comience a utilizar las nuevas tablas grant. (Recargar las tablas sobresee la opción `--skip-grant-tables`. Esto permite decirle al servidor que comience a utilizar las tablas sin tener que reiniciarlo.)

- Si todo lo anterior falla, inicie el servidor `mysqld` con una opción de depuración (por ejemplo, `--debug=d,general,query`). Esto imprime la información de host y usuario sobre los intentos de conexión, así como información sobre cada comando ejecutado. Consulte Sección D.1.2, “Crear ficheros de traza”.

- Si tiene cualquier otro problema con las tablas grant de MySQL y cree que debe enviar el problema a la lista de correo, proporcione siempre un volcado de las tablas grant MySQL. Puede hacer ete volcado con el comando `mysqldump mysql`. Como siempre, envíe su problema utilizando el script `mysqlbug`. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”. En elgunos casos, podría necesitar reiniciar `mysqld` con `--skip-grant-tables` para ejecutar `mysqldump`.

5.6.9. Hashing de contraseñas en MySQL 4.1

Las cuentas de usuario de MySQL se listan en la tabla `user` de la base de datos `mysql` . Cada cuenta MySQL tiene una contraseña asiganada, aunque lo que se guarda en la columna `Password` de la tabla `user` no es una versión en texto plano de la contraseña, si no un valor hash computado a partir de la misma. Los valores hash de las contraseñas se obtienen a partir de la función `PASSWORD()` .

MySQL usa contraseñas en dos fases de la comunicación cliente/servidor:

- Cuando un cliente trata de conectar al servidor, hay un paso inicial de autenticación en el que el cliente debe presentar una contraseña cuyo valor hash coincida con el valor hash almacenado en la tabla `user` para la cuenta que el cliente quiere usar.

- Una vez que el cliente conecta, puede (si tiene los suficientes permisos) cambiar o incializar los hashes de las contraseñas para las cuentas listadas en la tabla `user` . El cliente puede hacerlo mediante la función `PASSWORD()` para generar el hash de la contraseña, o mediante los comandos `GRANT` o `SET PASSWORD`.

En otras palabras, el servidor *usa* los valores hash durante la autenticación cuando un cliente trata de conectar por primera vez. El servidor *genera* valores hash si un cliente conectado invoca la función `PASSWORD()` o usa los comandos `GRANT` o `SET PASSWORD` para inicializar o cambiar una contraseña.

El mecanismo de hash de contraseñas se actualizó en MySQL 4.1. para proporcionar una mejor seguridad y para reducir el riesgo de intercepción de contraseñas. Sin embargo, este nuevo mecanismo sólo lo entienden los servidores y clientes MySQL 4.1. (y versiones posteriores), lo cual puede acarrear algunos problemas de compatibilidad. Un cliente 4.1. o posterior puede conectar a un servidor pre-4.1, ya que el cliente entiende los mecanismos de hashing de contraseñas antiguos y nuevos. Sin embargo, un cliente pre-4.1. que trate de conectar a un servidor 4.1. o posterior puede tener problemas. Por ejemplo, un cliente 3.23 `mysql` que trate de conectar a un servidor 5.0 puede fallar con el siguiente mensaje de error:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

Otro ejemplo común es cuando se trata de usar la antigua extensión de **mysql** para PHP tras actualizar a MySQL 4.1 o posterior. (Consulte Sección 24.3.1, “Problemas comunes con MySQL y PHP”.)

La siguiente discusión describe las diferencias entre el antiguo y nuevo mecanismo de contraseñas, y qué debe hacer si actualiza su servidor pero necesita matener compatibilidad con clientes versión pre-4.1. Puede encontrar información adicional en Sección A.2.3, “`Client does not support authentication protocol`”. Esta información es de especial importancia para programadores de PHP que migran de versiones de bases de datos MySQL 4.0 o anteriores a versiones 4.1. o posteriores.

Nota: Esta discusión contrasta el comportamiento 4.1. con el pre-4.1, pero el comportamiento 4.1 descrito aquí realmente empieza en el 4.1.1. MySQL 4.1.0 es una versión ``particular'' ya que tiene mecanismos ligeramente distintos a los implementados en 4.1.1 y posteriormente. Las diferencias entre 4.1.0 y reciones más recientes se describen con más detalle en Manual de referencia de MySQL 4.1.

Antes de MySQL 4.1, los hashes de contraseñas computados por la función `PASSWORD()` tienen una longitud de 16 bytes. Tales hashes tienen este aspecto:

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

La columna `Password` de la tabla `user` (en la que se guardan los hashes) también tiene una longitud de 16 bytes antes de MySQL 4.1.

En MySQL 4.1, la función `PASSWORD()` se modificó para producir un valor hash más largo de 41-bytes:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *43c8aa34cdc98eddd3de1fe9a9c2c2a9f92bb2098d75 |
+-------------------------------------------+
```

Por consiguiene, la columna `Password` en la tabla `user` debe tener una longitud de 41 bytes para almacenar estos valores:

- Si realiza una nueva instalación de MySQL 5.0, la columna `Password` se amplía a 41 bytes automáticamente.
- Actualizar desde MySQL 4.1 (4.1.1 o posterior en la serie 4.1) a MySQL 5.0 no debería afectar a nada de todo esto, ya que ambas versiones usan el mismo mecanismo de hash de contraseñas. Si desea actualizar una versión antorior de MySQL a 5.0, debe actualizar primero a la versión 4.1, y luego actualizar la instalación de 4.1. a 5.0.

Una columna `Password` más amplia puede almacenar hashes de contraseñas en el antiguo y nuevo formato. El formato de cualquier valor hash de una contraseña puede determinarse de dos formas:

- La diferencia óbvia es la longitud (16 bytes contra 41 bytes).
- Una segunda diferencia es que los hashes de contraseñas en el nuevo formato simpre empieza con un carácter '`*`', mientras que una contraseña en el antiguo formato nunca lo hace.

El hash de la contraseña más larga tiene mejores propiedades criptográficas, y la autenticación de clientes basada en hashes largos es más segura que la basada en los antiguos hashes cortos.

Las diferencias entre hashes cortos y largos son relevantes para cómo el servidor usa las contraseñas durante la autenticación y por cómo genera los hashes de contraseñas para clientes conectados que realizan operaciones de cambio de contraseña.

La forma en que el servidor usa los hashes de contraseñas durante la autenticación se ve afectada por la longitud de la columna `Password` :

- Si la columna es corta, sólo se usa autenticación de hash cortos.
- Si la columna es larga, puede soportar hashes cortos o largos, y el servidor puede usar cualquier formato:
  - Clientes pre-4.1 pueden conectar, aunque sólo conocen el antiguo mecanismo de hash, pueden autenticar sólo para cuentas que tengan hashes cortos.
  - Clientes 4.1 y posterior pueden autenticar para cuentas que tengan hashes cortos o largos.

Para cuentas con hash corto, el proceso de autenticación es un poco más seguro para clientes 4.1 y posteriores que para clientes más antiguos. Respecto a seguridad, el gradienet de menos a más seguro es:

- Cliente pre-4.1 autenticando con hash de contraseña corto
- Cliente 4.1 o posterior autenticando con hash de contraseña corto.
- Cliente 4.1 o posterior autenticando con hash de contraseña largo.

La forma en que el servidor genera los hashes de contraseña para clientes conectados se ve afectado por la longitud de la columna `Password` y por la opción `--old-passwords`. Un servidor 4.1. o posterior genera hashes largos sólo si se cumplen ciertas condiciones: La columna `Password` debe ser lo suficientemente larga para guardar valores largos y no debe darse la opción `--old-passwords`. Estas condiciones se aplican como sigue:

- La columna `Password` debe ser lo suficientemente grande para guardar hashes largos (41 bytes). Si la columna no se ha actualizado y todavía tiene la longitud pre-4.1 de 16 bytes, el servidor entiende que no puede guardar hashes largos y genere sólo hashes cortos cuando un cliente realiza opraciones de cambio de contraseña mediante `PASSWORD()`, `GRANT`, o `SET PASSWORD`. Este es el comportamiento que ocurre si ha actualizado a 4.1 pero no ha ejecutado todavía el script `mysql_fix_privilege_tables` para ensanchar la columna `Password` .
- Si la columna `Password` es amplia, puede almacenar tanto hashes de contraseñas largos como cortos. En este caso, `PASSWORD()`, `GRANT`, y `SET PASSWORD` generan hashes largos a no ser que el servidor se haya iniciado con la opción `--old-passwords`. Este opción fuerza al servidor a generar hashes de contraseñas cortos.

El propósito de la opción `--old-passwords` es permitirle mantener compatibilidad con clientes anteriores a 4.1 bajo circunstancias donde el servidor generaría hashes de contraseñas largos. La opción no afecta la autenticación (clientes 4.1. y posteriores pueden usar cuentas que tengan hashes largos de

contraseña), pero no evita la creación de hashes largos de contraseñas en la tabla `user` como resultado de una operación de cambio de contraseña. Si eso ocurre, la cuenta no puede usarse por clientes pre-4.1. Sin la opción `--old-passwords`, es posible el siguiente escenario no deseable:

- Un cliente viejo trata de conectar a una cuenta que tenga hash de contraseña corto.
- El cliente cambia su propia contraseña. Sin `--old-passwords`, esto acaba con la cuenta con un hash de contraseña largo.
- La siguiente vez que el viejo cliente trate de conectar a la cuenta, no podrá, ya que la cuenta tiene un hash de contraseña largo que requiere un nuevo mecanismo de hash durante la autenticación. (Una vez que una cuenta tiene un hash de contraseña largo en la tabla de usuario, sólo clientes 4.1. y posteriores pueden autenticar, ya que clientes pre-4.1. no entienden hashes largos.)

Este escenario ilustra que, si debe soportar clientes pre-4.1, es peligroso ejecutar un servidor 4.1 o posterior sin usar la opción `--old-passwords` . Ejecutando el servidor con `--old-passwords`, las operaciones de cambio de contraseña no generan hashes largos de contraseña y por lo tanto, no provocan que las cuentas sean inaccesibles para clientes antiguos. (Aquellos clientes no pueden bloquearse a ellos mismos mediante el cambio de su contraseña y acabando con un hash de contraseña largo.)

La desventaja de la opción `--old-passwords` es que cualquier contraseña que cree o cambie usará hashes cortos, incluso para clientes 4.1. Así, pierde la seguridad adicional proporcionada por los hashes de contraseña largos. Si quiere crear una cuenta que tiene un hash largo (por ejemplo, para usar con un cliente 4.1), debe hacerlo mientras el servidor se esté ejecutando sin `--old-passwords`.

Los siguientes escenarios son posibles al ejecutar un servidor 4.1 o posterior, incluyendo servidores MySQL 5.0:

Escenario 1: Columna `Password` corta en tabla de usuario:

- Sólo se pueden guardar hashes cortos en la columna `Password` .
- El servidor usa sólo hashes cortos durante la autenticación del cliente.
- Para clientes conectados, las operaciones de generación de hashes de contraseñas mediante `PASSWORD()`, `GRANT`, o `SET PASSWORD` usan hashes cortos exclusivamente. Cualquier cambio a una contraseña de una cuenta resulta en una cuenta teniendo un hash de contraseña corto.
- La opción `--old-passwords` puede usarse pero es supérflua ya que con una columna `Password` corta, el servidor genera sólo hashes de contraseña cortos de todas formas..

Escenario 2: Columna `Password` larga; servidor no arrancado con la opción `--old-passwords` :

- Hashes cortos o largos pueden almacenarse en la columna `Password` .
- Clientes 4.1 y posteriores (incluyendo clientes 5.0) pueden autenticar para cuentas que tengan tanto hashes cortos como largos.
- Clientes pre-4.1 pueden autenticar sólo para cuentas que tengan hashes cortos.
- Para clientes conectados, operaciones generadoras de hash como `PASSWORD()`, `GRANT`, o `SET PASSWORD` usan hashes largos exclusivamente. Un cambio en la contraseña de una cuenta resulta en dicha cuenta con un hash largo.

Como se ha indicado, un peligro en este escenario es que es posible para cuentas con hash corto quedar inaccesibles para clientes pre-4.1. Un cambio en tales contraseñas hecho via `GRANT`, `PASSWORD()`, o `SET PASSWORD` resulta en la cuenta con una contraseña larga. A partir de ahí, ningún cliente pre-4.1 puede autenticar a dicha cuenta hasta que el cliente actualice a 4.1.

Para tratar este problema, puede cambiar una contraseña de forma especial. Por ejemplo, normalmente usa `SET PASSWORD` como sigue para cambiar una contraseña de cuenta:

```
mysql> SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

Para cambiar la contraseña pero crear un hash corto, use la función `OLD_PASSWORD()` en su lugar:

```
mysql> SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` es útil para situaciones en que explícitamente quiera generar un hash corto.

Escenario 3: Columna `Password` larga; servidor 4.1 o posterior arrancado con la opción `--old-passwords`:

- Hashes crotos o largos pueden guardarse en la columna `Password`.
- Clientes 4.1 y posteriores pueden autenticar para cuentas que tengan hashes cortos o largos (pero tenga en cuenta que es posible crear hashes largos sólo cuando el servidor se arranca sin `--old-passwords`).
- Clientes pre-4.1 clients pueden autenticar sólo para cuentas que tengan hashes cortos.
- Para clientes conectados, operaciones de generación de hashes como `PASSWORD()`, `GRANT`, o `SET PASSWORD` usan hashes cortos exclusivamente. Cualquier cambio en la contraseña de una cuenta resulta en que dicha cuenta tenga un hash de contraseña corto.

En este escenario, no puede crear cuentas que tengan hashes de contraseña cortos, ya que la opción `--old-passwords` evita la generación de hashes largos. También, si crea una cuenta con hashes largos antes de usar la opción `--old-passwords`, cambiar la contraseña de la cuenta mientras `--old-passwords` está en efecto resulta en la cuenta teniendo una contraseña corta, causando que se pierdan los beneficios de seguridad de un hash más largo.

Las desventajas de estos escenarios pueden resumirse así:

En el escenario 1, no puede beneficiarse de hashes largos que proporcionan más seguridad de autenticación.

En el escenario 2, las cuentas con hashes cortos son inaccesibles para clientes pre-4.1 si cambia sus contraseñas sin usar explícitamente `OLD_PASSWORD()`.

En el escenario 3, `--old-passwords` evita que las cuentas con hashes cortos sean inaccesibles, pero las operaciones de cambio de contraseña causa que las cuentas con hashes largos vuelvan a ser hashes cortos, y no puede volver a hacerlos hashes largos mientras `--old-passwords` tenga efecto.

5.6.9.1. Implicación del cambio en el hashing de contraseñas en aplicativos

Una actualización a MySQL 4.1 o posterior puede provocar problemas de compatibilidad para aplicaciones que usen `PASSWORD()` para generar contraseñas para sus propios propósitos. Las aplicaciones no deben hacer esto, ya que `PASSWORD()` debe usarse sólo para administrar contraseñas para cuentas MySQL. Pero algunas aplicaciones usan `PASSWORD()` para sus propios propósitos de todas formas.

Si actualiza a 4.1 o posterior desde versiones pre-4.1 de MySQL y ejecuta el servidor bajo condiciones donde genera hashes largos de contraseñas, una apliación usando `PASSWORD()` para sus propias contraseñas falla. El curso recomendado de ación en tales casos es modificar la aplicación para usar otra función, tal como `SHA1()` o `MD5()`, para producir hashes de valores. Si esto no es posible, puede usar la función `OLD_PASSWORD()` , que se proporciona para generar hashes cortos en el viejo formato. Sin

embargo, debe tener en cuenta que `OLD_PASSWORD()` puede dejar de ser soportado en alguna futura versión.

Si el servidor está ejecutándose bajo circunstancias donde genera hashes cortos, `OLD_PASSWORD()` está disponible pero es equivalente a `PASSWORD()`.

Los programadores de PHP migrando sus bases de datos MySQL de la versión 4.0 o anteriores a la versión 4.1 o posterior deben consultar Old Client.

5.7. Gestión de la cuenta de usuario MySQL

Esta sección describe cómo preparar cuentas para clientes en su servidor MySQL. Se discuten los siguientes tópicos:

- El significado de los nombres de cuenta y contraseñas usados en MySQL y cómo se compara con los nombres y contraseñas usadas por su sistema operativo.
- Cómo preparar una nueva cuenta y borrar una existente
- Cómo cambiar contraseñas
- Guías para usar contraseñas de forma segura
- Cómo usar conexiones seguras mediante SSL

5.7.1. Nombres de usuario y contraseñas de MySQL

Una cuenta MySQL se define en términos de un nombre de usuario y el equipo o equipos desde los que el usuario puede conectar al servidor. La cuenta también tiene una contraseña. Hay varias diferencias entre cómo se usan los nombres de usuario y contraseñas en MySQL y cómo los usa el sistema operativo:

- Los nombres de usuario, tal como los usa MySQL para autentificación, no tienen nada que ver con los nombres de usuario (nombres de logueo) tal y como los usa Windows o Unix. En Unix, la mayoría de clientes MySQL por defecto tratan de loguear usando el nombre de usuario Unix como el nombre de usuario MySQL, pero eso es sólo como conveniencia. El comportamiento por defecto puede cambiarse fácilmente, ya que el programa clienet permite especificar cualquier nombre de usuario con la opción `-u` o `--user` . Como esto significa que cualquiera puede intentar conectar al servidor usando cualquier nombre de usuario, no puede hacer una base de datos segura de ninguna forma a no ser que todas las cuentas MySQL tengan contraseña. Cualquiera que especifique un nombre de usuario para una cuenta que no tenga contraseña puede conectar al servidor.
- Nombre de usuarios en MySQL pueden tener como máximo 16 caracteres de longitudo. Este límite está hard-codeado en los servidores y clientes MySQL, y tratar de evitarlo mediante la modificación de las tablas en la base de datos `mysql` *no funciona* .

  Nota: *Nunca debe alterar ninguna de las tablas en la base de datos `mysql` de ninguna forma excepto mediante la ejecución de los scripts proporcionados expresamente para este propósito con la distribución MySQL. Tratar de redefinir las tablas de sistema MySQL de cualquier otra forma da como resultado un comportamiento indefinido (y no soportado)*.

  Nombres de usuario en el sistema operativo están completamente desligados de los nombres de usuario de MySQL y pueden tener longitud máxima diferente. Por ejemplo, los nombres de usuario Unix típicamente están limitados a 8 caracteres.
- Las contraseñas MySQL no tienen nada que ver con las contraseñas para loguear en el sistema operativo. No hay una conexión necesaria entre la contraseña que usa para entrar en una máquina Windows o Unix y la contraseña usada para acceder al servidor MySQL en esa máquina.

- MySQL cifra contraseñas usando su propio algoritmo. Este cifrado es diferente del usado durante el proceso de logueo de Unix. El cifrado de contraseña es el mismo que el implementado en la función `PASSWORD()` . El cifrado de contraseñas Unix es el mismo que el implementado por la función SQL `ENCRYPT()` . Consulte la descripción de las funciones `PASSWORD()` y `ENCRYPT()` en Sección 12.9.2, “Funciones de cifrado”. Desde la versión 4.1, MySQL usa un método más fuerte de autenticación que tiene una mejor protección de contraseña durante el proceso de conexión que en versiones anteriores. Es seguro incluso si los paquetes TCP/IP se esnifan o la base de datos `mysql` se captura. (En versiones anteriores, incluso aunque las contraseñas se guardan cifradas en la tabla `user`, se podía usar conocimiento de la contraseña cifrada para conectar al servidor MySQL.)

Cuando instala MySQL, las tablas de permisos se inicializan con un conjunto inicial de cuentas. Estas cuentas tienen nombres y privilegios de acceso descritos en Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”, que discute cómo asignarles contraseñas. Así mismo, normalmente inicialice, modifique y borre cuentas mediante los comandos `GRANT` y `REVOKE`. Consulte Sección 13.5.1.3, “Sintaxis de `GRANT` y `REVOKE`”.

Cuando conecta a un servidor MySQL con un cliente de líneas de comando, puede especificar el nombre de usuario y contraseña para la cuenta que desea usar:

```
shell> mysql --user=monty --password=guess db_name
```

Si prefiere opciones cortas, el comando es así:

```
shell> mysql -u monty -pguess db_name
```

No deben haber *espacios* entre la opción `-p` y el valor de contraseña a continuación. Consulte Sección 5.6.4, “Conectarse al servidor MySQL”.

El comando precedente incluye el valor de la contraseña en la línea de comando, lo que puede ser un riesgo de seguridad. Consulte Sección 5.7.6, “Guardar una contraseña de forma segura”. Para evitarlo, especifique la opción `--password` o `-p` sin ningún valor de contraseña:

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

A continuación, el programa cliente muestra un prompt y espera a que introduzca la contraseña. (En estos ejemplos, *`db_name` no* se interpreta como contraseña, ya que está separado de la precedente opción de contraseña con un espacio.)

En algunos sistemas, la llamada que MySQL usa para pedir una contraseña automáticamente limita la contraseña a ocho caracteres. Este es un problema con la librería de sistema, no con MySQL. Internamente, MySQL no tienen ningún límite para la longitud de la contraseña. Para solventar este problema, cambie su contraseña MySQL a un valor que tenga ocho o menos caracteres, o ponga su contraseña en un fichero de opciones.

5.7.2. Añadir nuevas cuentas de usuario a MySQL

Puede crear cuentas MySQL de dos formas:

- Usando comandos `GRANT`
- Manipulando las tablas de permisos MySQL directamente

El método preferido es usar comandos `GRANT` , ya que son más concisos y menos propenso a errores. . `GRANT` está disponible desde MySQL 3.22.11; su sintaxis se describe en Sección 13.5.1.3, “Sintaxis de `GRANT` y `REVOKE`”.

Otra opción para crear cuentas es usar uno de los diversos programas proporcionados por terceras partes que ofrecen capacidades para administradores de MySQL. `phpMyAdmin` es una de ellos.

Los siguientes ejemplos muestran cómo usar el programa cliente `mysql` para añadir nuevos usuarios. Estos ejemplos asumen que los permisos se inicializan según las pautas descritas en Sección 2.9.3, “Hacer seguras las cuentas iniciales de MySQL”. Esto significa que para realizar cambios, debe conectar al servidor MySQL como el usuario `root` , y la cuenta `root` debe tener el privilegio `INSERT` para la base de datos `mysql` y el permiso administrativo `RELOAD`.

En primer lugar, use el programa `mysql` para conectar al servidor como el usuario `root` :

```
shell> mysql --user=root mysql
```

Si ha asignado una contraseña a la cuenta `root`, necesitará la opción `--password` o `-p` para este comando `mysql` y también para los mostrados a continuación en esta sección.

Tras la conexión al servidor como `root`, puede añadir nuevas cuentas. El siguiente comando usa `GRANT` para inicializar nuevas cuentas:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> GRANT USAGE ON *.* TO 'dummy'@'localhost';
```

Las cuentas creadas con estos comandos `GRANT` tienen las siguientes propiedades:

- Dos de las cuentas tienen un nombre de usuario de `monty` y una contraseña de `some_pass`. Ambas cuentas son cuentas de superusuario con plenos permisos para hacer cualquier cosa. Una cuenta (`'monty'@'localhost'`) puede usarse sólo cuando se conecte desde el equipo local. La otra (`'monty'@'%'`) puede usarse para conectarse desde cualquier otro equipo. Note que es necesario tener ambas cuentas para que `monty` sea capaz de conectarse desde cualquier sitio como `monty`. Sin la cuenta `localhost`, la cuenta anónima para `localhost` creada por `mysql_install_db` tendría precedencia cuando `monty` conecte desde el equipo local. Como resultado, `monty` se trataría como un usuario anónimo. La razón para ello es que el usuario anónimo tiene un valor más específico en la columna `Host` que la cuenta `'monty'@'%'` y por lo tanto toma precedencia en la ordenación de la tabla `user`. (La ordenación de la tabla `user` se discute en Sección 5.6.5, “Control de acceso, nivel 1: Comprobación de la conexión”.)
- Una cuenta tiene un nombre de usuario de `admin` y no tiene contraseña. Esta cuenta puede usarse sólo desde el equipo local. Tiene los privilegios administrativos `RELOAD` y `PROCESS` . Éstos permiten al usuario `admin` ejecutar los comandos `mysqladmin reload`, `mysqladmin refresh`, y `mysqladmin flush-xxx` , así como `mysqladmin processlist` . No se dan permisos para acceder a ninguna base de datos. Puede añadir tal privilegio posteriormente mediante un comando `GRANT` adicional.
- Una cuenta tiene un nombre de usuario de `dummy` sin contraseña. Esta cuenta puede usarse sólo desde el equipo local. No tiene ningún privilegio. El permiso `USAGE` en el comando `GRANT` permite crear una cuenta sin darle ningún privilegio. Tiene el efecto de inicializar todos los privilegios globales a `'N'`. Se asume que se otorgarán privilegios específicos posteriormente.

Como alternativa a `GRANT`, puede crear la misma cuenta directamente mediante comandos `INSERT` y después diciendo al servidor que recargue las tablas de permisos usando `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');
mysql> INSERT INTO user SET Host='localhost',User='admin',
    ->     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

La razón de usar `FLUSH PRIVILEGES` al crear cuantas con `INSERT` es decir al servidor que vuelva a leer las tablas de permisos. De otro modo, los cambios no se tienen en cuenta hasta que se reinicie el servidor. Con `GRANT`, `FLUSH PRIVILEGES` no es necesario.

La razón para usar la función `PASSWORD()` con `INSERT` es cifrar las contraseñas. El comando `GRANT` cifra la contraseña, así que `PASSWORD()` no es necesario.

El valor `'Y'` activa permisos para las cuentas. Para la cuenta `admin` , puede emplear la sintaxis más clara extendida `INSERT` usando `SET`.

En el comando `INSERT` para la cuenta `dummy` account, sólo las columnas `Host`, `User`, y `Password` en el registro de la tabla `user` tienen valores asignados. Ninguna de las columnas de permisos se asignan explícitamente, así que MySQL les asigna a todas el valor por defecto de `'N'`. Esto es equivalente al funcionamiento de `GRANT USAGE`.

Para inicializar una cuenta de super usuario, sólo es necesario crear una entrada en la tabla `user` con las columnas de permisos inicializadas a `'Y'`. Los privilegios de la tabla `user` son globales, así que no se necesitan registros en ninguna de las otras tablas de permisos.

Los siguientes ejemplos crean tres cuentas y les dan acceso a bases de datos específicas. Cada una de ellas tiene un nombre de usuario `custom` y contraseña `obscure`.

Para crear las cuentas con `GRANT`, use los siguientes comandos:

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON bankaccount.*
    ->     TO 'custom'@'localhost'
    ->     IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON expenses.*
    ->     TO 'custom'@'whitehouse.gov'
    ->     IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON customer.*
    ->     TO 'custom'@'server.domain'
    ->     IDENTIFIED BY 'obscure';
```

Las tres cuentas pueden usarse de la siguiente manera:

- La primera cuenta puede acceder a la base de datos `bankaccount`, pero sólo desde el equipo local.
- La segunda cuenta puede acceder la base de datos `expenses`, pero sólo desde el equipo `whitehouse.gov`.
- La tercera cuenta puede acceder la base de datos `customer`, pero sólo desde el equipo `server.domain`.

Para inicializar las cuentas `custom` sin usar `GRANT`, use los comandos `INSERT` como se explica para modificar las tablas de permisos directamente:

```
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('whitehouse.gov','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('server.domain','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
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('whitehouse.gov','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('server.domain','customer','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

Los primeros tres comandos `INSERT` añaden registros en la tabla `user` que permiten al usuario `custom` conectar desde los equipos con la contraseña dada, pero no otorga privilegios blobales (todos los privilegios se inicializan al valor por defecto `'N'`). Los siguientes tres comandos `INSERT` añaden registros en la tabla `db` que otorgan privilegios a `custom` para las bases de datos `bankaccount`, `expenses`, y `customer`, pero sólo cuando se accede desde los equipos apropiados. Como siempre, cuando modifique las tablas de permisos directamente, debe decirle al servidor que las recargue con `FLUSH PRIVILEGES` para que los cambios en los permisos tengan efecto.

Si quiere dar a un usuario específico acceso desde todas las máquinas dentro de un dominio dado (por ejemplo, `mydomain.com`), puede realizar un comando `GRANT` que use el carácter comodín `'%'` en la parte del equipo del nombre de cuenta:

```
mysql> GRANT ...
    ->     ON *.*
    ->     TO 'myname'@'%.mydomain.com'
    ->     IDENTIFIED BY 'mypass';
```

Para hacer lo mismo modificando las tablas de permisos directamente, haga lo siguiente:

```
mysql> INSERT INTO user (Host,User,Password,...)
    ->     VALUES('%.mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

5.7.3. Eliminar cuentas de usuario de MySQL

Para eliminar una cuenta, use el comando `DROP USER` , descrito en Sección 13.5.1.2, “Sintaxis de `DROP USER`”.

5.7.4. Limitar recursos de cuentas

Una forma de limitar los recursos de los servidores MySQL es asignar a la variable de sistema `max_user_connections` un valor distinto de cero. Sin embargo,este método es estrictamente global, y no está permitido para la administración de cuentas individuales. Además, limita sólo el número de conexiones simultáneas hechas usando una sóla cuenta, y no lo que un cliente puede hacer una

vez conectado. Ambos tipos de control son interesantes para muchos administradores de MySQL, particularmente aquéllos que trabajan en ISPs.

En MySQL 5.0, puede limitar los siguientes recursos de servidor para cuentas individuales:

- El número de consultas que una cuenta puede realizar por hora
- El número de actualizaciones que una cuenta puede hacer por hora
- El número de veces que una cuenta puede conectar con el servidor por hora

Cualquier comando que un cliente puede realizar cuenta en el límite de consultas. Sólo los comandos que modifiquen la base de datos o las tablas cuentan en el límite de actualizaciones.

Desde MySQL 5.0.3, es posible limitar el número de conexiones simultáneas al servidor por cuenta.

Una cuenta en este contexto es un registro en la tabla `user` . Cada cuenta se identifica unívocamente por los valores de las columnas `User` y `Host`.

Como prerrequisito para usar esta característica, la tabla `user` en la base de datos `mysql` debe contener las columnas relacionadas con el recurso. Los límites de recursos se guardan en las columnas `max_questions`, `max_updates`, `max_connections`, y `max_user_connections` . Si su tabla `user` no tiene estas columnas, debe actualizarla; consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Para cambiar el límite de recursos con un comando `GRANT` use la cláusula `WITH` que nombra cada recurso a ser limitado y un contador por hora indicando el valor límite. Por ejemplo, para crear una nueva cuenta que pueda acceder a la base de datos `customer`, pero sólo de forma limitada, utilice este comando:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     IDENTIFIED BY 'frank'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

No todos los tipos de límites necesitan nombrarse en la cláusula `WITH`, pero los nombrados pueden presentarse en cualquier orden. El valor para cada límite por hora debe ser un entero representando el contador por hora. Si el comando `GRANT` no tiene cláusula `WITH`, los límites se inicializan con el valor por defecto de cero (o sea, sin límite). Para `MAX_USER_CONNECTIONS`, el límite es un entero indicando el máximo número de conexiones simultáneas que la cuenta puede hacer en cualquier momento. Si el límite asignado es el valor por defecto de cero, la variable de sistema `max_user_connections` determina el número de conexiones simultáneas para la cuenta.

Para inicializar o cambiar los límites de una cuenta existente, use un comando `GRANT USAGE` a nivel global (`ON *.*`). El siguiente comando cambia el límite de consultas para `francis` a 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

Este comando deja los permisos existentes de la cuenta inalterados y modifica sólo los valores cuyo límite se especifica.

Para eliminar un límite existente, ponga su valor a cero. Por ejemplo, para eliminar el límite de cuántas veces por hora puede conectar `francis` , use este comando:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

El conteo del uso de recursos toma lugar cuando una cuenta tiene un límite distinto a cero para el uso de cualquier recurso.

Mientras el servidor está en ejecución, cuenta el número de veces que cada cuenta usa los recursos. Si una cuenta llega a su límite en el número de conexiones en la última hora, se rechazan cualquier intento de conexión mientras dure la hora. De forma similar, si la cuenta llega a su límite de consultas o actualizaciones, consultas o actualizaciones adicionales se rechazan mientras dure la hora. En cualquier caso, se muestra el mensaje de error apropiado

El conteo de recursos se hace por cuenta, no por cliente. Por ejemplo, si una cuenta tiene un límite de 50 consultas, no puede incrementar el límite a 100 haciendo dos conexiones simultáneas al servidor. Las consultas de ambas conexiones se cuentan juntas.

El contador actual por hora de uso de recursos puede reiniciarse globalmente para todas las cuentas, o individualmente para una cuenta dada:

- Para reiniciar los contadores actuales a cero para todas las cuentas, ejecute el comando `FLUSH USER_RESOURCES`. Los contadores también pueden reiniciarse recargando las tablas de permisos (por ejemplo,k con un comando `FLUSH PRIVILEGES` o `mysqladmin reload`).
- Los contadores para una cuenta individual pueden ponerse a cero cambiando cualquiera de sus límites. Para hacerlo, use `GRANT USAGE` como se ha descrito anteriormente y especifique un valor límtite igual al valor que tiene la cuenta en ese momento.

Los reinicios de contadores no afectan el límite `MAX_USER_CONNECTIONS` .

Todos los contadores empiezan a cero cuando el servidor arranca; los contadores no se guardan al reiniciar.

5.7.5. Asignar contraseñas a cuentas

Se pueden asignar contraseñas desde la línea de comandos usando el comando `mysqladmin` :

```
shell> mysqladmin -u nombres_usuario -h equipo password "nuevacontr"
```

La cuenta para la que este comando cambia la contraseña es la que tiene un registro en la tabla `user` que coincida el *`user_name`* con la columna `User` y un equipo cliente *desde el que se conecta* en la columna `Host`.

Otra forma de asignar una contraseña en una cuenta es con el comando `SET PASSWORD` :

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

Sólo los usuarios tales como `root` con acceso de modificación para la base de datos `mysql` puede cambiar la contraseña de otro usuario. Si no está conectado como un usuario anónimo, puede cambiar su propia contraseña omitiendo la cláusula `FOR` :

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

Puede usar el comando `GRANT USAGE` globalmente (`ON *.*`) para asignar una contraseña a una cuenta sin afectar los permisos actuales de la cuenta:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

Aunque generalmente es peferible asignar contraseñas usando uno de los métodos precedentes, puede hacerlo modificando la tabla `user` directamente:

- Para establecer una contraseña al crear una nueva cuenta, especifique un valor para la columna `Password`:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

- Para cambiar la contraseña en una cuenta existente, use `UPDATE` para especificar el valor de la columna `Password`:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('bagel')
    -> WHERE Host = '%' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```

Cuando especifique una contraseña en una cuenta mediante `SET PASSWORD`, `INSERT`, o `UPDATE`, debe usar la función `PASSWORD()` para cifrarla. (La única excepción es que no necesita usar `PASSWORD()` si la contraseña está vacía). `PASSWORD()` es necesario ya que la tabla `user` guarda las contraseñas cifradas, no en texto plano. Si olvida este hecho, es posible que guarde contraseñas así:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

El resultado es que el valor literal `'biscuit'` se guarda como contraseña en la tabla `user`, no el valor cifrado. Cuando `jeffrey` trate de conectar al servidor usando esta contraseña, el valor se cifra y se compara con el valor guardado en la tabla `user`. Sin embargo, el valor guardado es la cadena de caracteres literal `'biscuit'`, así que la comparación falla y el servidor rechaza la conexión:

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

Si inicializa la contraseña usando el comando `GRANT ... IDENTIFIED BY` o `mysqladmin password`, ambos cifran la contraseña. En estos casos, el uso de la función `PASSWORD()` no es necesario.

Nota: El cifrado de `PASSWORD()` es diferente al de contraseñas Unix. Consulte Sección 5.7.1, “Nombres de usuario y contraseñas de MySQL”.

5.7.6. Guardar una contraseña de forma segura

A nivel administrativo, nunca debería otorgar acceso para la tabla `mysql.user` a ninguna cuenta no administrativa.

Cuando ejecuta un programa cliente para conectarse al servidor MySQL, es desaconsejable especificar la clave de manera que sea visible o posible de averiguar para otros usuarios. Los métodos que debe utilizar para especificar su clave cuando ejecuta programas clientes son enumerados aquí, junto con una valoración de los riesgos de cada método:

- Utilice la opción `-p`*`your_pass`* o `--password=`*`your_pass`* en la línea de comandos. Por ejemplo:

```
shell> mysql -u francis -pfrank db_name
```

  Esto es conveniente pero inseguro, porque la clave se vuelve visible para programas de consulta del estado del sistema, como `ps` que pueden ser invocados por otros usuarios para mostrar líneas de

comando. Los clientes de MySQL normalmente sobreescriben el parámetro de la clave en la línea de comandos con ceros durante la secuencia de inicialización, pero aún así hay un breve intervalo de tiempo en que el valor es visible.

- Utilice la opción `-p` o `--password` sin especificar ningún valor para la clave. En este caso, el programa cliente solicita la clave desde el terminal:

```
shell> mysql -u francis -p db_name
Enter password: ********
```

Los caracteres '`*`' indican donde se introduce la clave. La clave no se muestra mientras se está introduciendo.

Es más seguro introducir la clave de esta manera que especificarla en la línea de comandos, porque así no es visible a otros usuarios. No obstante, este método es tan solo aplicable para programas que se ejecutan de manera interactiva. Si quiere invocar un programa cliente desde un script que no se ejecute interactivamente, no hay oportunidad de introducir la clave mediante el terminal. En algunos sistemas, incluso podría darse que la primera línea del script sea leida e interpretada (incorrectamente) como la clave.

- Almacene su clave en un archivo de opciones. Por ejemplo, en Unix puede introducir su clave en la sección `[client]` del archivo `.my.cnf` de su directorio personal.

```
[client]
password=your_pass
```

Si almacena su calve en `.my.cnf`, el archivo no debería ser accesible para nadie más que usted. Para asegurarse de esto, establezca el modo de acceso del archivo a `400` o `600`. Por ejemplo:

```
shell> chmod 600 .my.cnf
```

Sección 4.3.2, “Usar ficheros de opciones” habla sobre los archivos de opciones con más detalle.

- Almacene su clave en la variable de entorno `MYSQL_PWD`. Este método de especificar su clave MySQL debe ser considerado extremadamente inseguro y no debería ser utilizado. Algunas versiones de `ps` incluyen una opción para mostrar las varialbes de entorno de los procesos en ejecución. Si establece `MYSQL_PWD`, su clave estará expuesta a cualquier otro usuario que ejecute `ps`. Aún en sistemas con una versión tal de `ps`, no es inteligente asumir que no habrá cualquier otro método mediante el cual los usuarios puedan examinar las variables de entorno. Consulte Apéndice E, *Variables de entorno*.

De todas maneras, la manera más segura de hacerlo es, o bien hacer que el programa cliente pregunte por la clave, o especificarla en un archivo de opciones protegido.

5.7.7. Usar conexiones seguras

MySQL 5.0 incluye soporte para conexiones seguras (cifradas) entre los clientes MySQL y el servidor, utilizando el protocolo SSL (Secure Sockets Layer). Esta sección explica como utilizar conexiones SSL. También explica una manera de configurar SSH en Windows.

La configuración de MySQL tiene la misión de ser tan rápida como sea posible, así que no se usan las conexiones cifradas por defecto. Hacerlo, haría que el protocolo cliente/servidor fuese mucho más lento. Cifrar datos es una operación que requiere un uso intensivo de CPU, y por tanto obliga a la máquina a realizar trabajo adicional que retrasa otras tareas de MySQL. Para aplicaciones que requieran la seguridad que proveen las conexiones cifradas, el trabajo de computación extra está justificado.

MySQL permite que el cifrado sea activado para conexiones individuales. Puede escoger entre una conexión normal sin cifrar, o una segura cifrada mediante SSL dependiendo de los requerimientos de las aplicaciones individuales.

5.7.7.1. Conceptos básicos de SSL

Para entender como MySQL utiliza SSL, es necesario explicar algunos conceptos básicos sobre SSL y X509. Aquellos ya familiarizados con ellos, pueden saltarse esta parte.

Por defecto, MySQL utiliza conexiones sin cifrar entre el cliente y el servidor. Esto significa que cualquiera con acceso a la red podría ver el tráfico y mirar los datos que están siendo enviados o recibidos. Incluso podría cambiar los datos mientras están aún en tránsito entre el cliente y el servidor. Para mejor la seguridad un poco, puede comprimir el tráfico entre el cliente y el servidor utilizando la opción `--compress` cuando ejecute programas cliente. No obstante, esto parará a un atacante con determinación.

Cuando necesita mover información sobre una red de una manera segura, una conexión sin cifrar es inaceptable. El cifrado es la manera de hacer que cualquier dato sea ilegible. De hecho, hoy en día la prácitca requiere muchos elementos adicionales de seguridad en los algoritmos de cifrado. Deben resistir muchos tipos de ataques conocidos.

El protocolo SSL utiliza diferentes algoritmos de cifrado para asegurarse de que los datos recibidos a través de una red pública son seguros. Tiene mecanismos para detectar cambios de datos, pérdidas, o reenvíos. SSL también incorpora algoritmos que proveen de verificación de identidad, utilizando el X509.

X509 hace posible identificar a alguien en Internet. Es utilizado comúnmente en aplicaciones de comercio electrónico. En resumen, debe haber alguna compañía, llamada ``Autoridad Certificada'' (CA) que asigna certificados electrónicos a cualquiera que los necesita. Los certificados se basan en algoritmos de cifrado asimétricos que tienen dos claves de cifrado (una pública, y otra secreta). El propietario de un certificado puede enseñarselo a otra entidad como prueba de su identidad. Un certificado consiste en la clave pública de su propietario. Cualquier dato cifrado con esta clave pública puede ser solo cifrada utilizando la clave secreta correspondiente, que está en posesión del propietario del certificado.

Si necesita más información sobre SSL, X509, o cifrado, utilice su buscador de Internet favorito para buscar las palabras clave en que esté interesado.

5.7.7.2. Requisitos (OpenSSL)

Para utilizar conexiones SSL entre el servidor MySQL y los programas cliente, su sistema debe tener la capacidad de ejecutar OpenSSL y su versión de MySQL debe ser la 4.0.0 o superior.

Para conseguir que las conexiones seguras funcionen con MySQL, debe hacer lo siguiente:

1. Instale la librería OpenSSL. MySQL ha sido comprobado con OpenSSL 0.9.6. Si necesita OpenSSL, visite http://www.openssl.org.
2. Cuando configure MySQL, ejecute el script `configure` con las opciones `--with-vio` y `--with-openssl`.
3. Asegúrese de que ha actualizado sus tablas grant para que las columnas relacionadas con SSL de la tabla `mysql.user` se hayan agregado. Esto es necesario si las tablas grant provienen de una versión de MySQL anterior a la 4.0.0. El procedimiento de actualización se explica en Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.
4. Para comprobar si un servidor `mysqld` que se está ejecutando tiene soporte para OpenSSL, examine el valor de la variable de sistema `have_openssl`:

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+---------------+-------+
```

```
| Variable_name | Value |
+---------------+-------+
| have_openssl  | YES   |
+---------------+-------+
```

Si el valor es YES, el servidor tiene soporte para conexiones OpenSSL.

5.7.7.3. Montar certificados SSL para MySQL

Aquí tiene un ejemplo para configurar certificados SSL para MySQL:

```
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)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/cacert.pem \
    -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 (optional)
#

openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca  -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 a passphrase from the key (optional)
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca  -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
#

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
```

```
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " '
' > $DIR/my.cnf
```

Para comprobar las conexiones SSL, inicie el servidor de la siguiente manera, donde `$DIR` es la ruta a el directorio donde está el archivo de opciones de ejemplo `my.cnf`:

```
shell> mysqld --defaults-file=$DIR/my.cnf &
```

Entonces ejecute un programa cliente utilizando el mismo archivo de opciones:

```
shell> mysql --defaults-file=$DIR/my.cnf
```

Si tiene una distribución de código fuente de MySQL, usted puede tambier comprobar su configuración modificando el archivo `my.cnf` precedente para que se refiera al certificado y los archivos de claves en el directorio `SSL` de la distribución.

5.7.7.4. Opciones SSL de `GRANT`

MySQL puede comprobar los atributos de un certificado X509 además de la autentificación usual que se basa en nombre de usuario y clave. Para especificar las opciones relacionadas con SSL para una cuenta MySQL, utilice la clausula `REQUIRE` de la sentencia `GRANT`. Consulte Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”.

Hay diferentes maneras de limitar los tipos de conexión para una cuenta:

- Si una cuenta no tiene requerimientos de SSL o X509, las conexiones sin cifrar se permiten siempre que el nombre de usuario y la clave sean válidas. De cualquier manera, se pueden también utilizar conexiones cifradas, si el cliente tiene los certificados y archivos de claves apropiados.
- La opción `REQUIRE SSL` limita al servidor para que acepte únicamente conexiones cifradas SSL para la cuenta. Tenga en cuenta que esta opción puede pasarse por alto si hay algún registro ACL que permite conexiones no-SSL.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
  ```

- `REQUIRE X509` significa que el cliente debe tener un certificado pero que el certificado exacto, entidad certificadora y sujeto no importan. El único requerimiento es que debería ser posible verificar su firma con uno de los certificados CA.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
  ```

- `REQUIRE ISSUER 'issuer'` coloca una restricción en la conexión mediante la cual el cliente debe presentar un certificado X509 válido, emitido por la CA `'issuer'`. Si el cliente presenta un certificado que es valido pero tiene un emisor diferente, el servidor rechaza la conexión. La utilización de certificados X509 siempre implica cifrado, así que la opción `SSL` no es necesaria.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret'
      -> REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
         O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
  ```

  Nótese que el valor de `ISSUER` debe introducirse como una única cadena de caracteres.

- `REQUIRE SUBJECT 'subject'` establece la restricción a los intentos de conexión de que el cliente debe presentar un certificado X509 válido con sujeto `'subject'`. Si el cliente presenta un certificado que, aunque válido, tiene un sujeto diferente, el servidor rechaza la conexión.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com';
```

  Nótese que el valor de `SUBJECT` debe ser introducido como una única cadena de caracteres.

- `REQUIRE CIPHER 'cipher'` es necesario para asegurar que se utilizan longitudes de cifra y claves suficientemente fuertes. El protocolo SSL por sí mismo puede ser débil si se utilizan viejos algorimots con claves de cifrado cortas. Utilizando esta opción, podemos pedir un método exacto de cifrado para permitir una conexión.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Las opciones `SUBJECT`, `ISSUER`, y `CIPHER` pueden combinarse en la sentencia `REQUIRE` de la siguiente manera:

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com'
    -> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Nótese que los valores de `SUBJECT` e `ISSUER` deben ser introducidos cada uno como una única cadena de caracteres.

En MySQL 5.0, la palabra clave `AND` es opcional entre las opciones de `REQUIRE`.

El orden de las opciones no importa, pero ninguna opción puede ser especificada dos veces.

5.7.7.5. Opciones relativas a SSL

La siguiente lista explica las opciones que se utilizan para especificar la utilización de SSL, y archivos de certificados y calves. Se pueden introducir en línea de comandos, o mediante un archivo de opciones.

- `--ssl`

  Para el servidor, esta opción especifica que el servidor permite conexiones SSL. Para un programa cliente, permite al cliente que se conecte al servidor utilizando SSL. Esta opción no es suficiente por si sola para causar que se utilice una conexión SSL. También se deben especificar las opciones `--ssl-ca`, `--ssl-cert`, y `--ssl-key`.

  Esta opción se utiliza más frecuentemente en su forma contraria para indicar que SSL *no* debe utilizarse. Para hacer esto, especifique la opción como `--skip-ssl` o `--ssl=0`.

  Nótese que el uso de `--ssl` *no requiere* una conexión SSL. Por ejemplo, si el servidor o el cliente han sido compilados sin soporte para conexiones SSL, se utilizará una conexión normal sin cifrar.

La manera más confiable de asegurarse de que se utiliza una conexión SSL es crear una cuenta en el servidor que incluya la clausula `REQUIRE SSL` en la sentencia `GRANT`. Then use this account to connect to the server, with both a server and client that have SSL support enabled.

- `--ssl-ca=file_name`

  La ruta a un archivo con una lista de CAs SSL en las que se confía.

- `--ssl-capath=directory_name`

  La ruta a un direcotrio que contiene certificados de CA confiables en formato pem.

- `--ssl-cert=file_name`

  El nombre del archivo de certificado SSL a utilizar para establecer una conexión segura.

- `--ssl-cipher=cipher_list`

  Una lista de cifras permisibles para usar en el cifrado SSL. *`cipher_list`* tiene el mismo formato que el comando `openssl ciphers`.

  Ejemplo: `--ssl-cipher=ALL:-AES:-EXP`

- `--ssl-key=file_name`

  El nombre del archivo de clave SSL utilizado para establecer una conexión segura.

5.7.7.6. Conectarse desde Windows a MySQL remotamente con SSH

Aquí hay una nota sobre como conectar para obtener una conexión segura a un servidor MySQL remoto mediante SSH (por David Carlson `<dcarlson@mplcomm.com>`):

1. Instale un cliente SSH en su máquina Windows. Como usuario, el mejor cliente no-libre que he encontrado es el `SecureCRT` de http://www.vandyke.com/. Otra opción es `f-secure` de http://www.f-secure.com/. También puede encontrar algunos libres mediante `Google` en http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/.

2. Ejecute su cliente SSH Windows. Establezca `Host_Name = URL_o_IP_suservidormysql`. Establezca `userid=su_id_usuario` para entrar al servidor. Este nombre de usuario podría no ser el mismo que el de su cuenta MySQL.

3. Establezca la redirección de puertos. Ya sea una redirección remota (Estableciendo `local_port: 3306`, `remote_host: yourmysqlservername_or_ip`, `remote_port: 3306`) o una redirección local (Estableciendo `port: 3306`, `host: localhost`, `remote port: 3306`).

4. Guarde todo, si no es así, tendrá que rehacerlo la próxima vez.

5. Entre en su servidor con la sesión SSH que acaba de crear.

6. En su máquina Windows, ejecute alguna aplicación ODBC (como Access).

7. Cree un nuevo archivo en Windows y enlácelo a MySQL utilizando el driver ODBC de la misma manera que lo haría normalmente, excepto que debe escribir `localhost` para el nombre de servidor MySQL, no `yourmysqlservername`.

Debería tener una conexión ODBC a MySQL, cifrada utilizando SSH.

5.8. Prevención de desastres y recuperaciones

Esta sección explica como hacer copias de seguridad (completas e incrementales) y como realizar mantenimiento de tablas. La sintaxis de las sentencias SQL descritas aquí se detalla en Sección 13.5, “Sentencias de administración de base de datos”. La mayor parte de la información aquí contenida se aplica principalmente a tablas `MyISAM`. Los procedimientos de copia de seguridad para `InnoDB` se explican en Sección 15.8, “Hacer una copia de seguridad y recuperar una base de datos `InnoDB`”.

5.8.1. Copias de seguridad de bases de datos

Debido a que las tablas de MySQL se almacenan como archivos, es fácil hacer una copia de seguridad. Para hacer una copia consistente haga un `LOCK TABLES` en las tablas relevantes, seguido de un `FLUSH TABLES` para las tablas. Consulte Sección 13.4.5, “Sintaxis de `LOCK TABLES` y `UNLOCK TABLES`” y Sección 13.5.5.2, “Sintaxis de `FLUSH`”. Solo necesita obtener un bloqueo de lectura; esto permite a otros clientes continuar consultando la tabla mientras usted está haciendo una copia de los archivos del directorio de la base de datos. La sentencia `FLUSH TABLES` es necesaria para asegurarse de que todas las páginas de índice activas se escriben al disco antes de que comience la copia.

Si quiere hacer una copia de una tabla a un nivel SQL, puede utilizar `SELECT INTO ... OUTFILE` o `BACKUP TABLE`. Para `SELECT INTO ... OUTFILE`, el archivo de salida no debe existir previamente. Esto también es cierto para `BACKUP TABLE`, ya que permitir que archivos externos sean sobreescritos sería un riesgo de seguridad. Consulte Sección 13.2.7, “Sintaxis de `SELECT`” y Sección 13.5.2.2, “Sintaxis de `BACKUP TABLE`”.

Otra técnica para hacer copias de seguridad de una base de datos es utilizar el programa `mysqldump` o el script `mysqlhotcopy script`. Consulte See Sección 8.7, “El programa de copia de seguridad de base de datos `mysqldump`” y Sección 8.8, “El programa de copias de seguridad de base de datos `mysqlhotcopy`”.

1. Hacer una copia completa de su base de datos:

   ```
   shell> mysqldump --tab=/path/to/some/dir --opt db_name
   ```

   O:

   ```
   shell> mysqlhotcopy db_name /path/to/some/dir
   ```

   También puede simplemente copiar todos los archivos de tablas (`*.frm`, `*.MYD`, y `*.MYI`) siempre que el servidor no esté actualizando nada. El script `mysqlhotcopy` utiliza este método. (Pero tenga en cuenta que estos métodos no funcionan si su base de datos contiene tablas `InnoDB`. `InnoDB` no almacena los contenidos de las tablas en directorios de base de datos, y `mysqlhotcopy` funciona solo para tablas `MyISAM` e `ISAM`.)

2. Pare `mysqld` si se está ejecutando, y después reinicielo con la opción `--log-bin[=file_name]`. Consulte Sección 5.10.3, “El registro binario (Binary Log)”. Los archivos binarios de registro le dan la información que necesita para replicar los cambios que se han producido en la base de datos tras el punto en que usted ejecutó `mysqldump`.

Para las tablas `InnoDB` es posible realizar una copia de seguridad en línea que no requiere bloqueos en las tablas; consulte Sección 8.7, “El programa de copia de seguridad de base de datos `mysqldump`”

MySQL tiene soporte para copias de seguridad incrementales: Usted necesita iniciar el servidor con la opción `--log-bin` para activar el registro binario; consulte Sección 5.10.3, “El registro binario (Binary Log)”. En el momento en que usted quiera realizar una copia de seguridad incremental (que contenga

todos los cambios que han ocurrido desde la última copia de seguridad, completa o incremental), usted debe rotar el registro binario utilizando `FLUSH LOGS`. Hecho esto, necesita copiar a la localización de seguridad todos los registros binarios que daten desde el momento de la última copia de seguridad hasta el último. Estos logs binarios son la copia de seguridad incremental; cuando necesite restaurar la copia, los puede aplicar tal como se explica más adelante. La próxima vez que haga una copia de seguridad compelta, también debe rotar el registro binario haciendo `FLUSH LOGS`, `mysqldump --flush-logs`, o `mysqlhotcopy --flushlogs`. Consulte Sección 8.7, "El programa de copia de seguridad de base de datos `mysqldump`" y Sección 8.8, "El programa de copias de seguridad de base de datos `mysqlhotcopy`".

Si su servidor MySQL es un servidor esclavo de replicación, entonces independientemente del método de copia de seguridad que elija, también debe copiar los archivos `master.info` y `relay-log.info` cuando copie los datos de su esclavo. Estos archivos son siempre necesarios para continuar la replicación después de una restauración de los datos del esclavo. Si su esclavo está replicando comandos `LOAD DATA INFILE`, debería también copiar cualquier archivo `SQL_LOAD-*` que pueda existir en el directorio especificado por la opción `--slave-load-tmpdir`. (Esta localización es por defecto el valor de la variable `tmpdir`, si no se especifica.) El esclavo necesita estos archivos para reiniciar la replicación de cualquier operación `LOAD DATA INFILE` interrumpida.

Si tiene que restaurar tablas `MyISAM`, intente recuperarlas utilizando `REPAIR TABLE` o `myisamchk -r` primero. Esto debería funcionar en el 99.9% de los casos. Si `myisamchk` falla, intente el siguiente procedimiento. Tenga en cuenta que solo funciona si tiene activado el registro binario iniciando el servidor MySQL con la opción `--log-bin`; consulte Sección 5.10.3, "El registro binario (Binary Log)".

1. Restaure la copia de seguridad original de `mysqldump`, o la copia de seguridad binaria.
2. Ejecute el siguiente comando para ejecutar de nuevo las actualizaciones de los registros binarios:

   ```
   shell> mysqlbinlog hostname-bin.[0-9]* | mysql
   ```

   En algunos casos, quizá quiera reejecutar solo ciertos registros binarios, desde ciertas posiciones (lo usual es querer reejecutar todos los registros binarios desde el punto de restauración, excepto, posiblemente, algunas sentencias incorrectas). Consulte Sección 8.5, "La utilidad `mysqlbinlog` para registros binarios" para más información sobre la utilidad `mysqlbinlog` y como utilizarla.

También puede hacer copias de seguridad selectivas de archivos individuales:

- Para volcar la tabla, utilice `SELECT * INTO OUTFILE '`*`file_name`*`' FROM `*`tbl_name`*.
- Para recargar la tabla, restaurela con `LOAD DATA INFILE 'file_name' REPLACE ...` Para evitar registros duplicados, la tabla tiene que tener un índice `PRIMARY KEY` o `UNIQUE`. La palabra clave `REPLACE` hace que los viejos registros sean reemplazados con los nuevos cuando un nuevo registro tiene la misma clave que uno antiguo.

Si tiene problema de rendimientos con su servidor mientras realiza copias de seguridad, una estrategia que puede ayudarle es crear replicación y hacer las copias de seguridad en el esclavo en vez de en el maestro. Consulte Sección 6.1, "Introducción a la replicación".

Si está utilizando un sistema de ficheros Veritas, puede hacer una copia de seguridad así:

1. Desde un programa cliente, ejecute `FLUSH TABLES WITH READ LOCK`.
2. Desde otra línea de comandos, ejecute `mount vxfs snapshot`.
3. Desde el primer cliente, ejecute `UNLOCK TABLES`.
4. Copie los archivos de la captura (snapshot).

5. Desmonte la captura.

5.8.2. Ejemplo de estrategia de copias de seguridad y recuperación

Esta sección explica un procedimiento para realizar copias de seguridad que le permiten recuperar datos tras diferentes tipos de problemas:

- Fallo del sistema operativo
- Fallo de energía
- Fallo del sistema de ficheros
- Problema de hardware (disco duro, placa madre, etc)

Los comandos de ejemplo no incluyen opciones como `--user` y `--password` para los programas `mysqldump` y `mysql`. Usted debería incluir las opciones que sean necesarias para que el servidor MySQL le permita conectarse.

Asumiremos que los datos están almacenados en el motor `InnoDB` de MySQL, que tiene soporte para transacciones y recuperación automática de fallos. Siempre asumiremos que el servidor MySQL está bajo carga de trabajo en el momento del fallo. Si no fuera así, no se necesitaría ninguna recuperación.

Para casos de fallos de energía o de sistema operativo, podemos asumir que el disco de datos de MySQL está disponible tras el reinicio. Puede que entonces los archivos de datos de `InnoDB` no contengan datos consistentes debido al fallo, pero `InnoDB` lee sus registros y encuentra en ellos la lista de transacciones confirmadas y no confirmadas que todavía no han sido volcadas a sus archivos de datos, y los vuelca. La información sobre este proceso de recuperación de errores se le muestra al usuario a través del registro de errores de MySQL. Lo siguiente, es un extracto de ejemplo del registro:

```
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
```

En casos de fallos del sistema de fichero o de hardware, podemos asumir que el disco de datos de MySQL *no* está disponible tras el reinicio. Esto significa que MySQL no puede arrancar normalmente porque algunos bloques de datos del disco no son legibles. En este caso, es necesario reformatear el disco, instalar uno nuevo, o en cualquier caso, corregir el problema subyacente. Después es necesario recuperar nuestros datos de MySQL desde copias de seguridad, lo que significa que tenemos que tener copias ya realizadas. Para asegurarse de que sea así, vayamos hacia atrás en el tiempo y diseñemos una política de copias de seguridad.

5.8.2.1. Política de copias de seguridad

Todos sabemos que las copias de seguridad deben programarse periodicamente. Las copias completos (una captura del estado de los datos en un momento del tiempo) puede hacerse con diferentes herramientas, en MySQL. Por ejemplo, `InnoDB Hot Backup` nos provee con una copia de seguridad en línea sin bloqueos de los archivos de datos de `InnoDB`, y con `mysqldump` obtenemos copias de seguridad lógicas online. En esta explicación utilizaremos `mysqldump`.

Supongamos que hacemos una copia de seguridad el domingo, a las 1 PM, cuando la carga es baja. El siguiente comando hace un hace una copia de seguridad completa de todas nuestras tablas `InnoDB` de todas las bases de datos:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

Esto es una copia de seguridad en línea, sin bloqueos, que no molesta a las lecturas y escrituras de las tablas. Hemos supuesto antes que nuestras tablas son `InnoDB`, así que `--single-transaction` hace una lectura consistente y garantiza que los datos vistos por `mysqldump` no cambian. (Los cambios hechos por otros clientes a las tablas `InnoDB` no son vistas por el proceso `mysqldump`.) Si también tenemos otros tipos de tablas, debemos suponer que no han sido cambiadas durante la copia de seguridad. Por ejemplo, para las tablas `MyISAM` en la base de datos `mysql`, debemos suponer que no se estaban haciendo cambios administrativos a las cuentas MySQL durante la copia de seguridad.

El archivo `.sql` resultante producido por el comando `mysqldump` contiene una serie de sentencias SQL `INSERT` que se pueden utilizar para recargar las tablas volcadas más tarde.

Las copias de seguridad completas son necesarias, pero no son siempre convenientes. Producen ficheros muy grandes y llevan tiempo generarse. No son óptimos en el sentido de que cada copia completa sucesiva incluye todos los datos, incluidas las partes que no han cambiado desde el último. Después de realizar una copia de seguridad completa inicial, es más eficiente hacer copias incrementales. Son más pequeñas, y llevan menos tiempo de realización. A cambio, en el momento de la recuperación, no podrá restaurarlo únicamente recargando la copia completa. También deberá procesar las copias incrementales para recuperar los cambios incrementales.

Para hacer copias de seguridad incrementales, necesitamos guardar los cambios incrementales. El servidor MySQL debería ser iniciado siempre con la opción `--log-bin` para que almacene estos cambios en un archivo mientras actualiza los datos. Esta opción activa el registro binario, así que el servidor escribe cada sentencia SQL que actualiza datos en un archivo llamado registro binario de MySQL. Miremos al directorio de datos de un servidor MySQL que fue iniciado con la opción `--log-bin` y que se ha estado ejecutando durante algunos días. Encontramos estos archivos de registro binario:

```
-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
```

Cada vez que se reinicia, el servidor MySQL crea un nuevo archivo de registro binario utilizando el siguiente número en la secuencia. Mientras el servidor se está ejecutando, podemos comunicarle manualmente que cierre el archivo de registro binario actual y comience otro nuevo, ejecutando la sentencia SQL `FLUSH LOGS` o con el comando `mysqladmin flush-logs`. `mysqldump` también tiene una opción para volcar los logs. El archivo `.index` en el directorio de datos contiene la lista de todos los archivos de registro binario de MySQL en el directorio. Este archivo se utiliza para replicación.

El registro binario de MySQL es importante para la restauración, porque son copias incrementales de datos. Si se asegura de volcar los registros cuando hace su copia de seguridad completa, entonces cualquier registro binario creado tras esa copia contiene todos los cambios hechos desde entonces. Modifiquemos el comando `mysqldump` previo un poco para que vuelque los registros binarios de MySQL en el momento de la copia de seguridad completa, y para que el archivo de volcado contenga el nombre del registro binario actual:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2
           --all-databases > backup_sunday_1_PM.sql
```

Tras ejecutar este comando, el directorio de datos contiene un nuevo archivo de registro binario, `gbichot2-bin.000007`. El archivo `.sql` resultante contiene estas líneas:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Como el comando `mysqldump` ha hecho una copia de seguridad completa, estas líneas significan dos cosas:

- El archivo `.sql` contiene todos los cambios hechos antes de cualquier cambio escrito al registro binario `gbichot2-bin.000007` o posterior.
- Todos los cambios registrados tras la copia de seguridad no están presentes en el archivo `.sql`, pero lo están en el registro binario `gbichot2-bin.000007` o posterior.

El lunes, a las 1 PM, podemos crear una copia de seguridad incremental volcando los registros para comenzar un nuevo registro binario. Por ejemplo, ejecutando un comando `mysqladmin flush-logs` creamos `gbichot2-bin.000008`. Todos los cambios producidos entre el domingo a la 1 PM cuando se hizo la copia completa, y el lunes a la 1 PM están en el archivo `gbichot2-bin.000007`. Esta copia incremental es importante, así que es una buena idea copiarla a un lugar seguro. (Por ejemplo, en cinta o DVD, o copiándolo a otra máquina.) El martes a la 1 PM, ejecute otro comando `mysqladmin flush-logs`. Todos los cambios desde el lunes a la 1 PM hasta el martes a la 1 PM están en el archivo `gbichot2-bin.000008` (que también debería ser copiado a un lugar seguro).

Los registros binarios de MySQL ocupan espacio de disco. Para ligerar espacio, púrguelos de vez en cuando. Una manera de hacerlo es borrar los registros binarios que no se necesiten, como cuando hacemos una copia de seguridad completa:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2
           --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Tenga en cuenta que: Borrar los registros binarios de MySQL con `mysqldump --delete-master-logs` puede ser peligroso si su servidor es un servidor maestro de replicación, porque los servidores esclavos pueden no haber procesado completamente los contenidos del registro binario.

La descripción de la sentencia `PURGE MASTER LOGS` explica lo que debe ser verificado antes de borrar los registros binarios de MySQL. Consulte Sección 13.6.1.1, “Sintaxis de PURGE MASTER LOGS”.

5.8.2.2. Usar copias de seguridad para una recuperación

Ahora suponga que tenemos un fallo catastrófico el miércoles a las 8 AM que requiere restauración de las copias de seguridad. Para recuperarnos, primero restauramos la última copia de seguridad completa que tenemos (la del domingo a la 1 PM). El archivo de copia completo es tan solo una serie de sentencias SQL, así que restaurarlo es muy fácil:

```
```

```
shell> mysql < backup_sunday_1_PM.sql
```

En este punto, el estado de los datos ha sido restaurado al del domingo a la 1 PM. Para restaurar los datos hechos desde entonces, debemos usar las copias incrementales, es decir los archivos de registro binario `gbichot2-bin.000007` y `gbichot2-bin.000008`. Extraigalos, si es necesario, de allá donde estuviesen guardados, y luego procese sus contenidos de la siguiente manera:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

Ahora hemos recuperado los datos hasta su estado del martes a la 1 PM, pero todavía hay cambios que faltan desde esa fecha hasta el momento del fallo. Para no perderlos, deberíamos haber hecho que el servidor MySQL almacenase sus registros MySQL en un lugar seguro (discos RAID, ...) diferente del lugar donde almacena sus archivos de datos, para que estos registros no estuvieran únicamente en el disco destruido. (Es decir, iniciar el servidor con la opción `--log-bin` que especifique una localización en un dispositivo físico diferente del que contiene el directorio de datos. De esta manera, los registros no se pierden aún si el dispositivo que contiene el directorio sí.) Si hubiésemos hecho esto, podríamos tener el archivo `gbichot2-bin.000009` a mano, y podríamos aplicarlo para restaurar hasta los cambios más recientes sin ninguna pérdida a como estaban en el momento del fallo.

5.8.2.3. Sumario de la estrategia de copias de seguridad

En caso de un fallo de sistema operativo o energía, `InnoDB` hace todo el trabajo de restauración de datos por si mismo. Pero para asegurarse de que puede dormir bien, tenga en cuenta los siguientes puntos:

- Ejecute siempre el servidor MySQL con la opción `--log-bin`, o mejor aún `--log-bin=`*`log_name`*, donde el archivo de registro está guardado en algún medio diferente al disco en donde está el directorio de datos. Si tiene un medio seguro tal, también puede ser bueno para hacer balanceo de carga de disco (que resulta en una mejora de rendimiento).
- Haga copias de seguridad completas periodicas, utilizando el último comando `mysqldump` dado previamente, que hace una copia de seguridad en línea sin bloqueo.
- Haga copias de seguridad incrementales periodicamente volcando los registros con `FLUSH LOGS` o `mysqladmin flush-logs`.

5.8.3. Mantenimiento de tablas y recuperación de un fallo catastrófico (crash)

El siguiente texto explica como utilizar `myisamchk` para comprobar o reparar tablas `MyISAM` (tablas con archivos `.MYI` and `.MYD`).

Puede utilizar la utilidad `myisamchk` para obtener información sobre las tablas de su base de datos, o para comprobar, reparar u optimizarlas. Las siguientes secciones explican como invocar `myisamchk` (incluyendo una descripción de sus opciones), como establecer un calendario de mantenimiento de tablas, y como utilizar `myisamchk` para que realice sus diferentes funciones.

Aunque la reparación de tablas con `myisamchk` es bastante segura, siempre es una buena idea hacer una copia de seguridad *antes* de hacer una reparación (o cualquier operación de mantenimiento que pueda hacer muchos cambios a una tabla)

Las operaciones de `myisamchk` que afectan a índices pueden causar que los índices `FULLTEXT` sean recreados con parámetros que son incompatibles con los valores utilizados por el servidor MySQL. Para evitar esto, lea las instrucciones en Sección 5.8.3.2, “Opciones generales demyisamchk”.

En muchos casos, podría encontrar más simple hacer el mantenimiento de las tablas `MyISAM` utilizando las sentencias SQL que realizan las mismas operaciones que `myisamchk`:

- Para comprobar o reparar tablas `MyISAM` tables, use `CHECK TABLE` o `REPAIR TABLE`.
- Para optimizar tablas `MyISAM`, use `OPTIMIZE TABLE`.
- Para analizar tablas `MyISAM`, use `ANALYZE TABLE`.

Estas sentencias pueden ser utilizadas directamente, o mediante el programa cliente `mysqlcheck`, que provee de una interfaz de línea de comandos.

Una de las ventajas de estas sentencias sobre `myisamchk` es que el servidor hace todo el trabajo. Con `myisamchk`, usted debe asegurare de que el servidor no utiliza las tablas al mismo tiempo. Si no es así, podría haber interacciones no deseadas entre `myisamchk` y el servidor.

5.8.3.1. Sintaxis para invocar `myisamchk`

Invoque `myisamchk` de la siguiente manera:

```
shell> myisamchk [opciones] tbl_name
```

Las *`options`* especifican lo que quiere que `myisamchk` haga. Se describen en las secciones posteriores. También puede obtener una lista de las opciones invocando `myisamchk --help`.

Sin opciones, `myisamchk` simplemente comprueba la tabla, como operación por defecto. Para obtener más información, o decirle a `myisamchk` que tome medidas correctivas, especifique las opciones tal como se explica en la siguiente guía.

`tbl_name` es el nombre de la tabla que quiere comprobar o reparar. Si ejecuta `myisamchk` desde otro lugar que no sea el directorio de la base de datos, debe especificar la ruta hasta este directorio, porque `myisamchk` no tiene la más mínima idea de en qué lugar se encuentra la base de datos. De hecho, `myisamchk` no se preocupa por si los archivos sobre los que trabaja se encuentran en un directorio de base de datos. Puede copiar los archivos que corresponden a la tabla en cualquier otro lugar, y hacer las operaciones de recuperación allí.

Puede invocar varios nombres de tablas en la línea de comandos de `myisamchk`, si lo desea. También puede especificar una tabla nombrando a su archivo de índices (el archivo con extensión `.MYI`). Esto permite que especifique todas las tablas de un directorio utilizando el patrón `*.MYI`. Por ejemplo, si se encuentra en un directorio de una base de datos, puede comprobar todas las tablas `MyISAM` de ese directorio de la siguiente manera:

```
shell> myisamchk *.MYI
```

Si no se encuentra en el directorio de la base de datos, puede comprobar todas las tablas especificando la ruta al directorio:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

Incluso se podría comprobar todas las tablas en todas las bases de datos especificando un comodín en la ruta a el archivo de datos MySQL:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

La manera recomendada de comprobar rápidamente todas las tablas `MyISAM` e `ISAM` es:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
shell> isamchk --silent /path/to/datadir/*/*.ISM
```

Si usted quiere comprobar todas las tablas `MyISAM` e `ISAM` y repararlas si alguna está corrompida, puede utilizar los siguientes comandos:

```
shell> myisamchk --silent --force --fast --update-state \
          -O key_buffer=64M -O sort_buffer=64M \
          -O read_buffer=1M -O write_buffer=1M \
          /path/to/datadir/*/*.MYI
shell> isamchk --silent --force -O key_buffer=64M \
          -O sort_buffer=64M -O read_buffer=1M -O write_buffer=1M \
          /path/to/datadir/*/*.ISM
```

Estos comandos asumen que tiene más de 64MB libres. Para más información sobre reserva de memoria con `myisamchk`, consulte Sección 5.8.3.6, “Utilización de la memoria por parte de `myisamchk`”.

Debe asegurarse de que ningún otro programa está utilizando las tablas mientras ejecute `myisamchk`. Si no es así, cuando ejecute `myisamchk`, puede obtener el siguiente mensaje de error:

```
warning: clients are using or haven't closed the table properly
```

Esto significa que está intentando comprobar una tabla que ha sido cambiada por otro programa (como podría ser el servidor `mysqld`) que no ha cerrado aún el archivo, o que ha muerto sin cerrar el archivo adecuadamente.

Si `mysqld` se está ejecutando, debe forzarlo a volcar cualquier modificación de tablas que todavía esté almacenada en memoria, utilizando `FLUSH TABLES`. Debería entonces asegurarse de que nadie está utilizando las tablas sobre las que se ejecuta `myisamchk`. La manera más fácil de evitar este problema es utilizar `CHECK TABLE` en vez de `myisamchk` para comprobar las tablas.

5.8.3.2. Opciones generales de `myisamchk`

Las opciones descritas en esta sección pueden utilizarse para cualquier tipo de operación de mantenimiento de tabla realizada por `myisamchk`. Las secciones que siguen a la presente explican las opciones que pertenecen tan solo a operaciones específicas, como la comprobación o reparación de tablas.

- `--help, -?`

  Muestra un mensaje de ayuda y finaliza.

- `--debug=`*`debug_options`*`, -# `*`debug_options`*

  Escribe un registro de depuración. La cadena *`debug_options`* suele ser frecuentemente `'d:t:o,`*`file_name`*`'`.

- `--silent, -s`

  Modo silencioso. Escribe solo cuando ocurre algún tipo de error. Puede usar `-s` dos veces (`-ss`) para hacer que `myisamchk` sea muy silencioso.

- `--verbose, -v`

  Modo explícito. Imprime más información. Esto puede ser utilizado en conjunción con `-d` y `-e`. Utilice `-v` múltiples veces (`-vv`, `-vvv`) para producir aún más información.

- `--version, -V`

  Muestra información sobre la versión y finaliza.

- `--wait, -w`

  En vez de terminar con un error si la tabla se encuentra bloqueada, espera a que la tabla se desbloquee antes de continuar. Tenga en cuenta que si está ejecutando `mysqld` con la opción `--skip-external-locking`, la tabla sólo puede haber sido bloqueada por otro comando `myisamchk`.

También puede establecer las variables utilizando `--var_name=value` opciones:

Variable	Valor por defecto
`decode_bits`	9
`ft_max_word_len`	dependiente-de-versión
`ft_min_word_len`	4
`ft_stopword_file`	lista interna
`key_buffer_size`	523264
`myisam_block_size`	1024
`read_buffer_size`	262136
`sort_buffer_size`	2097144
`sort_key_blocks`	16
`write_buffer_size`	262136

Las posibles variables de `myisamchk` y sus valores por defecto pueden ser examinadas con `myisamchk --help`:

`sort_buffer_size` se utiliza cuando las claves son reparadas mediante ordenación, que es el caso normal cuando se utiliza `--recover`.

`key_buffer_size` se utiliza cuando se está comprobando una tabla con `--extend-check` o cuando las claves se están reparando insertándolas registro a registro en la tabla (como cuando se hacen inserciones normales). La reparación a través del "key buffer" se utiliza en los siguientes casos:

- Ha utilizado `--safe-recover`.
- Los archivos temporales necesarios para ordenar las claves serían más del doble de grandes que cuando se crea el archivo de claves directmaente. Esto ocurre usualmente cuando hay valores grandes en la clave (tipos de columna como `CHAR`, `VARCHAR`, o `TEXT`), porque la operación de ordenamiento necesita almacenar los valores de las claves por completo mientras procede. Si tiene mucho espacio temporal y puede forzar a que `myisamchk` repare mediante ordenación, puede utilizar la opción `--sort-recover`.

La reparación mediante el "key buffer" necesita mucho menos espacio de disco que utilizar ordenación, pero es también mucho más lenta.

Si quiere una reparación más rápida, establezca las variables `key_buffer_size` y `sort_buffer_size` a un valor aproximadamente dle 25% de la memoria disponible. Puede subir ambas variables a un valor grande, porque solo se utiliza una de ellas a la vez.

`myisam_block_size` es el tamaño utilizado para los bloques de índices.

`ft_min_word_len` y `ft_max_word_len` indican la longitud de palabra mínima y máxima para índices `FULLTEXT`. `ft_stopword_file` indica un archivo de "palabra de parada". Esto necesita establecerse en las siguientes circunstancias.

Si utiliza `myisamchk` para realizar una operación que modifica los índices de las tablas (como una reparación o un análisis), los índices `FULLTEXT` son reconstruidos utilizando el archivo de palabra de parada por defecto a menos que especifique lo contrario. Esto puede resultar en el fallo de sentencias.

El problema ocurre porque estos parámetros son sólo conocidos por el servidor. No están almacenados en los archivos de índices de `MyISAM`. Para evitar el problema, si usted a modificado la longitud mínima o máxima de palabra, o el archivo de palabra de parada en el servidor, especifique los mismos valores de `ft_min_word_len`, `ft_max_word_len`, y `ft_stopword_file` para `myisamchk` que los que ha utilizado para `mysqld`. Por ejemplo, si ha establecido una longitud de palabra mínima de 3, puede reparar una tabla con `myisamchk` así:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

Para asegurarnos de que `myisamchk` y el servidor utilizan los mismos valores para los parámetros de full-text, podemos ponerlos en las secciones `[mysqld]` y `[myisamchk]` de un archivo de opciones:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

Una alternativa a utilizar `myisamchk` es usar las sentencias `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, o `ALTER TABLE`. Estas sentencias son ejecutadas por el servidor, que conoce los parámetros de full-text apropiados que debe utilizar.

5.8.3.3. Opciones de `myisamchk` para comprobar tablas

`myisamchk` soporta las siguientes opciones para las operaciones de comprobación de tablas:

- `--check, -c`

  Comprueba si hay errores en la tabla. Esta es la operación por defecto si no se especifica ninguna opción que seleccione la operación a realizar.

- `--check-only-changed, -C`

  Comprueba sólo las tablas que han cambiado desde la última comprobación.

- `--extend-check, -e`

  Comprueba la tabla muy concienzudamente. Esto es muy lento si la tabla tiene muchos índices. Esta opción debería ser utilizada únicamente en casos extremos. Normalmente, `myisamchk` o `myisamchk --medium-check` deberían ser capaces de determinar si hay algún error en la tabla.

  Si está utilizando `--extend-check` y tiene bastante memoria, establecer un valor grande para la variable `key_buffer_size` ayuda a que la operación de reparación vaya más rápido.

- `--fast, -F`

  Comprueba solo las tablas que no han sido cerrados apropiadamente.

- `--force, -f`

  Realiza una operación de reparación automaticamente si `myisamchk` encuentra cualquier error en la tabla. El tipo de reparación es el mismo que el que se espcifica con la opción `--repair` o `-r`.

- `--information, -i`

Imprime estadísticas informativas sobre la tabla que se comprueba.

- `--medium-check, -m`

  Realiza una comprobación que es más rápida que `--extend-check`. Esta comprobación encuentra solo el 99.99% de todos los errores, lo que debería ser suficiente en la mayor parte de los casos.

- `--read-only, -T`

  No señala la tabla como comprobada. Esto es útil si utiliza `myisamchk` para comprobar una tabla que está siendo utilizada por otra aplicación que no utiliza bloqueos, como `mysqld` cuando se ejecuta con la opción `--skip-external-locking`.

- `--update-state, -U`

  Almacena información en el archivo `.MYI` para indicar que la tabla ha sido comprobada, o si la tabla tiene fallos. Esto debería utilizarse par obtener el máximo beneficio de la opción `--check-only-changed`, pero no debería utilizarse esta opción si se está ejecutando el servidor `mysqld`, con la opción `--skip-external-locking`, y éste está utilizando la tabla.

5.8.3.4. Opciones de `myisamchk` para reparar tablas

`myisamchk` tiene soporte para las siguientes opciones en las operaciones de reparación de tablas:

- `--backup, -B`

  Realiza una copia de seguridad del archivo `.MYD` con el formato `file_name-time.BAK`

- `--character-sets-dir=`*`path`*

  El directorio donde los juegos de caracteres están instalados. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--correct-checksum`

  Corrige la información de checksum de la tabla.

- `--data-file-length=#, -D #`

  Longitud máxima del archivo de datos. (cuando se está reconstruyendo el archivo de datos porque está ``lleno'').

- `--extend-check, -e`

  Realiza una reparación que intenta recuperar todos los registros posibles del archivo de datos. Normalmente esto también encuentra un montón de registros basura. No utilice esta opción al menos que esté desesperado.

- `--force, -f`

  Sobreescribe los archivos temporales viejos (archivos con nombres como *`tbl_name`*`.TMD`) en vez de interrumpir la reparación.

- `--keys-used=#, -k #`

  Para `myisamchk`, el valor de esta opción indica qué índices tiene que actualizar. Cada bit binario de la opción, corresponde a un índice de la tabla, donde el primer índice es el bit 0. Para `isamchk`, esta

opción indica que sólo los primeros # índices de la tabla deben ser actualizados. En cualquier caso, un valor de 0 en la opción desahbilita las actualizaciones a todos los índices, lo que puede ser utilizado para obtener inserciones más rápidas. Los índices desactivados pueden ser reactivados utilizando `myisamchk -r`.

- `--no-symlinks, -l`

  No sigue los enlaces simbólicos. Normalmente repara una tabla que esté apuntada por un enlace simbólico. Esta opción no existe en MySQL 4.0, porque las versiones a partir de la 4.0 no eliminan los enlaces simbólicos durante las operaciones de reparación.

- `--parallel-recover, -p`

  Utiliza la misma técnica que `-r` y `-n`, pero crea todas las claves en paralelo, utilizando hilos de ejecución diferentes. *Esto es código en estado alpha. Utilícelo bajo su propia responsabilidad.*

- `--quick, -q`

  Consigue una reparación más rápida al no modificar el archivo de datos. Puede especificar esta opción dos veces para forzar a `myisamchk` a modificar el archivo original de datos en el caso de claves duplicadas.

- `--recover, -r`

  Realiza una reparación que puede resolver casi cualquier problema, excepto las claves únicas que no son únicas (que es un error extremadamente raro en tablas `MyISAM`). Si quiere recuperar una tabla, esta es la primera opción a intentar. Debería intentar con `-o` sólo si `myisamchk` comunica que la tabla no puede recuperarse con `-r`. (En el improbable caso de que `-r` falle, el archivo de datos permanece intacto.)

  Si tiene memoria suficiente, debería incrementar el valor de `sort_buffer_size`.

- `--safe-recover, -o`

  Hace una reparación utilizando un méto de recuperación antiguo que lee todos los registros en orden y actualiza todos los árboles de índices basándose en los registros encontrados. Esto es de un orden de magnitud más lento que `-r`, pero puede gestionar un puñado de casos muy improbables que `-r` no puede. Este método de recuperación también utiliza mucho menos espacio de disco que `-r`. Normalmente debería reparar primero con `-r`, y luego utilizar `-o` solo si `-r` falla.

  Si tiene mucha memoria, debería incrementar el valor de `key_buffer_size`.

- `--set-character-set=`*`name`*

  Cambia el juego de caracteres utilizado por los índices de tabla. Esta opción fue reemplazada por `--set-collation` en MySQL 5.0.3.

- `--set-collation=`*`name`*

  Cambia la colación utilizada para ordenar los índices de las tablas. El nombre del código de caracteres viene implícito en la primera parte del nombre de la colación. Esta opción fue añadida en MySQL 5.0.3.

- `--sort-recover, -n`

  Fuerza a que `myisamchk` utilice ordenación para establecer las claves aunque los archivos temporales puedan ser muy grandes.

- `--tmpdir=`*`path`*`, -t `*`path`*

La ruta a el directorio que debe utilizarse para almacenar archivos temporales. Si no está establecida, `myisamchk` utiliza el valor de la variable de entorno `TMPDIR`. `tmpdir` puede ser establecido como una lista de rutas de directorios que son utilizadas sucesivamente de una manera a lo "round-robin" para crear archivos temporales. El carácter de separación entre los nombres de directorio es el de dos puntos en Unix (':') y el punto y coma en Windows, Netware, y OS/2 (';').

- `--unpack, -u`

  Descomprime una tabla que fue comprimida con `myisampack`.

5.8.3.5. Otras opciones de `myisamchk`

`myisamchk` tiene soporte para las siguientes opciones en cuanto a acciones que no son comprobaciones ni reparaciones de tablas:

- `--analyze, -a`

  Analiza la distribución de las claves. Esto mejora el rendimiento de las join habilitando al optimizador de join para escober mejor el orden en que unirá las tablas y qué claves debería utilizar. Para obtener información sobre la distribución, utilice el comando `myisamchk --description --verbose tbl_name` o la sentencia `SHOW KEYS FROM tbl_name`.

- `--description, -d`

  Imprime información descriptiva sobre la tabla.

- `--set-auto-increment[=value], -A[value]`

  Fuerza que la numeración auto incremental para los nuevos registros comience en un valor dado (o superior, si existen registros con valores autoincrementales tan altos). Si *`value`* no está especificado, el incremente automático comienza con al valor más grande que haya actualmetne en la tabla, más uno.

- `--sort-index, -S`

  Ordena los bloques de árboles de índices de mayor a menor. Esto optimiza las búsquedas y hace que el escaneo por clave de las tablas sea más rápido.

- `--sort-records=#, -R #`

  Ordena los registros de acuerdo a un índice particular. Esto hace que sus datos están mucho más localizables y puede agilizar operaciones `SELECT` y `ORDER BY` basados en rangos que utilicen este índice. (La primera vez que utilice esta opción para ordenar una tabla, puede ser muy lento.) Para determinar los números de índices de una tabla, utilice `SHOW KEYS`, que muestra los índices de una tabla en el mismo orden en que `myisamchk` los ve. Los ínidces están numerados comenzando por 1.

5.8.3.6. Utilización de la memoria por parte de `myisamchk`

La reserva de memoria es importante cuando ejecute `myisamchk`. `myisamchk` no utiliza más memoria de la que se especifique con las opciones `-O`. Si usted va a utilizar `myisamchk` en tablas muy grandes, debería primero decidir cuanta memoria quiere utilizar. Por defecto, se utilizarán más o menos 3MB para realizar reparaciones. Utilizando valores más grandes, puede conseguir que `myisamchk` funcione más rápido. Por ejemplo, si tiene más de 32MB de RAM, podría utilizar estas otras opciones (además de las otras que pueda especificar):

```
shell> myisamchk -O sort=16M -O key=16M -O read=1M -O write=1M ...
```

Utilizar `-O sort=16M` probablemente sea suficiente para la mayoría de los casos.

Tenga en cuenta que `myisamchk` utiliza archivos temporales en `TMPDIR`. Si `TMPDIR` apunta a un sistema de ficheros en memoria, podría quedarse fácilmente sin memoria y obtener errores. Si esto ocurre, haga que `TMPDIR` apunte a algún directorio de un sistema de ficheros con más espacio y ejecute el comando `myisamchk` de nuevo.

Mientras está reparando, `myisamchk` también necesita mucho espacio de disco:

- Doble el espacio del archivo de datos (el original y una copia). Este espacio no se necesitará si hace una reparación con la opción `--quick`; en este caso solo se reconstruye el archivo de índices (la copia se crea en el mismo directorio que el original.)
- El espacio para el archivo de índices que reemplaza al viejo. El archivo de índices viejo se trunca en el inicio de la operación de reparación, así que usualmente este espacio se ignora. Este espacio debe estar en el mismo sistema de ficheros que el archivo de índices original.
- Cuando utilice `--recover` o `--sort-recover` (pero no al utilizar `--safe-recover`), necesitará espacio para un buffer de ordenación. El espacio requerido es:

  ```
  (clave_más_larga + longitud_de_puntero_a_registro) * número_de_registros * 2
  ```

  Puede comprobar la longitud de las claves y la longitud_de_puntero_a_registro con `myisamchk -dv tbl_name`. Este espacio se reserva en el directorio temporal (especificado por `TMPDIR` o `--tmpdir=path`).

Si tiene algún problema con el espacio de disco durante una reparación, puede intentar utilizar `--safe-recover` en vez de `--recover`.

5.8.3.7. Usar `myisamchk` para recuperación de desastres

Si ejecuta `mysqld` con `--skip-external-locking` (lo que es la opción por defecto en algunos sistemas, como Linux), no puede utilizar `myisamchk` de manera segura para comprobar una tabla si `mysqld` está utilizando esta misma tabla. Si puede asegurarse de que nadie está accediendo a las tablas a través de `mysqld` mientras ejecuta `myisamchk`, entonces solo tendrá que hacer `mysqladmin flush-tables` antes de comenzar la comprobación de las tablas. Si no puede garantizar esto, entonces debe parar `mysqld` mientras comprueba las tablas. Si ejecuta `myisamchk` mientras `mysqld` está actualizando las tablas, puede obtener un aviso de que una tabla está corrupta aún cuando no sea así.

Si no está utilizando `--skip-external-locking`, puede utilizar `myisamchk` para comprobar tablas en cualquier momento. Mientras hace esto, todos los clientes que intenten actualizar la tabla esperarán hasta que `myisamchk` esté listo para continuar.

Si utiliza `myisamchk` para reparar u optimizar tablas, *debe* siempre asegurarse de que el servidor `mysqld` no está utilizando la tabla (esto es también aplicable cuando el usted esté utilizando `--skip-external-locking`). Si no apaga `mysqld`, al menos debe hacer un `mysqladmin flush-tables` antes de ejecutar `myisamchk`. Sus tables *podrían corromperse* si el servidor y `myisamchk` acceden a las tablas simultáneamente.

Esta sección describe como comprobar y actuar si existe corrupción en bases de datos MySQL. Si sus tablas se corrompen frecuentemente, usted debería intentar encontrar la razón. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”.

Para una explicación de como las tablas `MyISAM` pueden corromperse, consulte Sección 14.1.4, “Problemas en tablas MyISAM”.

Cuando realice recuperación de fallos, es importante entender que cada tabla `MyISAM` con nombre *`tbl_name`* en una base de datos corresponde a tres archivos en el directorio de la base de datos:

Archivo	Propósito
`tbl_name``.frm`	Archivo de definición (formato)
`tbl_name``.MYD`	Archivo de datos
`tbl_name``.MYI`	Archivo de índices

Cada uno de estos tres tipos de archivos tiene un tipo de peligro de corrupción diferente, pero la mayoría de problemas ocurren más frecuentemente en los archivos de datos e índices.

`myisamchk` trabaja creando una copia de el archivo de datos `.MYD` registro a registro. Finaliza el estadio de reparación borrando el viejo archivo `.MYD` y renombrando el nuevo archivo a el nombre original. Si utiliza `--quick`, `myisamchk` no crea un archivo `.MYD` temporal, sino que asume que el archivo `.MYD` está correcto y solo genera un nuevo archivo de índices sin tocar el archivo `.MYD`. Esto es seguro porque `myisamchk` automáticamente detecta sí el archivo `.MYD` está corrompido, y para la reparación si lo está. También puede especificar la opción `--quick` dos veces. En este caso, `myisamchk` no aborta al encontrar algunos errores (como erroes de clave duplicada), sino que intenta resolverlo modificando el archivo `.MYD`. Normalmente, la utilización de dos opciones `--quick` es útil sólo si tiene muy poco espacio libre para realizar una reparación normal. En este caso, al menos debería hacer una copia de seguridad antes de ejecutar `myisamchk`.

5.8.3.8. Cómo comprobar tablas `MyISAM` en busca de errores

Para comprobar una tabla `MyISAM`, utilice los siguientes comandos:

- `myisamchk` *`tbl_name`*

  Esto encuentra el 99.99% de todos los errores. Lo que no puede encontrar es corrupción que involucre *sólo* al archivo de datos (lo que es muy inusual). Si quiere comprobar una tabla, normalmente debería ejecutar `myisamchk` sin opciones, o al menos con la opción `-s` o `--silent`.

- `myisamchk -m` *`tbl_name`*

  Esto encuentra el 99.999% de todos los errores. Primero comprueba todos los índices, y después lee todos los registros. Calcula en checksum para todas las claves en los registros, y verifica que coincida con el de las claves en el árbol de índices.

- `myisamchk -e` *`tbl_name`*

  Esto hace una comprobación completa y exhaustiva de todos los datos (`-e` significa ``comprobación extendida''). Hace una comprobación-lectura de la clave de cada registro para verificar que de hecho apuntan al registro correcto. Esto normalmente puede llevar mucho tiempo para una tabla grande que tenga muchos índices. Normalmente `myisamchk` se para tras el primer error que encuentra. Si quiere obtener más información, puede añadir la opción `--verbose` (`-v`). Esto causa que `myisamchk` durante un máximo de 20 errores.

- `myisamchk -e -i` *`tbl_name`*

  Igual que el comando previo, pero la opción `-i` le dice a `myisamchk` que imprima también algunas estadísticas informativas.

En la mayoría de casos, un comando `myisamchk` sin más argumentos que el nombre de la tabla original es suficiente para comprobar la tabla.

5.8.3.9. Cómo reparar tablas

La discusión en esta sección describe cómo usar `myisamchk` en tablas `MyISAM` (extensiones `.MYI` y `.MYD`).

También puede (y debe, si es posible) usar los comandos `CHECK TABLE` y `REPAIR TABLE` para chequear y reparar tablas `MyISAM` . Consulte Sección 13.5.2.3, “Sintaxis de CHECK TABLE” y Sección 13.5.2.6, “Sintaxis de REPAIR TABLE”.

Los síntomas de tablas corruptas incluyen consultas que abortan inesperadamente y errores observables como los siguientes:

- *`tbl_name`*`.frm` is locked against change
- Can't find file *`tbl_name`*`.MYI` (Errcode: *`###`*)
- Unexpected end of file
- Record file is crashed
- Got error *`###`* from table handler

Para obtener ejecución acerca de los errores puede ejectuar `perror` *`###`*, donde *`###`* es el número de error. El siguiente ejemplo muestra cómo usar `perror` para encontrar el significado de la mayoría de errores comunes que indican un problema con la tabla:

```
shell> perror 126 127 132 134 135 136 141 144 145
126 = Index file is crashed / Wrong file format
127 = Record-file is crashed
132 = Old database file
134 = Record was already deleted (or record file crashed)
135 = No more room in record file
136 = No more room in index file
141 = Duplicate unique key or constraint on write or update
144 = Table is crashed and last repair failed
145 = Table was marked as crashed and should be repaired
```

Tenga en cuenta que el error 135 (no more room in record file) y el error 136 (no more room in index file) no son errores que puedan arreglarse con una simple reparación. En este caso, debe usar `ALTER TABLE` para incrementar los valores de las opciones de tabla `MAX_ROWS` y `AVG_ROW_LENGTH`:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

Si no conoce los valores actuales de las opciones de tabla, use `SHOW CREATE TABLE` o `DESCRIBE`.

Para los otros errores, debe reparar las tablas. `myisamchk` normalmente detecta y arregla la mayoría de problemas que ocurren.

El proceso de reparación incluye hasta cuatro etapas, descritas aquí. Antes de empezar, debe cambiar la localización al directorio de la base de datos y comprobar los permisos de los ficheros de las tablas. En Unix, asegúrese que puede leerlos el usuario con el que corre `mysqld` (y con su usuario, ya que necesita acceder a los ficheros que está comprobando). En caso que necesite modificar ficheros, debe tener también permiso de escritura.

Las opciones que puede usar para el mantenimiento de tablas con `myisamchk` y `isamchk` se describen en varias de las primeras subsecciones de Sección 5.8.3, “Mantenimiento de tablas y recuperación de un fallo catastrófico (crash)”.

La siguiente sección es para los casos en los que los comandos anteriores fallen o si quiere usar las características extendidas que `myisamchk` y `isamchk` proporcionan.

Si va a reparar una tabla desde la línea de comandos, debe parar el servidor `mysqld` en primer lugar. Tenga en cuenta que cuando ejectua `mysqladmin shutdown` en un servidor remoto, el servidor `mysqld` todavía está activo durante un periodo de tiempo una vez que `mysqladmin` devuelve el control, hasta que todas las consultas están paradas y todas las claves se han volcado a disco.

Etapa 1: Comprobación de tablas

Ejecute `myisamchk *.MYI` o `myisamchk -e *.MYI` si tiene más tiempo. Use la opción `-s` (silencio) para suprimir información innecesaria.

Si el servidor `mysqld` está parado, debe usar la opción `--update-state` para decirle a `myisamchk` que marque la tabla como 'comprobada'.

Debe reparar sólo las tablas en que `myisamchk` anuncia un error. Para estas tables, pase a la Etapa 2.

Si obtiene errores extraños al hacer la comprobación (tales como errores `out of memory`), o si `myisamchk` cae, pase a la Etapa 3.

Etapa 2: Reparación sencilla

Nota: Si quiere que una operación de reparación sea mucho más rápida, debe cambiar los valores de las variables `sort_buffer_size` y `key_buffer_size` al 25% aproximado de la cantidad de memoria disponible al ejecutar `myisamchk` o `isamchk`.

En primer lugar, intente `myisamchk -r -q` *`tbl_name`* (`-r -q` significa ``modod de recuperación rápido''). Intenta reparar el fichero de indexación sin tocar el fichero de datos. Si el fichero de datos contiene todo lo que debería y los vínculos de borrado apuntan a la localización correcta dentro del fichero de datos, esto debería funcionar, y la tabla estaría reparada. Empiece a reparar la siguiente tabla. Si no es así, use el siguiente procedimiento:

1. Haga una copia de seguridad del fichero de datos antes de continuar.
2. Use `myisamchk -r` *`tbl_name`* (`-r` significa ``modo de recuperación''). Esto elimina registros incorrectos y registros borrados del fichero de datos y recunstruye el fichero de indexación.
3. Si el paso precedente falla, use `myisamchk --safe-recover` *`tbl_name`*. El modo de recuperación seguro usa un método de reucperación antiguo que soporta algunos casos que los metodos normales de recuperación no soportan (pero es más lento).

Si obtiene errores extraños al reparar (tales como errores `out of memory`), o si `myisamchk` cae, pase a la Etapa 3.

Etapa 3: Reparaciones complicadas

Debe llegar a esta etapa sólo si el primer bloque de 16KB en el fichero de indexación está destruido o contiene información incorrecta, o si el fichero de indexación no se encuentra. En este caso, es necesario crear un nuevo fichero de indexación. Hágalo así:

1. Mueva el fichero de datos a una ubicación segura.
2. Use el fichero descriptor de la tabla para crear unos ficheros de datos e indexación nuevos (vacíos):

```
shell> mysql db_name
```

```
mysql> SET AUTOCOMMIT=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

Si su versión de MySQL no soporta `TRUNCATE TABLE`, use `DELETE FROM tbl_name` en su lugar.

3. Copie el antiguo fichero de datos otra vez sobre el nuevo fichero de datos (recién creado). (No se limite a mover el fichero antiguo sobre el nuevo; debe guardar una copia por si algo falla.)

Vuelva a la Etapa 2. `myisamchk -r -q` debería funcionar. (Este no debería ser un bucle sin fin.)

Puede usar `REPAIR TABLE tbl_name USE_FRM`, que realiza el proceso completo automáticamente.

Etapa 4: Reparaciones muy complicadas

Debe llegar a esta etapa sólo si el fichero de descripción `.frm` ha tenido problemas. Esto no debería ocurrir nunca, ya que el fichero de descripción nunca cambia una vez que la tabla se crea:

1. Restaure el fichero de descripción desde una copia de seguridad y vuelva a la Etapa 3. También puede restaurar el fichero índice y volver a la Etapa 2. En este último caso, puede comenzar con `myisamchk -r`.

2. Si no tiene una copia de seguridad pero sabe exactamente cómo se creó la tabla, cree una copia de la tabla en otra base de datos. Borre el nuevo fichero de datos, luego mueva los ficheros `.frm` de descripción y `.MYI` de indexación desde la otra base de datos a la base de datos que tiene problemas. Esto le da unos nuevos ficheros de descripción e indexación, pero deja el fichero de datos `.MYD` solo. Vuelva a la Etapa 2 y trate de reconstruir el fichero de indexación.

5.8.3.10. Optimización de tablas

Para eliminar registros fragmentados y eliminar espacio desperdiciado resultante del borrado o actualización de registros, ejecute `myisamchk` en modo recuperación:

```
shell> myisamchk -r nombre_tabla
```

Puede optimizar una tabla de la misma forma usando el comando SQL `OPTIMIZE TABLE`. `OPTIMIZE TABLE` realiza una reparación de la tabla y un análisis de las claves, y también ordena el árbol de índices para obtener un mejor rendimiento en la búsqueda de claves. No hay posibilidad de interacción no deseada entre una utilidad y el servidor, ya que el servidor hace todo el trabajo cuando usa `OPTIMIZE TABLE`. Consulte Sección 13.5.2.5, “Sintaxis de OPTIMIZE TABLE”.

`myisamchk` tiene una serie de opciones que puede usar para mejorar el rendimiento de una tabla:

- `-S`, `--sort-index`
- `-R index_num`, `--sort-records=index_num`
- `-a`, `--analyze`

Para una descripción completa de estas opciones, consulte Sección 5.8.3.1, “Sintaxis para invocar myisamchk”.

5.8.4. Organizar un programa de mantenimiento de tablas

Es una buena práctica realizar chequeos de las tablas regularmente en lugar de esperar a que ocurran los problemas. Una forma de chequear y reparar tablas `MyISAM` es con los comandos `CHECK TABLE` y

`REPAIR TABLE` . Consulte Sección 13.5.2.3, “Sintaxis de `CHECK TABLE`” y Sección 13.5.2.6, “Sintaxis de `REPAIR TABLE`”.

Otro modo de chequear tablas es usar `myisamchk`. Para mantenimiento, puede usar `myisamchk -s`. La opción `-s` (forma corta de `--silent`) hace que `myisamchk` se ejecute en modo silencioso, mostrando mensajes sólo cuando hay algún error.

Es una buena idea chequear las tablas al arrancar el servidor. Por ejemplo, cada vez que la máquina se reinicia durante una actualización, normalmente necesita chequear todas las tablas que hayan podido ser afectadas. (Éstas son las ``tablas que creemos que han fallado.'') Para chequear automáticamente tablas `MyISAM` , arranque el servidor con la opción `--myisam-recover` .

Un test todavía mejor sería chequear cualquier tabla cuya fecha de última modificación es más reciente que la del fichero `.pid`.

Debe chequear las tablas regularmente durante operaciones normales del sistema. En MySQL AB, utilizamos un trabajo `cron` para chequear todas nuestras tablas importantes una vez a la semana, usando una línea como esta en un fichero `crontab` :

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

Esto muestra información acerca de tablas que han fallado de forma que podemos examinarlas y repararlas cuando es necesario.

Como no hemos tenido tablas que hayan fallado inesperadamente (tablas que se corrompen por razones distintas a fallos de hardware) durante un par de años (esto es cierto), una vez a la semana es más que suficiente para nosotros.

Recomendamos que para empezar, ejecute `myisamchk -s` cada noche en todas la tablas que se hayan actualizado durante las últimas 24 horas, hasta que confíe en MySQL tanto como nosotros.

Normalmente, las tablas MySQL necesitan poco mantenimiento. Si cambia las tablas `MyISAM` con registros de tamaño dinámico (tablas con columnas `VARCHAR`, `BLOB`, o `TEXT`) o tiene tablas con muchos registros borrados puede que quiera defragmentar/reaprovechar espacio de las tablas de vez en cuendo (una vez al mes?).

Puede hacerlo con `OPTIMIZE TABLE` en las tablas en cuestión. O, si puede parar el servidor `mysqld` por un rato, cambiando la localización al directorio de datos y usando este comando mientras el servidor está parado:

```
shell> myisamchk -r -s --sort-index -O sort_buffer_size=16M */*.MYI
```

5.8.5. Obtener información acerca de una tabla

Para obtener una descripción de tabla o estadísticas acerca de ella, use el comando mostrado a continuación:

- `myisamchk -d` *`nombre_tabla`*

  Ejecute `myisamchk` en ``modo descripción'' para producir una descripción de la tabla. Si inicia el servidor MySQL usando la opción `--skip-external-locking`, `myisamchk` puede reportar un error para una tabla que se actualice mientras está en ejecución. Sin embargo, ya que `myisamchk` no cambia la tabla en modo descripción, no hay riesgo de destruir los datos.

- `myisamchk -d -v` *`nombre_tabla`*

Añadiendo `-v` `myisamchk` se ejecuta en modo información (verbose) así que produce más información acerca de lo que está haciendo.

- `myisamchk -eis` *`nombre_tabla`*

  Muestra sólo la información más importante de una tabla. Esta operación es lenta ya que debe leer la tabla entera.

- `myisamchk -eiv` *`nombre_tabla`*

  Es como `-eis`, pero dice lo que está haciendo.

A continuación hay unos ejemplos de la salida de estos comandos. Están basados en una tabla con estos tamañanos de fichero de datos e índices:

```
-rw-rw-r--   1 monty    tcx     317235748 Jan 12 17:30 company.MYD
-rw-rw-r--   1 davida   tcx      96482304 Jan 12 18:35 company.MYM
```

Ejemplos de salida de `myisamchk -d`:

```
MyISAM file:     company.MYI
Record format:   Fixed length
Data records:    1403698  Deleted blocks:         0
Recordlength:    226

table description:
Key Start Len Index   Type
1   2     8   unique  double
2   15    10  multip. text packed stripped
3   219   8   multip. double
4   63    10  multip. text packed stripped
5   167   2   multip. unsigned short
6   177   4   multip. unsigned long
7   155   4   multip. text
8   138   4   multip. unsigned long
9   177   4   multip. unsigned long
    193   1           text
```

Ejemplo de salida de `myisamchk -d -v`:

```
MyISAM file:         company
Record format:       Fixed length
File-version:        1
Creation time:       1999-10-30 12:12:51
Recover time:        1999-10-31 19:13:01
Status:              checked
Data records:            1403698  Deleted blocks:              0
Datafile parts:          1403698  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     3
Max datafile length:  3791650815  Max keyfile length: 4294967294
Recordlength:               226

table description:
Key Start Len Index   Type                 Rec/key     Root Blocksize
1   2     8   unique  double                     1 15845376      1024
2   15    10  multip. text packed stripped       2 25062400      1024
3   219   8   multip. double                    73 40907776      1024
4   63    10  multip. text packed stripped       5 48097280      1024
5   167   2   multip. unsigned short          4840 55200768      1024
```

```
6   177   4   multip. unsigned long           1346 65145856      1024
7   155   4   multip. text                    4995 75090944      1024
8   138   4   multip. unsigned long             87 85036032      1024
9   177   4   multip. unsigned long            178 96481280      1024
    193   1           text
```

Ejemplo de salida de `myisamchk -eis`:

```
Checking MyISAM file: company
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:  98%  Packed:   17%

Records:          1403698    M.recordlength:     226
Packed:             0%
Recordspace used:     100%   Empty space:          0%
Blocks/Record:   1.00
Record blocks:    1403698    Delete blocks:        0
Recorddata:     317235748    Deleted data:         0
Lost space:             0    Linkdata:             0

User time 1626.51, System time 232.36
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 627, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 639, Involuntary context switches 28966
```

Ejemplo de salida de `myisamchk -eiv`:

```
Checking MyISAM file: company
Data records: 1403698   Deleted blocks:       0
- check file-size
- check delete-chain
block_size 1024:
index  1:
index  2:
index  3:
index  4:
index  5:
index  6:
index  7:
index  8:
index  9:
No recordlinks
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
- check data record references index: 2
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
- check data record references index: 3
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
- check data record references index: 4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
- check data record references index: 5
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 6
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 7
```

```
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 8
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 9
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:   9%  Packed:   17%

- check records and index references
[LOTS OF ROW NUMBERS DELETED]

Records:         1403698   M.recordlength:   226   Packed:           0%
Recordspace used:    100%  Empty space:        0%  Blocks/Record: 1.00
Record blocks:   1403698   Delete blocks:      0
Recorddata:    317235748   Deleted data:       0
Lost space:            0   Linkdata:           0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, Involuntary context switches 122798
```

Explicación de los tipos de información que produce `myisamchk` se dan a continación. ``Keyfile'' se refiere al fichero índice. ``Record'' y ``row'' son sinónimos.

- `MyISAM file`(Fichero MyISAM)

  Nombre del fichero índice `MyISAM`.

- `File-version`(Versión del fichero)

  Versión del formato `MyISAM`. Actualmente siempre es 2.

- `Creation time`(Fecha de creación)

  Cuando se creó el fichero de datos.

- `Recover time`(Fecha de recuperación)

  Cuando el fichero de índices/datos se reconstruyó por última vez.

- `Data records`(Fecha de recuperación)

  Cuántos registros hay en la tabla.

- `Deleted blocks`(Bloques borrados)

  Cuántos bloques borrados todavía tienen espacio reservado. Puede optimizar la tabla para minimizar este espacio. Consulte Sección 5.8.3.10, “Optimización de tablas”.

- `Datafile parts`(Partes de ficheros de datos)

  Para el formato de registros dinámicos, indica cuántos bloques de datos hay. Para una tabla optimizada sin registros fragmentados, es lo mismo que `Registros de datos`.

- `Deleted data`(Datos borrados)

  Cuántos bytes hay de datos borrados sin reclamar. Puede optimizar la tabla para minimizar este espacio. Consulte Sección 5.8.3.10, “Optimización de tablas”.

- `Datafile pointer`(Puntero fichero de datos)

El tamaño del puntero del fichero de datos, en bytes. Normalmente es de 2, 3, 4, o 5 bytes. La mayoría de tablas se administran con 2 bytes, pero MySQL no puede controlar esto todavía. Para tablas arregladas, esta es la dirección de un registro. Para tablas dinámicas, es la dirección de un byte.

- `Keyfile pointer`(Puntero fichero de claves)

  El tamaño del puntero del fichero índice, en bytes. Normalmente es de 1, 2, o 3 bytes. La mayoría de tablas se administran con 2 bytes, pero esto se calcula automáticamente por MySQL. Siempre es la dirección de un bloque.

- `Max datafile length`(Máximo tamaño fichero de datos)

  Cuánto puede crecer el fichero de datos, en bytes.

- `Max keyfile length`(Máximo tamaño fichero de claves)

  Cuánto puede creceer el fichero índice, en bytes.

- `Recordlength`(Tamaño registro)

  El tamaño de cada registro, en bytes.

- `Record format`(Formato de registro)

  El formato usado para almacenar registro de tablas. Los ejemplos precedentes usan `Fixed length`. Otros valores posibles son `Compressed` y `Packed`.

- `table description`(Descripción de tabla)

  Una lista de todas las claves en la tabla. Para cada clave, `myisamchk` muestra alguna información a bajo nivel:

  - `Key`(Clave)

    El número de esta clave.

  - `Start`(Inicio)

    Dónde en el registro comienza esta parte del índice.

  - `Len`(Longitud)

    El tamaño de esta parte del índice. Para números empaquetados, este debería ser siempre la longitud completa de la columna. Para cadenas de caracteres, podría ser menor que la longitud total de la columna indexada, ya que puede indexar un prefijo de una columna de cadenas de caracteres.

  - `Index`(Índice)

    Si un valor clave puede existir múltiples veces en el índice. Los valores son `unique` o `multip.` (múltiples).

  - `Type`(Tipo)

    Qué tipo de datos tiene esta parte del índice. Este es un tipo de datos `MyISAM` con las opciones `packed`, `stripped`, o `empty`.

  - `Root`(Raíz)

Dirección del bloque índice raíz.

- `Blocksize`(Tamaño de bloque)

  El tamaño de cada bloque de índices. Por defecto es de 1024, pero el valor puede cambiar en tiempo de compilación cuando MySQL se construye desde el código fuente.

- `Rec/key`(Registros/clave)

  Este es un valor estadístico usado por el optimizador. Dice cuántos registros hay por valor para esta clave. Una clave única siempre tiene un valor de 1. Puede actualizarse tras la carga de una tabla (o muy cambiada) con `myisamchk -a`. Si no se actualiza en absoluto, el valor por defecto es 30.

Para la tabla mostrada en los ejemplos, hay dos líneas para `table description` (descripción de tabla) para el noveno índice. Esto indica que es un índice de múltiples partes, en este caso con dos.

- `Keyblocks used`(Bloques de claves usados)

  Porcentaje de los bloques de clave usados. Cuando una tabla se ha reorganizad con `myisamchk`, como en los ejemplos, los valores son muy altos (muy cerca del máximo teórico).

- `Packed`(Empaquetado)

  MySQL intenta empaquetar las claves con un sufijo común. Sólo puede usarse para índices en columnas `CHAR`, `VARCHAR` . Para cadenas de caracteres largas que tienen partes izquierdas de la cadena similares, esto puede reducir significativamente el espacio usado. En el tercer ejemplo dado anteriomente, la cuarta clave tiene una longitud de 10 caracteres y se consigue una reducción del 60% del espacio.

- `Max levels`(Número máximo de niveles)

  La profunidad del árbol-B para esta clave. Tablas grandes con valores de clave grandes obtienen valores altos.

- `Records`Registros

  Número de registros en la tabla.

- `M.recordlength`Longitud de registro media

  La media de la longitud de los registros. Este valor es la longitud de registro para tablas con registros de longitud fija, ya que todos los registros tienen la misma longitud.

- `Packed`Empaquetado

  Espacios vacíos al final de las cadenas de caracteres en MySQL. El valor `Packed` indica el porcentaje de ahorro conseguido haciendo un empaquetado.

- `Recordspace used`Espacio de registro usado

  Porcentaje del fichero de datos usado.

- `Empty space`Espacio vacío

  Porcentaje del fichero de datos sin usar.

- `Blocks/Record`Bloques/Registro

Número medio de bloques por registro (de cuántos enlaces se compone un registro fragmentado). Siempre es 1.0 para una tabla con formato fijo. Este valor debe estar tan cercano a 1.0 como sea posible. Si crece mucho, puede reorganizar la tabla. Consulte Sección 5.8.3.10, “Optimización de tablas”.

- `Recordblocks`Bloques de registro

  Cuántos bloques (enlaces) se usan. Para formato fijo, esto es lo mismo que el número de registros.

- `Deleteblocks`Bloques borrados

  Cuántos bloques (enlaces) están borrados.

- `Recorddata`Datos de registro

  Cuántos bytes en el fichero de datos se usan.

- `Deleted data`Datos borrados

  Cuántos bytes en el fichero de datos están borrados (no usados).

- `Lost space`Espacio perdido

  Si un registro se actualiza a una longitudo más pequeña, se pierde algo de espacio. Este valor es la suma de todas estas pérdidas, en bytes.

- `Linkdata`Datos enlazados

  Cuando se usa el formato de tabla dinámico, los fragmentos de registros se enlazan con punteros (de 4 a 7 bytes cada uno). `Linkdata` es la suma de la cantidad de almacenamiento usada por estos punteros.

Si una tabla se ha comprimido con `myisampack`, `myisamchk -d` muestra información adicional acerca de cada columna de la tabla. Consulte Sección 8.2, “`myisampack`, el generador de tablas comprimidas de sólo lectura de MySQL”, para un ejemplo de esta información y una descripción de lo que significa.

5.9. Uso internacional y localización de MySQL

Esta sección explica como configurar el servidor para utilizar diferentes juegos de caracteres. También explica como establecer la zona horaria del servidor, y activar el soporte para zonas horarias por conexión.

5.9.1. El conjunto de caracteres utilizado para datos y ordenación

Por defecto, MySQL utiliza el juego de caracteres ISO-8859-1 (Latin1), con ordenación de acuerdo a las reglas Suecas/Finlandesas. Estos valores por defecto son aplicables a los Estados Unidos y la mayoría de Europa occidental.

Todas las distribuciones binarias de MySQL se compilan con la opción `--with-extra-charsets=complex`. Esto añade código a todos los programas que les permite manejar el juego de caracteres `latin1` y todos los juegos de caracteres multi-byte desde el binario. Otros juegos de caracteres se cargan desde un archivo de definición, cuando sea necesario.

El juego de caracteres determina qué caracteres se permiten en los nombres. También determina como las cadenas de caracteres se ordenan mediante las cláusulas `ORDER BY` y `GROUP BY` de la sentencia `SELECT`.

Puede cambiar el juego de caracteres con la opción `--default-character-set` cuando inicie el servidor. Los juegos de caracteres disponibles dependen de las opciones `--with-charset=charset` y `--with-extra-charsets=list-of-charsets | complex | all | none` del comando `configure`, y los archivos de configuración de juegos de caracteres enumerados en `SHAREDIR/charsets/Index`. Consulte Sección 2.8.2, “Opciones típicas de `configure`”.

En MySQL 5.0, también puede cambiar la colación del juego de caracteres con la opción `--default-collation` cuando inicie el servidor. La colación debe ser alguna que sea válida para el juego de caracteres por defecto. (Utilice la sentencia `SHOW COLLATION` para determinar qué colaciones están disponibles para cada juego de caracteres.) Consulte Sección 2.8.2, “Opciones típicas de `configure`”.

Si usted cambia el juego de caracteres cuando ejecuta MySQL, esto también podría cambiar la ordenación. En consecuencia, debe ejecutar `myisamchk -r -q --set-character-set=charset` en todas sus tablas, o sus índices podrían no estar ordenados correctamente.

Cuando un cliente se conecta a un servidor MySQL, el servidor indica al cliente cúal es el juego de caracteres por defecto del servidor. El cliente se adapta entonces para utilizar este juego de caracteres en su conexión.

Debería utilizar `mysql_real_escape_string()` cuando escape cadenas de caracteres para una sentencia SQL. `mysql_real_escape_string()` es idéntica a la antigua función `mysql_escape_string()`, excepto que toma el identificador de conexión `MYSQL` como primer parámetro para que pueda tenerse en cuenta el juego de caracteres apropiado cuando se escapan caracteres.

Si el cliente ha sido compilado con rutas diferentes a las del servidor que está instalado, y el usuario que configuró MySQL no incluyó todos los juegos de caracteres en el binario de MySQL, debe indicarle al cliente dónde puede encontrar los juegos de caracteres adicionales que necesita si el servidor ejecuta un juego de caracteres diferente.

Puede hacerlo especificando la opción `--character-sets-dir` para indicar la ruta al directorio donde se encuentra los juegos de caracteres dinámicos de MySQL. Por ejemplo, podría poner lo siguiente en un archivo de opciones:

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

Puede forzar a que el cliente utilice un juego de caracteres específico de la siguiente manera:

```
[client]
default-character-set=charset
```

De todas formas, normalmente, esto no es necesario.

5.9.1.1. Usar el conjunto de caracteres alemán

En MySQL 5.0, el juego de caracteres y la colación se especifican separadamente. Esto significa que si quiere la ordenación alemana, debería seleccionar el juego de caracteres `latin1` y las colaciones `latin1_german1_ci` o `latin1_german2_ci`. Por ejemplo, para iniciar el servidor con la colación `latin1_german1_ci`, utilice las opciones `--character-set-server=latin1` y `--collation-server=latin1_german1_ci`.

Para obtener más información sobre las diferencias entre estas dos colaciones, consulte Sección 10.10.2, “Conjuntos de caracteres de Europa occidental”.

5.9.2. Escoger el idioma de los mensajes de error

Por defecto, `mysqld` produce mensajes de error en inglés, pero también puede producirlos en cualquiera de estos otros idiomas: alemán, checo, danés, eslovaco, español, estonio, francés, griego, holandés, húngaro, italiano, japonés, koreano, noruego, polaco, portugués, rumano, ruso o sueco.

Para iniciar `mysqld` con un idioma particular para los mensajes de error, utilice la opción `--language` o `-L`. El valor de la opción puede ser el nombre de un idioma o la ruta completa al archivo de mensajes de error. Por ejemplo:

```
shell> mysqld --language=swedish
```

O:

```
shell> mysqld --language=/usr/local/share/swedish
```

El nombre del idioma debe ser especificado en minúsculas.

Los archivos de idiomas están almacenados (por defecto) en el directorio `share/LANGUAGE` bajo el directorio base de MySQL.

Para cambiar el archivo de mensajes de error, debería editar el archivo `errmsg.txt`, y después ejecutar el siguiente comando para generar el archivo `errmsg.sys`:

```
shell> comp_err errmsg.txt errmsg.sys
```

Si actualiza a una versión más nueva de MySQL, recuerde repetir los cambios con el nuevo archivo `errmsg.txt`.

5.9.3. Añadir un conjunto de caracteres nuevo

Esta sección explica el procedimiento para añadir nuevos juegos de caracteres a MySQL. Debe tener una distribución de código fuente de MySQL para utilizar estas instrucciones.

Para escoger el método apropiado, decida si el juego de caracteres es simple o complejo:

- Si el juego de caracteres no necesita rutinas especiales de colación para ordenar, y no necesita soporte para cáracteres multi-byte, es simple.
- Si necesita cualquiera de esas características, es complejo.

Por ejemplo, `latin1` y `danish` son juegos de caracteres simples, mientras que `big5` y `czech` son complejos.

En los siguientes procedimientos, el nombre de su juego de caracteres está representado por *`MYSET`*.

Para un juego de caracteres simple, haga lo siguiente:

1. Añada *`MYSET`* al final del archivo `sql/share/charsets/Index`. Asígnele un número único.
2. Cree el archivo `sql/share/charsets/MYSET.conf`. (puede utilizar una copia de `sql/share/charsets/latin1.conf` en la que basarse.)

   La sintaxis de este archivo es muy simple:

   - Los comentarios comienzan con un carácter'#' y continuan hasta el final de la línea.
   - Las palabras están separadas por un número de espacios arbitrario.

- Al definir el juego de caracteres, cada palabra debe ser un número en formato hexadecimal.
- El array `ctype` ocupa las primeras 257 palabras. Los arrays `to_lower[]`, `to_upper[]` y `sort_order[]` ocupan 256 cada uno después de eso.

Consulte Sección 5.9.4, “Los vectores de definición de caracteres”.

3. Añada el nombre del juego de caracteres a las listas `CHARSETS_AVAILABLE` y `COMPILED_CHARSETS` en `configure.in`.
4. Reconfigurar, recompilar, y comprobar.

Para un juego de caracteres complicado, haga lo siguiente:

1. Cree el archivo `strings/ctype-MYSET.c` en la distribución de código fuente MySQL.
2. Añada *`MYSET`* al final del archivo `sql/share/charsets/Index`. Asígnele un número único.
3. Mira uno de los archivos `ctype-*.c` existentes (por ejemplo `strings/ctype-big5.c`) para ver qué necesita definir. Tenga en cuenta que los arrays del archivo deben tener nombres como `ctype_MYSET`, `to_lower_MYSET`, y etc. Estos corresponden a los arrays para un juego de caracteres simple. Consulte Sección 5.9.4, “Los vectores de definición de caracteres”.
4. Cerca del inicio del archivo, ponga un comentario especial como este:

```
/*
 * This comment is parsed by configure to create ctype.c,
 * so don't change it unless you know what you are doing.
 *
 * .configure. number_MYSET=MYNUMBER
 * .configure. strxfrm_multiply_MYSET=N
 * .configure. mbmaxlen_MYSET=N
 */
```

   El programa `configure` utiliza este comentario para incluir el juego de caracteres en la librería MySQL automaticamente.

   Las líneas `strxfrm_multiply` y `mbmaxlen` se explican en las siguientes secciones. Necesita incluirlas sólo si necesita las funciones de colación de cadenas de caracteres o de caracteres multi-bye, respectivamente.

5. Debería crear entonces alguna de las siguientes funciones:
   - `my_strncoll_MYSET()`
   - `my_strcoll_MYSET()`
   - `my_strxfrm_MYSET()`
   - `my_like_range_MYSET()`

   Consulte Sección 5.9.5, “Soporte para colación de cadenas de caracteres”.

6. Añada el nombre del juego de caracteres a las listas `CHARSETS_AVAILABLE` y `COMPILED_CHARSETS` en `configure.in`.
7. Reconfigure, recompile, y compruebe.

El archivo `sql/share/charsets/README` incluye instrucciones adicionales.

Si quiere que el juego de caracteres sea incluido en la distribución MySQL, envíe un parche a la lista de correo `internals` de MySQL. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”.

5.9.4. Los vectores de definición de caracteres

`to_lower[]` y `to_upper[]` son arrays simples que contienen los caracteres en minúsucula y mayúscula correspondientes a cada miembro del juego de caracteres. Por ejemplo:

```
to_lower['A'] debería contener 'a'
to_upper['a'] debería contener 'A'
```

`sort_order[]` es un mapa indicando cómo deberían ser ordenados los caracteres para comparación y propósitos de ordenación. Frecuentemente (pero no en todos los juegos de caracteres) este es idéntico a `to_upper[]`, lo que significa que la ordenación es independiente de la capitalización. MySQL ordena caracteres basándose en los valores de los elementos de `sort_order[]`. Para reglas de ordenación más complicadas, consulte la explicación de colación de cadena de caracteres en Sección 5.9.5, “Soporte para colación de cadenas de caracteres”.

`ctype[]` es un array de valores binarios, con un elemento para cada carácter. (Tenga en cuenta que `to_lower[]`, `to_upper[]`, y `sort_order[]` son idexados por valor del carácter, pero `ctype[]` es indexado por el valor del carácter + 1. Esto es una convención antigua heredada para poder gestionar `EOF`.)

Puede entontrar las siguientes definiciones de máscaras de bits en `m_ctype.h`:

```
#define _U      01      /* Uppercase */
#define _L      02      /* Lowercase */
#define _N      04      /* Numeral (digit) */
#define _S      010     /* Spacing character */
#define _P      020     /* Punctuation */
#define _C      040     /* Control character */
#define _B      0100    /* Blank */
#define _X      0200    /* heXadecimal digit */
```

El valor de `ctype[]` para cada carácter debería ser la unión de los valors de máscara aplicables que describan al carácter. Por ejemplo, `'A'` es un carácter en mayúsculas (`_U`) y también es un dígito hexadecimal (`_X`), así que `ctype['A'+1]` debería tener el valor:

```
_U + _X = 01 + 0200 = 0201
```

5.9.5. Soporte para colación de cadenas de caracteres

Si las reglas de ordenación para su lenguaje son demasiado complejas para ser gestionadas con la sencilla tabla `sort_order[]`, necesita utilizar las funciones de colación de cadenas de caracteres.

La mejor documentación para este propósito son los juegos de caracteres existentes. Mire los juegos de caracteres `big5`, `czech`, `gbk`, `sjis`, y `tis160` en busca de ejemplos.

Debe especificar el valor `strxfrm_multiply_`*`MYSET=N`* en el comentario especial al principio del archivo. *`N`* debería tener el máximo ratio en que pueda crecer las cadenas de caracteres durante `my_strxfrm_`*`MYSET`* (debe ser un entero positivo).

5.9.6. Soporte de caracteres multi-byte

Si quiere añadir soporte para un juego de caracteres nuevo que incluye caracteres multi-byte, necesita utilizar las funciones de carácter multi-byte.

La mejor documentación para este propósito son los juegos de caracteres existentes. Mire los juegos de caracteres `euc_kr`, `gb2312`, `gbk`, `sjis`, y `ujis` para encontrar ejemplos. Estos se encuentran en el archivo `ctype-charset.c` del directorio `strings`.

Debe especificar el valor de `mbmaxlen_MYSET=N` en el comentario especial al inicio del archivo fuente. *`N`* debería tener el tamaño en bytes del carácter más grande en el juego.

5.9.7. Problemas con conjuntos de caracteres

Si trata de utilizar un juego de caracteres que no está compilado en el binario, podría encontrarse con los siguientes problemas:

- Su programa tiene la ruta incorrecta de almacenamiento de los juegos de caracteres. (Por defecto `/usr/local/mysql/share/mysql/charsets`). Esto puede arreglarse utilizando la opción `--character-sets-dir` para ejecutar el programa en cuestión.
- El juego de caracteres es un juego multi-byte que no puede cargarse dinámicamente. En este caso, debe recompilar el programa con soporte para el juego de caracteres.
- El juego de caracteres es un juego de caracteres dinámico, pero no tiene un fichero de configuración para él. En este caso, debería instalar el archivo de configuración de una nuevo distribución MySQL.
- Si su archivo `Index` no contiene el nombre del juego de caracteres, su programa mostrará el siguiente mensaje de error:

```
ERROR 1105: File '/usr/local/share/mysql/charsets/?.conf'
not found (Errcode: 2)
```

  En este caso, debería obtener un nuevo archivo `Index` o bien añadir el nombre manualmente de cualquier juego de caracteres que falte en el archivo actual.

Para las tablas `MyISAM`, puede comprobar el nombre del juego de caracteres y el número de una tabla con `myisamchk -dvv tbl_name`.

5.9.8. Soporte de zonas horarias en el servidor MySQL

En MySQL 5.0, el servidor mantiene varios ajustos de zonas horarias:

- La zona horaria del sistema. Cuando el servidor se inicia, intenta determinar la zona horaria de la máquina que lo aloja y lo utiliza para establecer la variable de sistema `system_time_zone`.
- La zona horaria del servidor. La variable global del sistema `time_zone` indica la zona horaria en que el servidor está operando actualmente. El valor inicial es `'SYSTEM'`, que indica que la zona horaria del servidor es la misma que la del sistema. El valor inicial puede ser especificado explícitamente con la opción `--default-time-zone=timezone`. Si tiene el privilegio `SUPER`, puede establecer el valor global durante la ejecución, con esta sentencia:

```
mysql> SET GLOBAL time_zone = timezone;
```

- Zonas horarias por conexión. Cada cliente que se conecta tiene su propio ajuste de zona horaria, otorgado por la variable de sesión `time_zone`. Inicialmente esta es la misma que la variable global `time_zone`, pero puede reestablecerse con esta sentencia:

```
mysql> SET time_zone = timezone;
```

Los valores actuales de las zonas horarias globales y por conexión pueden consultarse así:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

Los valores de *timezone* pueden darse como cadenas de caracteres indicando una variación con respecto a UTC, como por ejemplo '+10:00' o '-6:00'. Si las tablas relativas a las zonas horarias han sido creadas y rellenadas, también puede utilizar zonas horarias por su nombre, como 'Europe/Helsinki', 'US/Eastern', o 'MET'. El valor 'SYSTEM' indica que la zona horaria debería ser la misma que la del sistema. Los nombres de zonas horarias no distinguen capitalización (mayúsculas/minúsculas).

El procedimiento de instalación de MySQL crea las tablas de zonas horarias en la base de datos mysql, pero no las carga. Debe hacerlo usted manualmente. (Si está actualizando a MySQL 4.1.3 o superior desde una versión anterior, debe crear las tablas actualizando su base de datos mysql. Utilice las instrucciones que puede encontrar en Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.)

Si su sistema tiene su propia base de datos de información de zonas (el conjunto de archivos que describen zonas horarias), debería utilizar el programa mysql_tzinfo_to_sql para rellenar las tablas de zonas horarias. Ejemplos de esos sistemas son algunos como Linux, FreeBSD, Sun Solaris, y Mac OS X. Una localización probable para estos ficheros es el directorio /usr/share/zoneinfo. Si su sistema no tiene una base de datos de información de zonas, puede utilizar un paquete descargable descrito más adelante en esta sección.

El programa mysql_tzinfo_to_sql se utiliza para cargar las tablas de zonas horarias. En la línea de comandos, pase la ruta del directorio de los archivos de información de zonas a mysql_tzinfo_to_sql y envíe la salida al programa mysql. Por ejemplo:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

mysql_tzinfo_to_sql lee los archivos de zona horaria de su sistema y genera sentencias SQL a partir de ellos. mysql procesa esas sentencias y las carga en las tablas de zona horaria.

mysql_tzinfo_to_sql también puede utilizarse para cargar un solo archivo de zona horaria.

Para cargar un solo archivo de zona horaria *tz_file* que corresponde a una zona horaria *tz_name*, invoque a mysql_tzinfo_to_sql de esta manera:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

Si su zona horaria necesita tener en cuenta segundos de diferencia, inicialice la información sobre segundos de diferencia de esta manera, donde *tz_file* es el nombre de su archivo de zona horaria:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

Si su sistema no tiene una base de datos de información de zonas (por ejemplo, Windows o HP-UX), puede utilizar el paquete preconfigurado de tablas de zonas horarias que está disponible en http://dev.mysql.com/downloads/timezones.html. Este paquete contiene los archivos .frm, .MYD, y .MYI para las tablas de zonas horarias MyISAM. Estas tablas deben pertenecer a la base de datos mysql, así que debe depositar los archivos en el subdirectorio mysql de su directorio de datos de MySQL. Debería apagar el servidor mientras hace esto.

¡Cuidado! Por favor, no utilice el paquete descargable si su sistema ya tiene una base de datos de información de zonas. Utilice en cambio la utilidad mysql_tzinfo_to_sql. Si no es así, podría causar una diferencia en el manejo de fechas entre MySQL y otras aplicaciones de su sistema.

Para información sobre los ajustes de zonas horarias en configuraciones de replicación consulte Sección 6.7, “Características de la replicación y problemas conocidos”.

5.10. Los ficheros de registro (log) de MySQL

MySQL tiene varios archivos de registro diferentes que pueden ayudarle a encontrar lo que está ocurriendo en `mysqld`:

Archivo de registro	Tipo de información registrado en el archivo
El registro de error	Registra problemas encontrados iniciando, ejecutando, o parando `mysqld`.
El registro de consultas	Registra las conexiones de clientes establecidas, y las sentencias ejecutadas.
El registro de actualizaciones The update log	Registra las sentencias que cambian datos. Este registro está ya en desuso.
El registro binario	Registra todas las sentencias que cambian datos. También utilizado para replicación.
El registro de lentitud	Registra todas las sentencias que tardarón más de `long_query_time` segundos en ejecutarse, o no utilizaron índices.

Por defecto, todos los registros son creados en el directorio de datos de `mysqld`. Puede forzar a `mysqld` a que cierre y reabra los archivos de registro (o en algunos casos, cambiar a un nuevo registro) mediante el volcado de registros. El volcado de registros ocurre cuando ejecuta la sentencia `FLUSH LOGS` o el comando `mysqladmin flush-logs` or `mysqladmin refresh`. Consulte Sección 13.5.5.2, “Sintaxis de FLUSH”.

Si está utilizando las capacidades de replicación de MySQL, los servidores esclavos de replicación mantienen unos registros adicionales llamados registros relay. Estos se explican en Capítulo 6, *Replicación en MySQL*.

5.10.1. El registro de errroes (Error Log)

El archivo de registro de errores contiene información que indica cuando se ha iniciado y parado `mysqld` y también si ha ocurrido algún error crítico mientras el servidor se estaba ejecutando.

Si `mysqld` termina inesperadamente y `mysqld_safe` necesita reiniciarlo, `mysqld_safe` escribe un mensaje `restarted mysqld` en el registro de errores. Si `mysqld` se da cuenta de que hay una tabla que necesita ser automáticamente comprobada o reparada, escribe un mensaje al registro de errores.

En algunos sistemas operativos, el registro de errores contiene un volcado de la pila si `mysqld` muere. El volcado puede ser utilizado para determinar cuando `mysqld` murió. Consulte Sección D.1.4, “Usar stack trace”.

En MySQL 5.0, puede especificar dónde quiere que `mysqld` almacene el registro de errores con la opción `--log-error[=file_name]`. Si no se proporciona ningún valor para *`file_name`*, `mysqld` utiliza el nombre *`host_name`*`.err` y escribe el archivo en el directorio de datos. Si ejecuta `FLUSH LOGS`, el registro de errores es renombrado con el sufijo `-old` y `mysqld` crea un nuevo archivo de registro.

Si no especifica `--log-error`, o (en Windows) no utiliza la opción `--console`, los errores se escriben en stderr, la salida estándar de erroes. Usualmente esto es su terminal.

En Windows, la salida de errores es siempre escrita al archivo `.err` si no se especifica la opción `--console`.

5.10.2. El registro general de consultas

Si quiere saber qué pasa en `mysqld`, debe iniciarlo con la opción `--log[=file_name]` o `-l [file_name]`. Si no se da un valor para *`file_name`*, el nombre por defecto es *`host_name`*`.log`. Esto registra todas las conexiones y sentencias a un archivo. Este registro puede ser muy útil cuando sospeche que hay un error en un cliente y quiera saber exactamente qué envió el cliente a `mysqld`.

`mysqld` escribe las sentencias al registro de consultas en el orden en que las recibe. Este orden puede ser diferente del de ejecución. Esto es aquí diferente que en el registro de actualizaciones o el registro binario, que son escritos tras la ejecución de la sentencia, pero antes de que se libere cualquier bloqueo. (El registro de consultas también contiene todas las sentencias, mientras que el registro binario no contiene sentencias que solo seleccionen datos.)

Los reinicios del servidor y volcado de registros no provocan que se genere un nuevo archivo de registro de consultas general (aunque el volcado lo cierra y lo reabre). En Unix, puede renombrar el archivo y crear uno nuevo utilizando los siguientes comandos:

```
shell> mv hostname.log hostname-old.log
shell> mysqladmin flush-logs
shell> cp hostname-old.log to-backup-directory
shell> rm hostname-old.log
```

En Windows, no puede renombrar el archivo de registro mientras el servidor lo tiene abierto. Debe parar el servidor y renombrar el registro. Después, reinicie el servidor para crear el nuevo registro.

5.10.3. El registro binario (Binary Log)

El registro binario contiene toda la información que está disponible en el registro de actualizaciones, en un formato más eficiente y de una manera que es segura para las transacciones.

El registro binario contiene todas las sentencias que han actualizado datos o podrían haberlo hecho (por ejemplo, un `DELETE` que no encontró filas concordantes). Las sentencias se almacenan en la forma de "eventos" que describen las modificaciones.

Atención: El registro binario ha reemplazado al antiguo registro de actualizaciones, que ya no está disponible a partir de MySQL 5.0.

El registro binario también contiene información sobre cuanto ha tardado cada sentencia que actualizó la base de datos. No contiene sentencias que no hayan modificado datos. Si quiere registrar todas las sentencias (por ejemplo, para identificar una sentencia problemática) debería utilizar el registro de consultas general. Consulte Sección 5.10.2, "El registro general de consultas".

El propósito principal del registro binario es el de actualizar la base de datos durante una operación de recuperación tan completamente como sea posible, porque el registro binario contiene todas las actualizaciones hechas tras la copia de seguridad.

El registro binario también se utiliza en los servidores maestros de replicación como recordatorio de las sentencias que deben ser enviadas a los servidores esclavos. Consulte Capítulo 6, *Replicación en MySQL*.

Ejecutar el servidor con el registro binario activado hace que el rendimiento baje un 1%. Aún así, los beneficios del registro binario para las operaciones de restauración y el hecho de permitirnos poder establecer replicación generalmente son superiores a este decremento de rendimiento.

Cuando se ha iniciado con la opción `--log-bin[=file_name]` `mysqld` escribe un archivo de registro que contiene todos los comandos SQL que actualizan datos. Si no se da ningún valor para *`file_name`*, el valor por defecto es el nombre de la máuqina host seguido por `-bin`. Si se da el nombre del archivo, pero ninguna ruta, el archivo se escribe en el directorio de datos. Se recomienda especificar un nombre de archivo, consulte Sección A.8.4, "Cuestiones abiertas en MySQL" para ver el motivo.

Si usted proporciona una extensión en el nombre del registro (por ejemplo, `--log-bin=file_name.extension`), la extensión se ignora y elimina sin aviso.

`mysqld` agraga una extensión numérica a el nombre del registro binario. Este número se incrementa cada vez que se inicia el servidor o se vuelcan los registros. También se crea un nuevo registro binario cuando el actual llega al tamaño especificado en `max_binlog_size`. Un registro binario puede llegar a ser más grande de `max_binlog_size` si está utilizando transacciones grandes: Una transacción se escribe en el registro de una sola pieza, nunca se parte entre diferentes registros.

Para poder averiguar qué archivos de registro binario diferentes han sido utilizados, `mysqld` también crea un archivo de índice de los registros binarios que contiene los nombres de todos los archivos de registro binario utilizados. Por defecto éste tiene el mismo nombre que el archivo de registro binario, con la extensión `'.index'`. Puede cambiar el nombre del archivo de índice con la opción `--log-bin-index[=file_name]`. No debería editar este archivo manualmente mientras `mysqld` se está ejecutando; hacerlo podría confundir a `mysqld`.

Puede borrar todos los archivos de registro binario con la sentencia `RESET MASTER`, o sólo algunos de ellos con `PURGE MASTER LOGS`. Consulte Sección 13.5.5.5, “Sintaxis de RESET” y Sección 13.6.1, “Sentencias SQL para el control de servidores maestros”.

El formato del registro binario tiene algunas limitaciones conocidas que pueden afectar a la recuperación de copias de seguridad. Consulte Sección 6.7, “Características de la replicación y problemas conocidos”.

El registro binario de procedimientos almacenados y disparadores se hace tal como se explica en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

Puede utilizar las siguientes opciones de `mysqld` para determinar lo que se registra en el registro binario. Vea también la explicación que sigue a esta lista de opciones.

- `--binlog-do-db=db_name`

  Le dice al maestro que debería registrar los cambios en el registro binario si la base de datos actual (es decir, la que está seleccionada mediante `USE`) es *`db_name`*. Todos las otras bases de datos que no sean mencionadas explícitamente son ignoradas. Si utiliza esto, debería asegurarse de que sólo hace cambios en la base de datos actual.

  Tenga en cuenta que hay una excepción a esto en lo que respecta a las sentencias `CREATE DATABASE`, `ALTER DATABASE`, y `DROP DATABASE`, que utilizan la base de datos manipulada en vez de la actual para decidir si deberían registrar la sentencia.

  Un ejemplo de algo que no funciona como podría esperarse: Si el servidor se inició con `binlog-do-db=sales`, y usted ejecuta `USE prices; UPDATE sales.january SET amount=amount+1000;`, esta sentencia no llega a escribirse en el registro binario.

- `--binlog-ignore-db=db_name`

  Le dice al maestro que las actualizaciones donde la base de datos actual (es decir, la que ha sido seleccionada mediante `USE`) es *`db_name`* no deberían ser almacenadas en el registro binario. Si utiliza esto, debería asegurarse de que solo hace actualizaciones en la base de datos actual.

  Un ejemplo de algo que no funciona como podría esperarse: Si el servidor se inició con `binlog-ignore-db=sales`, y ejecuta `USE prices; UPDATE sales.january SET amount=amount +1000;`, esta sentencia no se escribe en el registro binario.

  De la misma forma que en el caso de `--binlog-do-db`, hay una excepción para las sentencias `CREATE DATABASE`, `ALTER DATABASE`, y `DROP DATABASE`, que utilizan la base de datos manipulada para decidir si deben registrar la sentencia, en vez de la base de datos actual.

Para registrar o ignorar múltiples bases de datos, utilice múltiples opciones, especificando la opción apropiada una vez para cada base de datos.

Las reglas para registrar o ignorar actualizaciones en el registro binario son evaluadas de acuerdo a las siguientes normas. Tenga en cuenta que hay una excepción para las sentencias `CREATE DATABASE`, `ALTER DATABASE`, y `DROP DATABASE`. En esos casos, la base de datos que está siendo *creada, alterada, o eliminada* reemplaza a la base de datos actual en las reglas siguientes.

1. ¿Hay alguna regla `binlog-do-db` o `binlog-ignore-db`?
   - No: Escribir la sentencia al registro binario y salir.
   - Sí: Proceder al siguiente paso.
2. Hay algunas reglas (`binlog-do-db` o `binlog-ignore-db` o ambas). ¿Hay una base de datos actual (¿ha sido seleccionada alguna base de datos mediante `USE`?)?
   - No: *No* escribir la sentencia, y salir.
   - Sí: Proceder al siguiente paso.
3. Hay una base de datos actual. ¿Hay alguna regla `binlog-do-db`?
   - Sí: ¿Concuerda la base de datos con alguna de las reglas `binlog-do-db`?
     - Sí: Escribir la sentencia y salir.
     - No: *No* escribir la sentencia, y salir.
   - No: Proceder al siguiente paso.
4. Hay alguna regla `binlog-ignore-db`. ¿Concuerda la base de datos con alguna de las reglas `binlog-ignore-db`?
   - Sí: No escribir la sentencia, y salir.
   - No: Escribir la sentencia y salir.

Por ejemplo, un esclavo ejecutándose con sólo `binlog-do-db=sales` no escribe en el registro binario ninguna sentencia en la que la base de datos actual sea diferente de `sales` (es decir, a veces `binlog-do-db` puede significar ``ignora otras bases de datos'').

Si está utilizando replicación, debería no borrar los archivos de registros binarios viejos hasta que esté seguro de que ningún esclavo los necesita. Una manera de averiguarlo es hacer `mysqladmin flush-logs` una vez al día y borrar cualquier registro que tenga más de tres días de antigüedad. Puede borrarlos manualmente, o mejor aún mediante `PURGE MASTER LOGS` (consulte Sección 13.6.1, “Sentencias SQL para el control de servidores maestros”), que además también actualiza de manera segura el archivo de índice de registros binarios (y que puede recibir parámetros de fecha).

Un cliente con el privilegio `SUPER` puede desactivar el registro binario de sus propias sentencias utilizando una sentencia `SET SQL_LOG_BIN=0`. Consulte Sección 13.5.3, “Sintaxis de SET”.

Puede examinar el archivo de registro binario con la utilidad `mysqlbinlog`. Esto podría ser útil cuando usted quiera reprocesar sentencias almacenadas en el registro. Por ejemplo, puede actualizar un servidor MySQL desde el registro binario de la siguiente manera:

```
shell> mysqlbinlog log-file | mysql -h server_name
```

Consulte Sección 8.5, “La utilidad `mysqlbinlog` para registros binarios” para obtener más información sobre la utilidad `mysqlbinlog` y su utilización.

Si usted está utilizando transacciones, debería utilizar el registro binario de MySQL para hacer las copias de seguridad, en vez del antiguo registro de actualizaciones.

El registro binario se hace inmediatamente después de que se completa una consulta, pero antes de que se libere cualquier bloqueo o se haga ningún commit. Esto asegura que el registro está almacenado en el orden de ejecución.

Las actualizaciones a las tablas no-transaccionales se almacenan en el registro binario inmediatamente después de su ejecución. Para las tablas transaccionales como las tablas `BDB` o `InnoDB`, todas las actualizaciones (`UPDATE`, `DELETE`, o `INSERT`) que cambian alguna tabla son almacenadas en cache hasta que se recibe una sentencia `COMMIT` en el servidor. En ese momento `mysqld` escribe la transacción completa al registro binario antes de que se ejecute el `COMMIT`. Cuando el flujo de ejecución que gestiona la transacción comienza, reserva un buffer de tamaño `binlog_cache_size` para almacenar consultas. Si una sentencia es más grande de esto, el flujo abre un archivo temporal para almacenar la transacción. El archivo temporal se borra cuando acaba el flujo.

La variable de estado `Binlog_cache_use` muestra el número de transacciones que han utilizado este buffer (y posiblemente un archivo temporal) para almacenar sentencias. La variable de estado `Binlog_cache_disk_use` muestra cuantas de esas transacciones llegaron realmente a utilizar un archivo temporal. Estas dos variables pueden utilizarse para establecer el valor de `binlog_cache_size` y evitar el uso de archivos temporales.

El tamaño `max_binlog_cache_size` (por defecto 4GB) se puede utilizar para restringir el tamaño total utilizado para almacenar una transacción con múltiples sentencias. Si una transacción es más grande que esto, falla y se hace un rollback.

Si usted está utilizando el registro de actualizaciones o el binario, las inserciones concurrentes se convierten a inserciones normales cuando se utiliza `CREATE ... SELECT` o `INSERT ... SELECT`. Esto se hace así para asegurarse de que se puede reconstruir una copia exacta de sus tablas aplicando los registros a una copia de seguridad.

Tenga en cuenta que el formato del registro binario es diferente en MySQL 5.0 al de versiones anteriores de MySQL, debido a mejoras en la replicación. Consulte Sección 6.5, “Compatibilidad entre versiones de MySQL con respecto a la replicación”.

Por defecto, el registro binario no se sincroniza con el disco en cada escritura. Así que si el sistema operativo o la máquina (no únicamente el servidor MySQL) falla, existe la posibilidad de que las últimas sentencias del registro binario se pierdan. Para prevenir esto, puede hacer que el registro binario se sincronice con el disco tras cada *`N`* escrituras, con la variable global `sync_binlog` (siendo 1 el valor más seguro, pero también el más lento). Consulte Sección 5.3.3, “Variables de sistema del servidor”. Aún con el valor de `sync_binlog` establecido en 1, existe una posibilidad de que haya una inconsistencia entre las tablas y el registro binario en el caso de fallo. Por ejemplo, si está utilizando tablas `InnoDB`, y el servidor MySQL procesa una sentencia `COMMIT`, escribe la transacción completa al registro binario, y después la envía a `InnoDB`. Si el fallo se produce entre estas dos operaciones, al reiniciar la transacción es rechazada por `InnoDB`, pero todavía existe en el registro binario. Este problema puede resolverse con la opción `--innodb-safe-binlog`, que añade consistencia entre el contenido de las tablas `InnoDB` y el registro binario.

Para que esta opción proporcione un grado mayor de seguridad, el servidor MySQL debe estar también configurado para sincronizar a disco, en cada transacción, el registro binario (`sync_binlog=1`) y (lo que es cierto por defecto) los registros `InnoDB`. El efecto de esta opción es que al reiniciar tras un fallo, o tras hacer un rollback de transacciones, el servidor MySQL elimina las transacciones `InnoDB` que han sufrido una cancelación del registro binario. Esto asegura que el registro binario refleje los datos exactos que hay

en las tablas `InnoDB` y, así, el esclavo permanece sincronizado con el maestro (no recibe una sentencia que ha sido cancelada).

Tenga en cuenta que `--innodb-safe-binlog` se puede utilizar aún cuando el servidor MySQL actualice otros motores de almacenamiento que no sean `InnoDB`. Sólo las sentencia/transacciones que afecten a tablas `InnoDB` son candidatas a ser eliminadas del registro binario durante la recuperación de un fallo de `InnoDB`. Si durante la recuperación el servidor MySQL descubre que el registro binario es más corto de lo que debería ser (es decir, le falta al menos una transacción `InnoDB` realizada con éxito), lo que no debería ocurrir con `sync_binlog=1` y el disco/sistema de archivos hace una sincronización real cuando se le pide (algunos no lo hacen), muestra un mensaje de error ("The binary log <name> is shorter than its expected size"). En este caso el registro binario no es correcto, y la replicación debería reiniciarse desde una nueva imagen de los datos del maestro.

La escrituras al archivo del registro binario y al de índice de registros son gestionados de la misma manera que las escrituras a las tablas `MyISAM`. Consulte Sección A.4.3, “Cómo se comporta MySQL ante un disco lleno”.

5.10.4. El registro de consultas lentas (Slow Query Log)

Cuando se inicia con la opción `--log-slow-queries[=`*`file_name`*`]`, `mysqld` escribe un archivo de registro que contiene todos las sentencias SQL que llevaron más de `long_query_time` segundos para ejecutarse completamente. El tiempo para adquirir los bloqueos de tabla iniciales no se cuenta como tiempo de ejecución.

Si no se da ningún valor a *`file_name`*, el nombre por defecto es el nombre de la máquina host con el sufijo `-slow.log`. Si se da un nombre de archivo, pero no como ruta absoluta, el archivo se escribe en el directorio de datos.

Una sentencia se registra en el registro de consultas lentas después de que haya sido ejecutada y todos los bloqueos liberados. El orden de registro puede diferir del de ejecución.

El registro de consultas lentas se puede utilizar para encontrar consultas que tomen excesivo tiempo y sean por tanto candidatos a optimización. De cualquier modo, examinar un registro de consultas lentas puede convertirse en una tarea difícil. Para hacerlo más simple, puede procesar el registro de consultas lentas utilizando el comando `mysqldumpslow` que le ofrecerá un resumen de las sentencias que aparecen en el registro.

En el registro de consultas lentas de MySQL 5.0, las consultas lentas que no utilizan índices se registran igual que las que sí utilizan. Para prevenir que las consultas que no utilizan índices sean registradas en el registro de consultas lentas, utilice la opción `--log-short-format`. Consulte Sección 5.3.1, “Opciones del comando `mysqld`”.

En MySQL 5.0, la opción `--log-slow-admin-statements` del servidor le permite demandar el registro de sentencias administrativas lentas como `OPTIMIZE TABLE`, `ANALYZE TABLE`, y `ALTER TABLE` sean escritas en el registro de consultas lentas.

Las consultas gestionadas por la cache de consultas no son añadidas al registro de consultas lentas, ni tampoco las consultas que no se beneficien de la presencia de un índice porque la tabla tenga cero o una filas.

5.10.5. Mantenimiento de ficheros de registro (log)

El servidor MySQL puede crear un número de archivos de registro diferente que faciliten el ver que está pasando. Consulte Sección 5.10, “Los ficheros de registro (log) de MySQL”. De cualquier modo, debe limpiar estos archivos regularmente para asegurarse de que no ocupan demasiado espacio.

Cuando se utiliza MySQL con el registro activado, debería hacer copias de seguridad y eliminar los registros viejos de vez en cuando, y decirle a MySQL que comience a registrar en archivos nuevos. Consulte Sección 5.8.1, “Copias de seguridad de bases de datos”.

En una instalación Linux (Red Hat), puede utilizar el script `mysql-log-rotate` para esto. Si instaló MySQL desde una distribución RPM, el script debería haber sido instalado automáticamente. Debería tener cuidado con este script si está utilizando el registro binario para replicación; no elimine los registros binarios hasta que tenga la certeza de que sus contenidos han sido procesados por todos los esclavos.

En otros sistemas, debe instalar un script corto usted mismo desde `cron` o algo equivalente para gestionar los archivos de registros.

Puede forzar a MySQL para que comience a utilizar archivos de registro nuevos usando `mysqladmin flush-logs` o con la sentencia SQL `FLUSH LOGS`.

Una operación de volcado de registros hace lo siguiente:

- Si se está utilizando registro (`--log`) o registro de consultas lentas (`--log-slow-queries`), cierra y reabre el archivo de registro (`mysql.log` y `` `hostname`-slow.log `` por defecto).
- Si se está utilizando registro de actualizaciones (`--log-update`) o registro binario (`--log-bin`) cierra el registro, y abre un nuevo archivo de registro con un número de secuencia superior.

Si está utilizando tan solo el registro de actualizaciones, tan solo tiene que renombrar el archivo de registro y posteriormente volcar los registros antes de hacer una copia de seguridad. Por ejemplo, puede hacer algo como esto:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

Luego, haga una copia de seguridad y elimine `mysql.old`.

5.11. Ejecutar más de un servidor MySQL en la misma máquina

En algunos casos, podría querer ejecutar múltiples servidores `mysqld` en la misma máquina. Quizá quiera probar una nueva versión de MySQL dejando la configuración de producción sin cambios. O quizá quiera dar acceso para diferentes usuarios a diferentes servidores `mysqld` que ellos mismos puedan administrar. (Por ejemplo, podría ser un proeedor de servicios de Internet que proporcione instalaciones de MySQL independientes para cada cliente.)

Para ejecutar múltiples servidores en una única máquina, cada servidor tiene que tener valores únicos para los diferentes parámetros de operación. Estos se pueden establecer en la línea de comandos o en archivos de opciones. Consulte Sección 4.3, “Especificar opciones de programa”.

Al menos las siguientes opciones deben ser diferentes para cada servidor:

- `--port=`*`port_num`*

  `--port` controla el número de puerto de las conexiones TCP/IP.

- `--socket=`*`path`*

  `--socket` controla la ruta del archivo socket de Unix y el nombre de la named pipe en Windows. En Windows, es necesario especificar diferentes nombres de pipe solo para los servidores que soporten conexiones mediante named pipe.

- `--shared-memory-base-name=`*`name`*

Esta opción, actualmente, se utiliza sólo en Windows. Designa el nombre de memoria compartida utilizado por un servidor Windows para permitir a los clientes conectarse mediante memoria compartida.

- `--pid-file=path`

  Esta opción se utiliza únicamente en Unix. Indica el nombre del archivo en el que el servidor escribe su ID de proceso.

Si utiliza las siguientes opciones de archivo de registro, deben ser diferentes para cada servidor:

- `--log=path`
- `--log-bin=path`
- `--log-update=path`
- `--log-error=path`
- `--bdb-logdir=path`

Las opciones de archivo de registro se explican en Sección 5.10.5, “Mantenimiento de ficheros de registro (log)”.

Para un mejor rendimiento, puede especificar las siguientes opciones a cada servidor, de manera que se distribuya la carga entre discos físicos:

- `--tmpdir=path`
- `--bdb-tmpdir=path`

Tener diferentes directorios temporales también es una buena práctica, para hacer más fácil determinar qué servidor MySQL creó cualquier archivo temporal.

Generalmente, cada servidor debería utilizar un directorio de datos diferente, lo que se especifica con la opción `--datadir=path`.

Atención: Normalmente no debería tener dos servidores que actualicen los datos de la misma base de datos. Esto podría llevar a obtener sorpresas indeseadas si su sistema operativo no tiene soporte para bloqueos sin posibilidad de fallo. Si (a pesar de este aviso) ejecuta múltiples servidores que utilicen el mismo directorio de datos y tienen el registro activado, debería utilizar las opciones adecuadas para especificar nombres de archivos de registro que sean únicos para cada servidor. De otra manera, lo servidores intentan registrar en los mismos archivos. Por favor, tengan cuenta que este tipo de configuración sólo funciona con tablas `MyISAM` y `MERGE`, y no con ningún otro de los motores de almacenamiento.

Este aviso en contra de compartir un directorio de datos entre servidores también se aplica en entornos NFS. Permitir que múltiples servidores MySQL accedan a un directorio común es una *muy mala idea*.

- El principal problema es que NFS es un cuello de botella de velocidad. No está diseñado para un uso tal.
- Otro riesgo con NFS es que tiene que encontrar una manera de asegurarse que dos o más servidores no se interfieran unos con otros. Usualmente, el bloqueo de archivos de NFS está gestionado por el demonio `lockd`, pero de momento no hay ninguna plataforma que realice bloqueos 100% seguros en todas las situaciones.

Facilítese las cosas: Olvídese de compartir un directorio de datos entre servidores sobre NFS. Una solución mejor es tener una máquina que tenga diferentes CPUs y utilizar un sistema operativo que gestione los hilos de ejecución eficientemente.

Si tiene múltiples instalaciones de MySQL en diferentes lugares, normalmente puede especificar el directorio base de instalación para cada uno con la opción `--basedir=path`, de manera que cada servidor utilice un directorio de datos, archivo de registro, y archivo de PID diferentes. (Los valores por defecto de todos ellos son determinados en relación al directorio base). En ese caso, las únicas opciones adicionales que necesita especificar son las opciones `--socket` y `--port`. Por ejemplo, suponga que debe instalar diferentes versiones de MySQL utilizando distribuciones binarias en archivos `tar`. Se instalan en diferentes lugares, así que puede iniciar el servidor de cada instalación utilizando el comando `bin/mysqld_safe` bajo su correspondiente directorio base. ¡`mysqld_safe` determina la opción `--basedir` apropiada para pasarle a `mysqld`, y usted sólo necesita especificar las opciones `--socket` y `--port` a `mysqld_safe`.

Tal como se explica en las siguientes secciones, es posible iniciar servidores adicionales mediante el establecimiento de sus variables de entorno, o especificando opciones de línea de comandos apropiadas. Aún así, si necesita ejecutar servidores múltiples de manera permanente, es más conveniente utilizar archivos de opciones para especificar a cada servidor las opciones que deben ser únicas para él.

5.11.1. Ejecutar varios servidores en Windows

Puede ejecutar varios servidores en Windows iniciándolos manualmente desde la línea de comandos, cada uno con sus parámetros apropiados. En sistemas basados en Windows NT, tiene la opción de instalar varios servidores como servicios Windows y ejectuarlos de ese modo. Las instrucciones generales para ejecutar servidores MySQL desde la línea de comandos o como servicios se dan en Sección 2.3, “Instalar MySQL en Windows”.Esta sección describe cómo asegurarse que cada servidor arranca con distintos valores en las opciones de arranque que deban ser únicas para cada servidor, tales como el directorio de datos. Estas opciones se describen en Sección 5.11, “Ejecutar más de un servidor MySQL en la misma máquina”.

5.11.1.1. Arrancar múltiples servidores desde la línea de comandos en Windows

Para arrancar manualmente múltiples servidores desde la línea de comandos, puede especificar las opciones apropiadas en la línea de comandos o en un fichero de opciones. Es más conveniente especificar las opciones en un fichero, pero es necesario asegurarse que cada servidor obtiene su propio conjunto de opciones. Para ello, cree un fichero de opciones para cada servidor y arranque el servidor con la opción `--defaults-file` cuando lo ejecute.

Suponga que desea ejectuar `mysqld` en el puerto 3307 con el directorio de datos en `C:\mydata1`, y `mysqld-max` en el puerto 3308 con un directorio de datos en `C:\mydata2`. (Para ello, asegúrese antes de copiar la base de datos `mysql` que contiene las tablas de permisos.)

A continuación cree dos ficheros de opciones. Por ejemplo, cree un fichero llamado `C:\my-opts1.cnf` que tenga este contenido:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

Cree un segundo fichero llamado `C:\my-opts2.cnf` con este contenido:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

A continuación, arranque cada servidor con su fichero de opciones:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-max --defaults-file=C:\my-opts2.cnf
```

En NT, cada servidor arranca en primer plano (no aparece un nuevo prompt hasta que acaba el servidor); necesitará ejectura estos dos comandos en una consola separada.

Para parar los servidores, debe conectarse al puerto apropiado:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Los servidores configurados tal y como se ha descrito permiten a los clientes la conexión mediante TPC/IP. Si su versión de Windows soporta named pipes y quiere permitir este tipo de conexiones, use los servidores `mysqld-nt` o `mysqld-max-nt` y especifique opiones que permitan la named pipe y especifique su nombre. Cada servidor que soporte conexiones mediante named pipes debe usear un nombre único. Por ejemplo, el fichero `C:\my-opts1.cnf` puede escribirse así:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

A continuación, arranque el servidor así:

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

Modique `C:\my-opts2.cnf` similarmente para que lo use el segundo servidor.

5.11.1.2. arrancar varios servidores Windows como servicios

En sistemas basados en NT, un servidor MySQL puede correr como un servicio de Windows. Los procedimientos para instalar, controlar, y eliminar un servico MySQL se describen en Sección 2.3.12, “Arrancar MySQL como un servicio de Windows”.

También puede instalar múltiples servidores MySQL como servicios. En este caso, debe asegurarse que cada servidor usa un nombre de servicio distinto además de todos los otros parámetros que deben ser únicos para cada servidor.

Para las siguientes instrucciones, se asume que desea ejectuar el servidor `mysqld-nt` a partir de dos versiones diferentes de MySQL que estén instaladas en `C:\mysql-4.1.8` y `C:\mysql-5.0.7`, respectivamente. (Este podría ser el caso si está ejecutando 4.1.8 como servidor de producción, pero quiere realizar tests usando 5.0.7.)

Los siguientes principios se aplican al instalar un servicio MySQL con las opciones `--install` o `--install-manual`:

- Si no especifica un nombre para el servicio, el servidor usa el nombre de servicio por defecto de `MySQL` y el servidor lee las opciones del grupo `[mysqld]` en el fichero de opciones.
- Si especifica un nombre de servicio tras la opción `--install` , el servidor ignora el grupo de opciones `[mysqld]` y lee las opciones del grupo que tenga el mismo nombre que el servicio. El servidor lee opciones del fichero de opciones.
- Si especifica la opción `--defaults-file` tras el nombre del servicio, el servidor ignora el fichero estándar de opciones y lee opciones sólo del grupo `[mysqld]` del fichero especificado.

Nota: Antes de MySQL 4.0.17, sólo un servidor instalado usando el nombre de servicio por defecto (`MySQL`) o instalado explícitamente con el nombre de servicio de `mysqld` lee el grupo `[mysqld]` en el fichero de opciones . Como en 4.0.17, todos los servidores leen el grupo `[mysqld]` si leen el fichero de opciones , incluso si están instalados usando otro nombre de servicio. Esto permite usar el grupo `[mysqld]` para opciones que deben usarse por todos los servicios MySQL, y un grupo de opciones nombrado tras cada servicio para uso del servidor instalado con ese nombre de servicio.

Basándonos en la información precedente, hay varias formas de inicializar múltiples servicios. Las siguientes instrucciones describen algunos ejemplos. Antes de probar cualquiera de ellos, asegúrese que apaga y elimina cualquier servicio MySQL antes de nada.

- **Aproximación 1:** Especifique las opciones para todos los servicios en uno de los ficheros de opciones . Para ello, use un nombre de servicio distinto para cada servidor. Suponga que quiere ejecutar `mysqld-nt` 4.1.8 usando el nombre de servicio de `mysqld1` y `mysqld-nt` 5.0.7 usando el nombre de servicio `mysqld2`. En ese caso, puede usar el grupo `[mysqld1]` para 4.1.8 y el grupo `[mysqld2]` para 5.0.7. Por ejemplo, puede rellenar `C:\my.cnf` así:

  ```
  # options for mysqld1 service
  [mysqld1]
  basedir = C:/mysql-4.1.8
  port = 3307
  enable-named-pipe
  socket = mypipe1

  # options for mysqld2 service
  [mysqld2]
  basedir = C:/mysql-5.0.7
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Instale los servicios como se describe a continuación, usando las rutas completas de los servidores para asegurarse que Windows registra el programa ejecutable correcto para cada servicio:

  ```
  C:\> C:\mysql-4.1.8\bin\mysqld-nt --install mysqld1
  C:\> C:\mysql-5.0.7\bin\mysqld-nt --install mysqld2
  ```

  Para arrancar los servicios, use el administrador de servicios, o use `NET START` con los nombres de servicio apropiados:

  ```
  C:\> NET START mysqld1
  C:\> NET START mysqld2
  ```

  Para parar los servicios, use el administrador de servicios, o use `NET STOP` con el nombre de servicio apropiado:

  ```
  C:\> NET STOP mysqld1
  C:\> NET STOP mysqld2
  ```

- **Aproximación 2:** Especifique opciones para cada servidor en ficheros separados y use `--defaults-file` cuando instale los servicios para decirle a cada servidor que fichero usar. En ese caso, cada fichero debe listar las opciones en el grupo `[mysqld]`.

  Con esta aproximación, para especificar opciones para `mysqld-nt` 4.1.8, cree un fichero `C:\my-opts1.cnf` como se muestra a continuación:

  ```
  ```

```
[mysqld]
basedir = C:/mysql-4.1.8
port = 3307
enable-named-pipe
socket = mypipe1
```

Para `mysqld-nt` 5.0.7 , cree un fichero `C:\my-opts2.cnf` como se muestra a continuación:

```
[mysqld]
basedir = C:/mysql-5.0.7
port = 3308
enable-named-pipe
socket = mypipe2
```

Instale los servicios como se muestra (introduzca cada comando en una línea distinta):

```
C:\> C:\mysql-4.1.8\bin\mysqld-nt --install mysqld1
          --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-5.0.7\bin\mysqld-nt --install mysqld2
          --defaults-file=C:\my-opts2.cnf
```

Para usar la opción `--defaults-file` cuando instale un servidor MySQL como servicio, debe preceder la opción con el nombre de servicio.

Tras instalar los servicios, arranque y párelos igual que en el ejemplo precedente.

Para eliminar múltiples servicios, use `mysqld --remove` para cada uno, especificando un nombre de servicio con la opción `--remove` a continuación. Si el nombre de servicio es el nombre por defecto (`MySQL`), puede omitirlo.

5.11.2. Ejecutar varios servidores en Unix

La forma más sencilla de ejecutar múltiples servidores en Unix es compilarlos con diferentes puertos TCP/IP y ficheros socket Unix de forma que cada uno esté escuchando en distintas interfaces de red. Además, al compilarlos en distintos directorios base para cada instalación, automáticamente se configuran distintos directorios de datos, fichero de log, y localización del fichero PID para cada uno de los servidores.

Tenga en cuenta que un servidor 4.1.8 existente está configurado para el puerto TCP/IP por defecto (3306) y fichero socket Unix (`/tmp/mysql.sock`). Para configurar un nuevo servidor 5.0.7 con parámetros distintos, use un comando `configure` parecido a este:

```
shell> ./configure --with-tcp-port=port_number \
            --with-unix-socket-path=file_name \
            --prefix=/usr/local/mysql-5.0.7
```

Aquí, *`port_number`* y *`file_name`* deben ser distintos del puerto TPC/IP por defecto y de la ruta del fichero socket Unix, y el valor `--prefix` debe especificar un fichero de instalación distinto del que hay donde MySQL se ha instalado por defecto.

Si tiene un servidor MySQL escuchando en un puerto dado, puede usar el siguiente comando para encontrar que parámetros está usando para algunas importantes variables de configuración, incluyendo el directorio base y el nombre del fichero socket Unix:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

Con la información mostrada por este comando, puede decir los valores de las opciones que *no* debe usar para configurar un servidor adicional.

Tenga en cuenta que si especifica `localhost` como nombre de equipo, el comportamiento por defecto de `mysqladmin` es usar una conexión mediante un fichero socket Unix en lugar de TCP/IP. A partir de MySQL 4.1 , puede especificar explícitamente el protocolo de conexión a usar usando la opción `--protocol={TCP | SOCKET | PIPE | MEMORY}`.

No tiene que compilar un nuevo servidor MySQL para arrancar con un fichero socket Unix distinto o con otro puerto TCP/IP. También es posible especificar estos valores en tiempo de ejecución. Una forma de hacerlo es usando opciones por líneas de comandos:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

Para arrancar un segundo servidor, proporcione distintas opciones para `--socket` y `--port` option , e incluya la opción `--datadir=path` para `mysqld_safe` de forma que el servidor use un directorio de datos distinto.

Otra forma de conseguir un efecto similar es usar variables de entorno para inicializar el nombre de fichero socket Unix y el puerto TCP/IP:

```
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 &
```

Hay una forma rápida de arrancar un segundo servidor para hacer pruebas. Lo bueno de este método es que los cambios en las variables de entorno se aplican a cualquier programa cliente que invoque desde la misma consola. Así, las conexiones de dichos clientes se redireccionan automáticamente al segundo servidor!

Apéndice E, *Variables de entorno* incluye una lista de otras variables de entorno que puede usar para afectar a `mysqld`.

Para ejecución automática del servidor, el script de arranque que se ejecuta al arrancar debe ejecutar el siguiente comando una vez para cada servidor con la ruta apropiada del fichero de opciones para cada comando:

```
mysqld_safe --defaults-file=path
```

Cada fichero de opciones debe contener valores específicos para un servidor dado.

En Unix, el script `mysqld_multi` es otra forma de arrancar múltiples servidores. Consulte Sección 5.1.5, “El programa `mysqld_multi` para gestionar múltiples servidores MySQL”.

5.11.3. Utilización de programas cliente en un entorno de múltiples servidores

Cuando quiera conectar con un programa cliente a un servidor MySQL que está escuchando en una interfaz de red distinta de la compilada en su cliente, puede usar los siguientes métodos:

- Arranque el cliente con `--host=host_name --port=port_number` para conectar via TCP/IP a un servidor remoto, con `--host=127.0.0.1 --port=port_number` para conectar via TCP/IP a un servidor local, o con `--host=localhost --socket=file_name` para conectar a un servidor local via fichero socket Unix o una named pipe de Windows.
- Como en MySQL 4.1, arranque el cliente con `--protocol=tcp` para conectar via TCP/IP, `--protocol=socket` para conectar via fichero socket Unix , `--protocol=pipe` para conectar via named pipe, o `--protocol=memory` para conectar via memoria compartida. Para conexiones TCP/

IP, puede necesitar especificar las opciones `--host` y `--port`. Para los otros tipos de conexión, puede necesitar especificar la opción `--socket` para especificar un fichero socket Unix o nombre de named pipe, o la opción `--shared-memory-base-name` para especificar el nombre de la memoria compartida. Las conexiones de memoria compartida se soportan sólo en Windows.

- En Unix, inicialice las variables de entorno `MYSQL_UNIX_PORT` y `MYSQL_TCP_PORT` para que apunten al fichero socket Unix y al puerto TCP/IP antes de arrancar los clientes. Si normalmente usa un fichero socket o puerto específico, puede preparar comandos para inicializar estas variables de entorno en su fichero `.login` para que se apliquen cada vez que entre. Consulte Apéndice E, *Variables de entorno*.
- Especifique el fichero socket Unix y el puerto TCP/IP por defecto en el grupo `[client]` de un fichero de opciones. Por ejemplo, puede usar `C:\my.cnf` en Windows, o el fichero `.my.cnf` en su directorio "home" en Unix. Consulte Sección 4.3.2, “Usar ficheros de opciones”.
- En un programa C, puede especificar los argumentos para el fichero socket o puerto en la llamada `mysql_real_connect()`. PUede tener las opciones leídas por el programa llamando a `mysql_options()`. Consulte Sección 24.2.3, “Descripción de funciones de la API C”.
- Si usa el módulo de Perl `DBD::mysql` , puede leer las opciones de los ficheros de opciones MySQL. Por ejemplo:

```
$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);
```

Consulte Sección 24.4, “La API Perl de MySQL”.

Otras interfaces de programación pueden proporcionar funcionalidad similar para leer ficheros de opciones.

5.12. La caché de consultas de MySQL

MySQL 5.0 Server proporciona una query cache. Cuando se usa, la query cache almacena el texto de una consulta `SELECT` junto con el resultado que se le envió al cliente. Si se recibe una consulta idéntica posteriormente, el servidor devuelve el resultado de la caché de consultas en lugar de parsear y ejecutar la consulta de nuevo.

La caché de consultas es muy útil en un entorno donde tiene tablas que no cambian frecuentemente y donde el servidor recibe muchas consultas idénticas. Esta es la típica situación de muchos servidores Web que generan muchas páginas dinámicas basadas en contenido de base de datos.

Nota: La caché de consultas no devuelve datos antiguos. Cuando las tablas se modifican, cualquier entrada relevante en la caché de consultas se eliminia.

Nota: La caché de consultas no funciona en un entorno donde tiene muchos servidores `mysqld` actualizando las mismas tablas `MyISAM`.

Nota: La caché de consultas no se usa para comandos preparados en la parte del servidor. Si está usando este tipo de comandos preparados, considere que no se beneficiarán de la caché de consultas. Consulte Sección 24.2.4, “Sentencias preparadas de la API C”.

A continuación algunos datos de rendimiento de la caché de consultas. Estos resultados se generaron con el MySQL benchmark suite en un Linux Alpha 2 x 500MHz con 2GB RAM y 64MB de caché de consultas.

- Si todas las consultas que está ejecutando son simples (tales como seleccionar un registro de una tabla con un registro), pero difierente de forma que las consultas no pueden cachearse, la pérdida de rendimiento por tener activa la caché de consultas es del 13%. Este puede considerarse el peor

escenario posible. En el mundo real, las consultas suelen ser mucho más complicadas, así que la pérdida de rendimiento es considerablemente menor.

- Las búsquedas para un registro en una tabla de un registro són un 238% más rápidas con la caché de consultas que sin ella. Esto puede considerarse como la mínima mejora esperada para una consulta que se cachea.

Para desactivar la caché de consultas al arrancar el servidor, ponga la variable de sistema `query_cache_size` a 0. Al desactivar el código de caché de consultas, no hay una pérdida de rendimiento evidente. Las capacidades de la caché de consultas pueden quitarse totalmente del servidor usando la opción `--without-query-cache` con `configure` al compilar MySQL.

5.12.1. Cómo opera la caché de consultas

Esta sección describe cómo funciona la caché de consultas cuando está operacional. Sección 5.12.3, “Configuración de la caché de consultas” describe cómo controlar si está o no operacional.

Las consultas se comparan antes de parsearla, así que las siguientes dos consultas se consideran diferentes por la caché de consultas:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Las consultas deben ser *exactamenten* las mismas (byte a byte) para se consideradas idénticas. Además, las consultas que son idénticas pueden tratarse como diferentes por otras razones. Las consultas que usan distintas bases de datos, distintas versiones de protocolo, o distintos conjuntos de caracteres se consideran distintas consultas y se cachean por separado.

Antes que una consulta se guarde en la cache de consultas, MySQL comprueba que el usuario tenga permisos de `SELECT` para todas las bases de datos y tablas involucradas. Si no es el caso, el resultado cacheado no se usa.

Si un resultado de consulta se retorna desde la caché de consultas, el servidor incrementa la variable de estado `Qcache_hits`, no `Com_select`. Consulte Sección 5.12.4, “Estado y mantenimiento de la caché de consultas”.

Si una tabla cambia, entonces todas las consultas cacheadas que usen esa tabla pasan a ser inválidas y se eliminan de la caché.Esto incluye consultas que usen tablas `MERGE` que mapeen las tablas cambiadas. Una tabla puede cambiarse por varios tipos de comandos, tales como `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `ALTER TABLE`, `DROP TABLE`, o `DROP DATABASE`.

Las tablas transaccionales `InnoDB` que se han cambiado se invalidan cuando se realiza un `COMMIT`.

En MySQL 5.0, la caché de consultas también funciona dentro de transacciones cuando se usa tablas `InnoDB` ,haciendo uso del número de versión para detectar si sus contenidos han cambiado.

En MySQL 5.0, las consultas generadas por vistas se cachean. En MySQL 5.0.3 se resolvió un problema con los resultados en la caché de consultas generados por vistas que no se invalidaban apropiadamente tras algunas operaciones en dichas vistas. Consulte Sección C.1.9, “Cambios en la entrega 5.0.3 (23 Mar 2005: Beta)”.

Antes de MySQL 5.0, una consulta que empezase con un comentario podía cachearse, pero no podía obtenerse de la caché. Este problema está resuelto en MySQL 5.0.

La caché de consultas funciona para consultas del tipo `SELECT SQL_CALC_FOUND_ROWS ...` y `SELECT FOUND_ROWS()`. `FOUND_ROWS()` devuelve el valor correcto incluso si la consulta precedente se obtuvo de la cache debido a que el número de registros encontrados también se almacena en la caché.

Una consulta no puede cachearse si contiene cualquiera de las siguientes funciones:

BENCHMARK()	CONNECTION_ID()	CURDATE()
CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
CURTIME()	DATABASE()	ENCRYPT() con un parámetro
FOUND_ROWS()	GET_LOCK()	LAST_INSERT_ID()
LOAD_FILE()	MASTER_POS_WAIT()	NOW()
RAND()	RELEASE_LOCK()	SYSDATE()
UNIX_TIMESTAMP() sin parámetros	USER()	

Una consulta tampoco se cachea bajo las siguientes condiciones:

- Se refiere a funciones definidas por el usuario (UDFs).
- Se refiere a variables de usuario.
- Se refiere a las tablas en la base de datos del sistema mysql .
- Es cualquiera de las siguientes formas:

```
SELECT ... IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

  La última forma no se cachea ya que se usa como solución de ODBC para obtener el último ID insertado. Consulte “Obtaining Auto-Increment Values”.

- Se usó como comando preparado, incluso sin emplear marcadores. Por ejemplo, la consulta usada aquí:

```
char *my_sql_stmt = "SELECT a, b FROM table_c";
/* ... */
mysql_stmt_prepare(stmt, my_sql_stmt, strlen(my_sql_stmt));
```

  no se cachea. Consulte Sección 24.2.4, “Sentencias preparadas de la API C”.

- Usa tablas TEMPORARY.
- No usa ninguna tabla.
- El usuario tiene permisos a nivel de columan para cualquiera de las tablas involucradas.

5.12.2. Opciones de SELECT para la caché de consultas

Dos opciones relacionadas con la caché de consultas pueden especificarse en los comandos SELECT:

- SQL_CACHE

  El resultado de la consulta se cachea si el valor de la variable de sistema query_cache_type es ON o DEMAND.

- SQL_NO_CACHE

La consulta resultado no se cachea.

Ejemplos:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

5.12.3. Configuración de la caché de consultas

La variable de sistema `have_query_cache` indica si la caché de consultas está disponible:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

Cuando se usa un binario MySQL 5.0 estándar, este valor siempre es `YES`, incluso si el cacheo de consultas está desactivado.

Muchas otras variables de sistema controlan las operaciones de la caché de consultas. Pueden especificarse en un fichero de opciones o en la línea de comandos al arrancar `mysqld`. Todas las variables de sistema de la caché de consultas tienen nombres que empiezan con `query_cache_`. Se describen brevemente en Sección 5.3.3, “Variables de sistema del servidor”, con información adicional de configuracion dada aquí.

Para especificar el tamaño de la caché de consulta, inicialice la variable de sistema `query_cache_size`. Ponerla a 0 desactiva la caché de consultas. El tamaño por defecto es 0; esto es, que está desactivada.

Si el tamaño de la caché de consultas es mayor que 0, la variable `query_cache_type` influye en su funcionamiento. Esta variable puede tener los siguientes valores:

- Un valor de `0` o `OFF` evita cachear o recibir valores cacheados.
- Un valor de `1` o `ON` permite el cacheo excepto para aquellos comandos que empiecen con `SELECT SQL_NO_CACHE`.
- Un valor de `2` o `DEMAND` provoca el cacheo de sólo los comandos que empiecen con `SELECT SQL_CACHE`.

Inicializar con el valor `GLOBAL` la variable `query_cache_type` determina el comportamiento de la caché de consultas para todos los clientes que se conecten tras el cambio. Cada clientes puede controlar el comportamiento de la caché para su propia conexión meidante el valor `SESSION` de `query_cache_type`. Por ejemplo, un cliente puede desactivar el uso de la caché de consultas para sus propias consultas así:

```
mysql> SET SESSION query_cache_type = OFF;
```

Para controlar el tamaño máximo de resultados de consultas individuales que pueden cachearse, inicialice la variable `query_cache_limit` variable. El valor por defecto es 1MB.

Cuando se cachea una consulta, su resultado (los datos que se envían al cliente) se guardan en la caché a medida que se reciben. Por lo tanto los datos normalmente no se guardan en un gran paquete. La caché reserva bloques para guardar estos datos bajo demanda, así que cuando se llena

un bloque, se prepara un nuevo bloque. Puesto que la reserva de memoria es una operación costosa (lenta), la caché de consultas reserva bloques con un tamaño mínimo dado por la variable de sistema `query_cache_min_res_unit`. Cuando la consulta queda ejecutada, el último bloque de resultados se ajusta a su tamaño real para liberar la memoria no usada. En función del tipo de consulta que ejecute el servidor, puede encontrar útil cambiar el valor de `query_cache_min_res_unit`:

- El valor por defecto de `query_cache_min_res_unit` es 4KB. Debería ser un valor adecuado para la mayoría de los casos.
- Si tiene muchas consultas con resultados pequeños, el tamaño por defecto del bloque puede llevar a fragmentación de memoria, que se evidencia con un alto número de bloques libres. La fragmentación puede obligar a la caché a borrar consultas por falta de memoria. En este caso, debe decrementar el valor de `query_cache_min_res_unit`. El número de bloques libres y de consultas borradas debido a falta de espacio se da en las variables `Qcache_free_blocks` y `Qcache_lowmem_prunes`.
- Si la mayoría de las consultas tienen resultados grandes (compruebe las variables de estado `Qcache_total_blocks` y `Qcache_queries_in_cache`), puede incrementar el rendimiento incrementando `query_cache_min_res_unit`. Sin embargo, tenga cuidado de no hacerlo demasiado grande (consulte el punto anterior).

5.12.4. Estado y mantenimiento de la caché de consultas

Puede comprobar si la caché de consultas está presente en su servidor MySQL con el siguiente comando:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

Puede defragmentar la caché de consultas para mejor uso de esta memoria con el comando `FLUSH QUERY CACHE` . El comando no elimina ninguna consulta de la caché.

El comando `RESET QUERY CACHE` elimina todos los resultados de consultas de la caché de consultas. El comando `FLUSH TABLES` también lo hace.

Para monitorizar el rendimiento de la caché de consultas, use `SHOW STATUS` para ver las variables de estado de la caché:

```
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    |
+-------------------------+--------+
```

Las descripicones de estas variables se dan en Sección 5.3.4, “Variables de estado del servidor”. Algunos de sus usos se describen aquí.

El número total de consultas `SELECT` es igual a:

```
  Com_select
+ Qcache_hits
+ consultas con errores detectados por el parser
```

El valor `Com_select` es igual a:

```
  Qcache_inserts
+ Qcache_not_cached
+ consultas con errores encontrados durante la comprobación de columnas/permisos
```

La caché de consultas usa bloques de tamaño variable, así que `Qcache_total_blocks` y `Qcache_free_blocks` pueden indicar fragementación de memoria de la caché. Tras `FLUSH QUERY CACHE`, sólo queda un bloque de memoria libre.

Cada consulta cacheada requiere como mínimo dos bloques (uno para el texto de la consulta y otro o más para el resultado de la misma). Además, cada tabla utilizada por una consulta requiere un bloque. Sin embargo, si dos o más consultas usan la misma tabla, sólo se reserva un bloque.

La información proporcionada por la variable de estado `Qcache_lowmem_prunes` puede ayudarle a ajustar el tamaño de la caché de consultas. Cuenta el número de consultas que se han eliminado de la caché para liberar memoria con el fin de cachear nuevas consultas. La caché de consultas usa una estrategia de resultado usado menos recientemente (LRU, siglas en inglés de least recently used) para decidir qué consultas eliminar de la caché. Más información para afinar las variables en Sección 5.12.3, “Configuración de la caché de consultas”.

Capítulo 6. Replicación en MySQL

Tabla de contenidos

Las capacidades de replicación que permiten a las bases de datos de un servidor MySQL ser duplicadas en otro se introdujeron en MySQL 3.23.15. Este capítulo describe las características de replicación proporcionadas por MySQL . Presenta conceptos de replicación, muestra cómo preparar servidores de replicación, y sirve como referencia para las opciones de replicación disponibles. También proporciona una lista de preguntas frecuentes (con respuestas), y consejos para resolver problemas.

Para una descripción de la sintaxis de comandos SQL relacionados con replicación, consulte Sección 13.6, “Sentencias de replicación”.

Sugerimos que visite nuestra página web http://www.mysql.com frecuentemente así como que chequee para revisiones de este capítulo. La replicación está siendo mejorada constantemente, y actualizamos frecuentemente el manual con la información más actual.

6.1. Introducción a la replicación

Las características de MySQL 5 soportan replicación asíncrona unidireccional: un servidor actúa como maestro y uno o más actúan como esclavos. (Esto contrasta con la replicación *síncrona* que es una característica de MySQL Cluster — consulte Capítulo 16, *MySQL Cluster*). El servidor maestro escribe actualizaciones en el fichero de log binario, y mantiene un índice de los ficheros para rastrear las rotaciones de logs. Estos logs sirven como registros de actualizaciones para enviar a los servidores esclavos. Cuando un esclavo se conecta al maestro, informa al maestro de la posición hasta la que el esclavo ha leído los logs en la última actualización satisfactoria. El esclavo recibe cualquier actualización que han tenido lugar desde entonces, y se bloquea y espera para que el master le envíe nuevas actualizaciones.

Un esclavo servidor puede servir como maestro si quiere preparar una cadena de replicaciones de replicación.

Tenga en cuenta que cuando usa replicación, todas las actualizaciones de las tablas que se replican deben realizarse en el servidor maestro. De otro modo, debe ser cuidadoso para evitar conflictos entre

actualizaciones que hacen los usuarios a las tablas en el maestro y las actualizaciones que hacen en las tablas de los esclavos.

La replicación unidireccional tiene beneficios para la robustez, velocidad, y administración del sistema:

- La robustez se incrementa con un escenario maestro/esclavo. En caso de problemas con el maestro, puede cambiar al esclavo como copia de seguridad.
- Puede conseguirse un mejor tiempo de respuesta dividiendo la carga de consultas de clientes a procesar entre los servidores maestro y esclavo. Se puede enviar consultas `SELECT` al esclavo para reducir la carga de proceso de conultas del maestro. Sin embargo, las sentencias que modifican datos deben enviarse siempre al maestro, de forma que el maestro y el esclavo no se desincronicen. Esta estrategia de balanceo de carga es efectiva si dominan consultas que no actualizan datos, pero este es el caso más habitual.
- Otro beneficio de usar replicación es que puede realizar copias de seguridad usando un servidor esclavo sin molestar al maestro. El maestro continúa procesando actualizaciones mientras se realiza la copia de seguridad. Consulte Sección 5.8.1, “Copias de seguridad de bases de datos”.

6.2. Panorámica de la implementación de la replicación

La replicación en MySQL se basa en un servidor maestro que toma nota de todos los cambios en las bases de datos (actualizaciones, borrados, y así) en los logs binarios. Por lo tanto, para usar replicación, debe activar el log binario en el servidor maestro. Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

Cada servidor esclavo recibe del maestro las actualizaciones guardadas que el maestro ha guardado en su log binario, de forma que el esclavo puede ejecutar las mismas actualizaciones en su copia de los datos.

Es *extremadamente* importante tener en cuenta que el log binario simplemente es un registro que comienza en un punto fijo en el tiempo en el que activa el log binario. Cualquier esclavo que inicialice necesita copias de las bases de datos del maestro *tal y como estaban en el momento en que activó el log binario en el maestro*. Si arranca sus esclavos con bases de datos que no están en el mismo estado que las del maestro cuando arrancó el log binario, es muy posible que fallen sus esclavos.

Una forma de copiar los datos del maestro al esclavo es usar el comando `LOAD DATA FROM MASTER` . Tenga en cuenta que `LOAD DATA FROM MASTER` funciona sólo si todas las tablas en el maestro usan el motor de almacenamiento `MyISAM`. Además, este comando adquiere un bloqueo de lectura global, así que no se puede actualizar el maestro mientras las tablas se transfieren al esclavo. Cuando implementemos la copia en caliente sin bloqueo, ya no será necesario el bloqueo global de lectura.

Debido a estas limitaciones, recomendamos que en este punto use `LOAD DATA FROM MASTER` sólo si el conjunto de datos en el maestro es relativamente pequeño, o si se puede realizar un bloqueo de lectura prolongado en el maestro. Mientras que la velocidad real de `LOAD DATA FROM MASTER` puede variar de sistema a sistema, una buena regla de estimar cuánto tarda es 1 segundo por 1MB de datos. Esta es una estimación aproximada, pero puede encontrar que es bastante buena si tanto el maestro como el esclavo son equivalentes a Pentium 700MHz y conectados mediante una red a 100Mbps .

Tras inicializar el esclavo con una copia de los datos del maestro, se conecta al maestro y espera a procesar actualizaciones. Si el maestro falla, o el esclavo pierde conectividad con el maestro, el esclavo sigue intentando conectar periódicamente hasta que es capaz de reanudar la espera de actualizaciones. El intervalo de reintento lo controla la opción `--master-connect-retry`. Por defecto es de 60 segundos.

Cada esclavo registra dónde lo dejó. El servidor maestro tiene conocimiento de cuántos esclavos hay o cuántos están activos en un momento dado.

6.3. Detalles de la implementación de la replicación

Las capacidades de replicación de MySQL están implementadas usando tres flujos (uno en el servidor maestro y dos en el esclavo). Cuando se ejecuta un `START SLAVE`, el esclavo crea un flujo de entrada/salida, que conecta al maestro y le pide que envíe los comandos guardados en su log binario. El maestro crea un flujo para enviar los contenidos del log binario al esclavo. Este flujo puede identificarse como el flujo `Binlog Dump` en la salida de `SHOW PROCESSLIST` en el maestro. El flujo de entrada/salida del esclavo lee lo que el flujo `Binlog Dump` del maestro envía y copia estos datos en ficheros locales, llamados *logs retardados*, en el directorio de datos del esclavo. El tercer flujo es el flujo SQL, que crea el esclavo para leer los logs retardados y para ejecutar las actualizaciones que contiene.

En la descripción precedente, hay tres flujos por esclavo. Un maestro que tiene varios esclavos crea un flujo para cada esclavo conectado; cada esclavo tiene sus propios flujos de entrada/salida y SQL.

Leer dos comandos y ejecutarlos se divide en dos tareas independientes. La tarea de leer comandos no se ralentiza si la ejecución es lenta. Por ejemplo, si el servidor esclavo no ha estado en ejecución durante un periodo de tiempo, su flujo de entrada/salida puede tratar rápidamente todos los contenidos del log binario del maestro cuando arranca el esclavo, incluso si el flujo SQL se realentiza mucho. Si el esclavo para antes que el flujo SQL haya ejecutado todos los comandos tratados, el flujo de entrada/salida ha tratado todo de forma que hay una copia de los comandos almacenada localmente en los logs retardados, preparados para ejecutarse la siguiente vez que arranque el esclavo. Esto permite que se purguen los logs binarios del maestro, ya que no necesita esperar que los esclavos traten sus contenidos.

El comando `SHOW PROCESSLIST` proporciona información que le dice qué está ocurriendo en el maestro y en el esclavo teniendo en cuenta la replicación.

El siguiente ejemplo ilustra cómo los tres flujos se muestran en `SHOW PROCESSLIST`.

En el servidor maestro, la salida de `SHOW PROCESSLIST` tiene este aspecto:

```
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
```

Aquí, el flujo 2 es un flujo de replicación para un esclavo conectado. La información indica que todas las actualizaciones se han enviado al esclavo y que el maestro está esperando más actualizaciones.

En el servidor esclavo, la salida para `SHOW PROCESSLIST` tiene este aspecto:

```
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
```

Esta información indica que el flujo 10 es el flujo de entrada/salida que se comunica con el maestro, y el flujo 11 es el flujo SQL que procesa las actualizaciones almacenadas en los logs retardados. Cuando se ejecuta `SHOW PROCESSLIST` ambos flujos estaban en espera de actualizaciones nuevas.

Tenga en cuenta que los valores en la columna `Time` pueden mostrar la diferencia de tiempo en la comparación del esclavo con el maestro. Consulte Sección 6.9, “Preguntas y respuestas sobre replicación”.

6.3.1. Estados de los subprocesos del maestro de replicación

La siguiente lista muestra los estados más comunes que puede ver en la columna `State` del flujo maestro `Binlog Dump` . Si no ve ningún flujo `Binlog Dump` en un servidor maestro, esto significa que la replicación no está corriendo — esto es, que no hay ningún esclavo conectado.

- `Envío de eventos del binlog al esclavo`

  Los logs binarios consisten en *eventos*, donde un evento usualmente es una actualicación más otra información. El flujo lee un evento del log binario y lo envía al esclavo.

- `Finished reading one binlog; switching to next binlog`

  El flujo ha acabado de leer un fichero de log binario y está abriendo el siguiente para enviar al esclavo.

- `Has sent all binlog to slave; waiting for binlog to be updated`

  El flujo ha leído todas las actualizaciones destacadas del log binario y las ha enviado al esclavo. El flujo ahora está en espera, esperando nuevos eventos en el log binario que resulten en nuevas actualizaciones en el maestro.

- `Waiting to finalize termination`

  Estado muy breve que ocurre cuando el flujo está parando.

6.3.2. Estados de proceso E/S (I/O) del esclavo de replicación

La siguiente lista muestra los estados más comunes que puede ver en la columna `State` para un flujo de entrada/salida en el servidor esclavo. Este estado también aparece en la columna `Slave_IO_State` mostrada por `SHOW SLAVE STATUS`. Esto significa que puede obtener una vista muy buena de lo que ocurre simplemente usando este comando.

- `Connecting to master`

  El flujo trata de conectar al maestro.

- `Checking master version`

  Estado que ocurre brevemente, inmediatamente tras establecer la conexión con el maestro.

- `Registering slave on master`

Estado que ocurre brevemente, inmediatamente tras establecer la conexión con el maestro.

- `Requesting binlog dump`

  Estado que ocurre brevemente, inmediatamente tras establecer la conexión con el maestro. El flujo envía al maestro una petición para los contenidos de su log binario, comenzando por el nombre del log binario pedido y la posición.

- `Waiting to reconnect after a failed binlog dump request`

  Si la petición del volcado del log binario falla (debido a una desconexión), el flujo pasa a este estado mientras duerme, luego trata de reconectar periódicamente. El intervalo entre reintentos puede especificarse usando la opción `--master-connect-retry` .

- `Reconnecting after a failed binlog dump request`

  El flujo está tratando de reconectar con el maestro.

- `Waiting for master to send event`

  El flujo ha conectado con el maestro y está esperando a que lleguen los eventos del log binario. Esto puede tardar un tiempo largo si el maestro está en espera. Si la espera dura `slave_read_timeout` segundos, se produce un timeout. En este punto, el flujo considera la conexión rota e intenta reconectar.

- `Queueing master event to the relay log`

  El flujo ha leído un evento y lo está copiando en el log retardado para que el flujo SQL pueda procesarlo.

- `Waiting to reconnect after a failed master event read`

  Un error ocurre durante la lectura (debido a una desconexión). El flujo duerme durante `master-connect-retry` segundos antes de intentar reconectar.

- `Reconnecting after a failed master event read`

  El flujo trata de reconectar con el maestro. Cuando la conexión se vuelve a establecer, el estado pasa a `Waiting for master to send event`.

- `Waiting for the slave SQL thread to free enough relay log space`

  Está usando un valor `relay_log_space_limit` distinto a cero, y el log retardado ha crecido hasta que su tamaño combinado escede este valor. El flujo de entrada/salida está en espera hasta que el flujo SQL libera suficiente espacio procesando los contenidos del log retardado de forma que pueda leer algunos ficheros de log retardado.

- `Waiting for slave mutex on exit`

  Estado que ocurre brevemente al parar el flujo.

6.3.3. Estados del flujo SQL de un esclavo de replicación

La siguiente lista muestra los estados más comunes que puede ver en la columna `State` para el flujo SQL del servidor esclavo:

- `Reading event from the relay log`

  El flujo ha leído un evente del log retardado para que el evento pueda ser procesado.

- `Has read all relay log; waiting for the slave I/O thread to update it`

  El flujo ha procesado todos los eventos en los ficheros de log retardado y está esperando que el flujo de entrada/salida escriba nuevos eventos en el log retardado.

- `Waiting for slave mutex on exit`

  Estado breve que ocurre cuando el flujo está parándose.

La columna `State` para el flujo de entrada/salida también puede mostrar el texto de un comando. Esto indica que el flujo ha leído un evento del log retardado, ha extraído el comando de él y lo ha ejecutado.

6.3.4. Ficheros de replicación, retardados y de estado

Por defecto, los logs retardados se nombran usando nombres de ficheros de la forma *`host_name`*`-relay-bin.`*`nnnnnn`*, donde *`host_name`* es el nombre del servidor esclavo y *`nnnnnn`* es un número de secuencia. Los ficheros de logs retardados sucesivos en MySQL 5.0 se crean usando números de secuencia sucesivos, comenzando a partir de `000001`. El esclavo guarda los logs retardados en unso en un fichero índice. El nombre del índice del log retardado por defecto es *`host_name`*`-relay-bin.index`. Por defecto, estos ficheros se crean en el directorio de datos del esclavo. Por defecto los nombres de fichero pueden cambiarse con las opciones de servidor `--relay-log` y `--relay-log-index` . Consulte Sección 6.8, “Opciones de arranque de replicación”.

Los logs retardados tienen el mismo formato que los logs binarios, y pueden leerse usando `mysqlbinlog`. Un log retardado se borra automáticamente por el flujo SQL en cuanto ha ejecutado todos sus eventos y no los necesita más. No hay mecanismo explícito para borrar logs retardados, ya que el flujo SQL se encarga de ello. Sin embargo, `FLUSH LOGS` rota los logs retardados, lo que influye cuando el flujo SQL los borra.

Un log retardado se crea bajo las siguientes condiciones:

- En MySQL 5.0, se crea un nuevo log retardado cada vez que arranca el flujo de entrada/salida.
- Cuando los logs se vuelcan; por ejemplo, con `FLUSH LOGS` o `mysqladmin flush-logs`.
- Cuando el tamaño del log retardado actual es demasiado grande. El significado de "muy grande" se determina así:
  - `max_relay_log_size`, si `max_relay_log_size` > 0
  - `max_binlog_size`, si `max_relay_log_size` = 0

Un servidor esclavo de replicación crea dos pequeños ficheros adicionales en el directorio de datos. Estos *ficheros de estado* se llaman `master.info` y `relay-log.info` por defecto. Contienen información como la mostrada en la salida del comando `SHOW SLAVE STATUS` (consulte Sección 13.6.2, “Sentencias SQL para el control de servidores esclavos” para una descripción de este comando). Como ficheros en disco, sobreviven una parada del servidor esclavo. La siguiente vez que arranca el servidor esclavo, lee estos ficheros para determinar hasta dónde ha procesado los logs binarios del maestro y sus propios logs retardados.

El fichero `master.info` lo actualiza el flujo de entrada/salida. En MySQL 5.0, la correspondencia entre las líneas en el fichero y las columnas mostradas por`SHOW SLAVE STATUS` es la siguiente:

Línea	Descripción
1	Número de líneas en el fichero
2	`Master_Log_File`

3	`Read_Master_Log_Pos`
4	`Master_Host`
5	`Master_User`
6	Contraseña (no mostrada por `SHOW SLAVE STATUS`)
7	`Master_Port`
8	`Connect_Retry`
9	`Master_SSL_Allowed`
10	`Master_SSL_CA_File`
11	`Master_SSL_CA_Path`
12	`Master_SSL_Cert`
13	`Master_SSL_Cipher`
14	`Master_SSL_Key`

El fichero `relay-log.info` lo actualiza el flujo SQL. La correspondencia entre las líneas en el fichero y las columnas mostradas por `SHOW SLAVE STATUS` se muestran aquí:

Línea	Descripción
1	`Relay_Log_File`
2	`Relay_Log_Pos`
3	`Relay_Master_Log_File`
4	`Exec_Master_Log_Pos`

Cuando hace una copia de seguridad de los datos del esclavo, debe incluir también estos dos ficheros, junto con sus ficheros de log retardado. Son necesarios para reanudar la replicación tras restaurar los datos del esclavo. Si pierde los logs retardados pero todavía tiene el fichero `relay-log.info` , puede examinarlo para determinar hasta dónde ha ejecutado el flujo SQL en el log binario del maestro. Luego puede usar `CHANGE MASTER TO` con las opciones `MASTER_LOG_FILE` y `MASTER_LOG_POS` para decirle al esclavo que vuelva a leer los logs binarios a partir de ese punto. Por supuesto, esto requiere que los logs binarios existan en el servidor maestro.

Si su esclavo está sujeto a replicación de comandos `LOAD DATA INFILE` , también debe incluir en la copia de seguridad cualquier fichero `SQL_LOAD-*` que existe en el directorio que el esclavo usa para este propósito. El esclavo necesita estos ficheros para reanudar replicación de cualquier operación `LOAD DATA INFILE` interrumpida. La localización del directorio se especifica usando la opción `--slave-load-tmpdir` . Si valor por defecto, si no se especifica, es el valor de la variable `tmpdir` .

6.4. Cómo montar la replicación

Aquí hay una breve descripción de cómo inicializar una replicación completa de su servidor MySQL. Asume que quiere replicar todas las bases de datos en el maestro y no tiene una replicación préviamente configurarda. Necesita parar el servidor maestro brevemente para completar los pasos descritos aquí.

Este procedimiento está escrito en términos de inicializar un esclavo único, pero puede usarlo para inicializar múltiples esclavos.

Mientras este método es el más directo para inicializar un esclavo, no es el único. Por ejemplo, si tiene una muestra de los datos del maestro, y el maestro tiene su ID de servidor y el log binario activo, puede preparar un esclavo sin parar el maestro o incluso sin bloquear actualizaciones para ello. Para más detalles, consulte Sección 6.9, “Preguntas y respuestas sobre replicación”.

Si quiere administrar la inicialización de una replicación MySQL, sugerimos que lea este capítulo entero y pruebe todos los comandos mencionados en Sección 13.6.1, “Sentencias SQL para el control de servidores maestros” y Sección 13.6.2, “Sentencias SQL para el control de servidores esclavos”. También debe familiarizarse con las opciones de arranque de replicación descritas en Sección 6.8, “Opciones de arranque de replicación”.

Nota: este procedimiento y algunos de los comandos de replicación SQL mostrados en secciones posteriores necesita el privilegio `SUPER`.

1. Asegúrese de que las versiones de MySQL instalado en el maestro y en el esclavo son compatibles según dice la tabla mostrada en Sección 6.5, “Compatibilidad entre versiones de MySQL con respecto a la replicación”. Idealmente, debe usar la versión más reciente de MySQL en maestro y servidor.

   Por favor no reporte bugs hasta que ha verificado que el problema está presente en la última versión de MySQL.

2. Prepare una cuenta en el maestro que pueda usar el esclavo para conectar. Este cuenta debe tener el privilegio `REPLICATION SLAVE` . Si la cuenta se usa sólo para replicación (lo que se recomienda), no necesita dar ningún privilegio adicional. (Para información sobre preparar cuentas de usuarios y privilegios, consulte Sección 5.7, “Gestión de la cuenta de usuario MySQL”.)

   Suponga que su dominio es `mydomain.com` y que quiere crear una cuenta con un nombre de usuario de `repl` que puedan usar los esclavos para acceder al maestro desde cualquier equipo en su dominio usando una contraseña de `slavepass`. Para crear la cuenta, use el comando `GRANT`:

   ```
   mysql> GRANT REPLICATION SLAVE ON *.*
       -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
   ```

   Si quiere usar los comandos `LOAD TABLE FROM MASTER` o `LOAD DATA FROM MASTER` desde el servidor esclavo, necesita dar a esta cuenta privilegios adicionales:

   - De a la cuenta el privilegio global `SUPER` y `RELOAD` .
   - De el privilegio `SELECT` para todas las tablas que quiere cargar. Cualquier tabla maestra desde la que la cuenta no puede hacer un `SELECT` se ignoran por `LOAD DATA FROM MASTER`.

3. Si usa sólo tablas `MyISAM` , vuelque todas las tablas y bloquee los comandos de escritura ejecutando un comando `FLUSH TABLES WITH READ LOCK` :

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   Deje el cliente en ejecución desde el que lanza el comando `FLUSH TABLES` para que pueda leer los efectos del bloqueo. (Si sale del cliente, el bloqueo se libera.) Luego tome una muestra de los datos de su servidor maestro.

   La forma más fácil de crear una muestra es usar un programa de archivo para crear una copia de seguidad binaria de las bases de datos en su directorio de datos del maestro. Por ejemplo. use `tar` en Unix, o `PowerArchiver`, `WinRAR`, `WinZip`, o cualquier software similar en Windos. Para usar `tar` para crear un archivo que incluya todas las bases de datos, cambie la localización en el directorio de datos del maestro, luego ejecute el comando:

   ```
   shell> tar -cvf /tmp/mysql-snapshot.tar .
   ```

   Si quiere que el archivo sólo incluya una base de datos llamada `this_db`, use este comando:

```
shell> tar -cvf /tmp/mysql-snapshot.tar ./this_db
```

Luego copie el archivo en el directorio `/tmp` del servidor esclavo. En esa máquina, cambie la localización al directorio de datos del esclavo, y desempaquete el fichero usando este comando:

```
shell> tar -xvf /tmp/mysql-snapshot.tar
```

Puede no querer replicar la base de datos `mysql` si el servidor esclavo tiene un conjunto distinto de cuentas de usuario a la existente en el maestro. En tal caso, debe excluírla del archivo. Tampoco necesita incluir ningún fichero de log en el archivo, o los ficheros `master.info` o `relay-log.info` files.

Mientras el bloqueo de `FLUSH TABLES WITH READ LOCK` está en efecto, lee el valor del nombre y el desplazamiento del log binario actual en el maestro:

```
mysql > SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

La columna `File` muestra el nombre del log, mientras que `Position` muestra el desplazamiento. En este ejemplo, el valor del log binario es `mysql-bin.003` y el desplazamiento es 73. Guarde los valores. Los necesitará más tarde cuando inicialice el servidor. Estos representan las coordenadas de la replicación en que el esclavo debe comenzar a procesar nuevas actualizaciones del maestro.

Una vez que tiene los datos y ha guardado el nombre y desplazamiento del log, puede reanudar la actividad de escritura en el maestro:

```
mysql> UNLOCK TABLES;
```

Si está usando tablas `InnoDB` , debería usar la herramienta `InnoDB Hot Backup`. Realiza una copia consistente sin bloquear el servidor maestro, y guarda el nombre y desplazamiento del log que se corresponden a la copia para usarlo posteriormente en el esclavo. `InnoDB Hot Backup` es una herramienta no libre (comercial) que no está incluída en la distribución de MySQL estándar. Consulte la página web de `InnoDB Hot Backup` en http://www.innodb.com/manual.php para información detallada.

Sin la herramienta `Hot Backup` , la forma más rápida de hacer una copia binaria de los datos de las tablas `InnoDB` es parar el maestro y copiar los ficheros de datos `InnoDB`, ficheros de log, y ficheros de definición de tablas (ficheros `.frm`). Para guardar los nombres de ficheros actual y desplazamientos, debe ejecutar el siguiente comando antes de parar el servidor:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Luego guarde el nombre del log y el desplazamiento desde la salida de `SHOW MASTER STATUS` como se mostró antes. Tras guardar el nombre del log y el desplazamiento, pare el servidor *sin* bloquear las tablas para asegurarse que el servidor para con el conjunto de datos correspondiente al fichero de log correspondiente y desplazamiento:

```
shell> mysqladmin -u root shutdown
```

Una alternativa que funciona para tablas `MyISAM` y `InnoDB` es realizar un volcado SQL del maestro en lugar de una copia binaria como se describe en la discusión precedente. Para ello, puede usar `mysqldump --master-data` en su maestro y cargar posteriormente el fichero de volcado SQL en el esclavo. Sin embargo, esto es más lento que hacer una copia binaria.

Si el maestro se ha ejecutado previamente sin habilitar `--log-bin` , el nombre del log y las posiciones mostradas por `SHOW MASTER STATUS` o `mysqldump --master-data` están vacíos. En ese caso, los valores que necesita usar posteriormente cuando especifica el fichero de log del esclavo y la posición son una cadena vacía (`''`) y `4`.

4. Asegúrese que la sección `[mysqld]` del fichero `my.cnf` en el maestro incluye una opción `log-bin` . Esta sección debe también tener la opción `server-id=master_id` , donde `master_id` debe ser un entero positivo de 1 a 2^32 - 1. Por ejemplo:

   ```
   [mysqld]
   log-bin=mysql-bin
   server-id=1
   ```

   Si estas opciones no están presentes, añádalas y reinicie el servidor.

5. Pare el servidor que se vaya a usar como esclavo y añada lo siguiente a su fichero `my.cnf` :

   ```
   [mysqld]
   server-id=slave_id
   ```

   El valor `slave_id` , como el valor `master_id` , debe ser un entero positivo de 1 a 2^32 - 1. Además, es muy importante que el ID del esclavo sea diferente del ID del maestro. Por ejemplo:

   ```
   [mysqld]
   server-id=2
   ```

   Si está preparando varios esclavos, cada uno debe tener un valor de `server-id` único que difiera del maestro y de cada uno de los otros esclavos. Piense en los valores de `server-id` como algo similar a las direcciones IP: estos IDs identifican unívocamente cada instancia de servidor en la comunidad de replicación.

   Si no especifica un `server-id`, se usa 1 si no ha definido un `master-host`, de otro modo se usa 2. Tenga en cuenta que en caso de omisión de `server-id`, un maestro rechaza conexiones de todos los esclavos, y un esclavo rechaza conectar a un maestro. Por lo tanto, omitir el `server-id` es bueno sólo para copias de seguridad con un log binario.

6. Si ha hecho una copia de seguridad binara de los datos del maestro, cópielo en el directorio de datos del esclavo antes de arrancar el esclavo. Asegúrese que los privilegios en los ficheros y directorios son correctos. El usuario que ejecuta el servidor MySQL debe ser capaz de leer y escribir los ficheros, como en el maestro.

   Si hizo una copia de seguridad usando `mysqldump`, arranque primero el esclavo (consulte el siguiente paso).

7. Arranque el esclavo. Si ha estado replicando préviamente, arranque el esclavo con la opción `--skip-slave-start` para que no intente conectar inmediatamente al maestro. También puede arrancar el esclavo con la opción `--log-warnings` (activada por defecto en MySQL 5.0), para obtener más mensajes en el log de errores acerca de problemas (por ejemplo, problemas de red o conexiones). En MySQL 5.0, las conexiones abortadas no se loguean en el log de errores a no ser que el valor sea mayor que 1.

8. Si hace una copia de seguridad de los datos del maestro usando `mysqldump`, cargue el fichero de volcado en el esclavo:

```
shell> mysql -u root -p < dump_file.sql
```

9. Ejecute los siguientes comandos en el esclavo, reemplazando los valores de opciones con los valores relevantes para su sistema:

```
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;
```

   La siguiente tabla muestra la longitud máxima para las opciones de cadenas de caracteres:

`MASTER_HOST`	60
`MASTER_USER`	16
`MASTER_PASSWORD`	32
`MASTER_LOG_FILE`	255

10. Arranque el flujo esclavo:

```
mysql> START SLAVE;
```

Una vez realizado este procedimiento, el esclavo debe conectar con el maestro y atapar cualquier actualización que haya ocurrido desde que se obtuvieron los datos.

Si ha olvidado asignar un valor para `server-id` en el maestro, los esclavos no son capaces de conectar.

Si olvida asignar un valor para `server-id` en el esclavo, obtiene el siguiente error en el log de errores:

```
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.
```

También encuentra mensajes de error en el log de errores del esclavo si no es capaz de replicar por ninguna otra razón.

Una vez que un esclavo está replicando, puede encontrar en su directorio de datos un fichero llamado `master.info` y otro llamado `relay-log.info`. El esclavo usa estos dos ficheros para saber hasta que punto ha procesado el log binario del maestro. **No** borre o edite estos ficheros, a no ser que realmente sepa lo que hace y entienda las implicaciones. Incluso en tal caso, es mejor que use el comando `CHANGE MASTER TO`.

Nota: El contenido de `master.info` subedita algunas de las opciones especificadas en línea de comandos o en `my.cnf`. Consulte Sección 6.8, “Opciones de arranque de replicación” para más detalles.

Una vez que tiene una copia de los datos, puede usarlo para actualizar otros esclavos siguiendo las porciones del procedimiento descrito. No necesita otra muestra de los datos del maestro; puede usar la misma para todos los esclavos.

Nota: para la mayor durabilidad y consistencia posible en una inicialización de replicación usando `InnoDB` con transacciones debe usar `innodb_flush_logs_at_trx_commit=1`, `sync-binlog=1`, y `innodb_safe_binlog` en su fichero `my.cnf` del maestro.

6.5. Compatibilidad entre versiones de MySQL con respecto a la replicación

Nota: El formato del log binario tal y como se implementa en MySQL 5.0 es considerablemente distinto al usado en versiones previas. Los cambios principales se hicieron en MySQL 5.0.3 (para mejoras en el tratamiento de conjuntos de caracteres y `LOAD DATA INFILE`) y 5.0.4 (para mejoras en el tratamiento de zonas horarias). Esto tiene consecuencias tremendas en actualizar servidores en un entorno de replicación, como se describe en Sección 6.6, “Aumentar la versión de la replicación”.

Recomendamos usar la versión de MySQL más reciente disponible, ya que las capacidades de replicación se mejoran contínuamente. También recomendamos usar la misma versión para el maestro y el servidor. Recomendamos actualizar el maestro y esclavo que tengan versiones alfa o beta a versiones de producción. La replicación desde un maestro con versión 5.0.3 a un esclavo 5.0.2 falla; dsde un maestro 5.0.4 a un esclavo 5.0.3 también fallará. En general, los esclavos ejecutando MySQL 5.0.x pueden usarse con maestro antiguos (incluso aquéllos ejecutando MySQL 3.23, 4.0, or 4.1), pero no al revés.

La información precedente pertenece a la compatibilidad de replicación a nivel de protocolo. También puede haber restricciones de compatibilidad a nivel de SQL, como se discute en Sección 6.7, “Características de la replicación y problemas conocidos”.

6.6. Aumentar la versión de la replicación

Cuando actualiza servidores que participan en una actualización de replicación, el procedimiento para actualizar depende en la versión actual del servidor y la versión que está actualizando.

6.6.1. Aumentar la versión de la replicación a 5.0

Esta sección se aplica para actualizar replicaciones de MySQL 3.23, 4.0, o 4.1 a 5.0. Un servidor 4.0 debe ser 4.0.3 o posterior.

Cuando actualice un maestro desde MySQL 3.23, 4.0, o 4.1 a 5.0, primero debe asegurarse que todos los esclavos de su maestro están usando la misma versión 5.0.x . Si no es así, debe primero actualizar todos los esclavos. Para actualizar cada esclavo, párelo, actualícelo a la versión 5.0.x apropiado, reinícielo, y reinicie la replicación. El esclavo 5.0 es capaz de leer los logs retardados antiguos escritos antes de la actualización y ejecutar los comandos que contiene. Los logs retardados creados por el esclavo tras la actualización están en formato 5.0.

Una vez que se han actualizado los esclavos, pare el maestro, actualícelo a la misma versión 5.0.x que los esclavos, y reinícielo. El maestro 5.0 es capaz de leer los antiguos logs binarios escritos antes de la actualización y enviarlos a los esclavos 5.0. Los esclavos reconocen el antiguo formato y lo tratan apropiadamente. Los logs binarios creados por el maestro después de la actualización están en formato 5.0. Estos también son reconocidos por los esclavos 5.0.

En otras palabras, no hay medidas a tomar cuando se actualiza a 5.0, excepto que los esclavos deben ser 5.0 antes que pueda actualizar el maestro a 5.0. Tenga en cuenta que bajar de una versión 5.0 a una antigua no funciona tan sencillamente: debe asegurarse que cualquier log binario 5.0 o log retardado ha sido procesado por completo, para que pueda borrarlos antes de cambiar de versión.

6.7. Características de la replicación y problemas conocidos

En general, la compatibilidad de la replicación en nivel SQL requiere que cualquier característica usada sea soportado tanto por el maestro como por los servidores esclavos. Por ejemplo, la función `TIMESTAMPADD()` se implementó en MySQL 5.0.0. Si usa esta función en el maestro, no puede replicar a un servidor esclavo que sea más antiguo que MySQL 5.0.0. Si planea usar replicación entre 5.0 y

versiones prévias de MySQL debe consultar el Manual de referencia de MySQL 4.1 para información acerca de las características de replicación en versiones prévias de MySQL.

La siguiente lista proporciona detalles acerca de qué se soporta y qué no. Información específica adicional de `InnoDB` acerca de replicación se da en Sección 15.6.5, “`InnoDB` y replicación MySQL”. Aspectos de replicación acerca de rutinas almacenadas y disparadores se describen en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

- La replicación se da correctamente con `AUTO_INCREMENT`, `LAST_INSERT_ID()`, y `TIMESTAMP` .
- Las funciones `USER()`, `UUID()`, y `LOAD_FILE()` se replican sin cambios y no funcionan correctamente con el esclavo.
- Las funciones que tratan bloqueos a nivel de usuario: `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, `IS_USED_LOCK()` se replican sin que el esclavo sepa el contexto de concurrencia del maestro; así que estas funciones no deben usarse para insertar en una tabla del maestro ya que el contexto del esclavo puede diferir (p.e. no haga `INSERT INTO mytable VALUES(GET_LOCK(...))`).
- Las variables `FOREIGN_KEY_CHECKS`, `SQL_MODE`, `UNIQUE_CHECKS`, and `SQL_AUTO_IS_NULL` se replican todas en MySQL 5.0. La variable `TABLE_TYPE`, también conocida como `STORAGE_ENGINE` no se replica todavía, lo que es bueno para replicación entre distintos motores de almacenamiento.
- A partir de MySQL 5.0.3 (maestro y esclavo), la replicación funciona bien incluso si el maestro y el esclavo tienen distintos conjuntos de caracteres globalres. A partir de MySQL 5.0.4 (maestro y esclavo), la replicación funciona bien incluso si el maestro y el esclavo tienen distintas variables de zonas horarias.
- Lo siguiente se aplica para replicación entre servidores MySQL usando distintos conjuntos de caracteres:

  1. Debe **siempre** usar las mismas colaciones y conjuntos de caracteres **globales** (`--default-character-set`, `--default-collation`) en el maestro y esclavo. De otro modo, puede obtener errores de claves duplicadas en el esclavo, debido a que una clave que se trata como única en el conjunto de caracteres del maestro puede no ser único en el conjunto de caracteres del esclavo.

  2. Si el maestro es anterior a MySQL 4.1.3, el conjunto de caracteres de la sesión nunca debería ser distinto al del valor global (en otras palabras, no use `SET NAMES`, `SET CHARACTER SET`, y así) ya que este cambio del conjunto de caracteres no es conocido por el esclavo. Si tanto el maestro coom el esclavo son de la versión 4.1.3 o posterior, la sesión puede cambiar los valores locales del conjunto de caracteres (tales como `NAMES`, `CHARACTER SET`, `COLLATION_CLIENT`, y `COLLATION_SERVER`) ya que estos cambios se escriben en el log binario y son conocidos por el esclavo. Sin embargo, la sesión no puede cambiar estos valores **globales** ya que el maestro y esclavo deben tener conjuntos de caracteres idénticos.

  3. Si tiene bases de datos en el maestro con distintos conjuntos de caracteres al valor global de `collation_server` , debe diseñar su comando `CREATE TABLE` que no se base en el conjunto de caracteres por defecto de la base de datos, ya que actualmente hay un bug (Bug #2326); una solución es poner el conjunto de caracteres y colación explícitamente en `CREATE TABLE`.

- Tanto el maestro como el esclavo deben tener la misma zona horaria. De otro modo algunos comandos, por ejemplo comandos que usen `NOW()` o `FROM_UNIXTIME()` no se replicarán apropiadamente. Se podría poner una zona horaria en que el servidor MySQL se ejecute usando la opción `--timezone=`*`timezone_name`* del script `mysqld_safe` o asignando un valor a la variable de entorno `TZ`. Tanto el maestro como el esclavo deben tener la misma zona horaria para las conexiones; esto es, el parámetro `--default-time-zone` debe tener el mismo valor para maestro y servidor.

- `CONVERT_TZ(...,...,@global.time_zone)` no se replica apropiadamente. `CONVERT_TZ(...,...,@session.time_zone)` se replica apropiadamente sólo si el maestro y esclavo son de la versión 5.0.4 o posterior.

- Las variables de sesión no se replican apropiadamente cuando se usan en comandos que actualizan tablas; por ejemplo: `SET MAX_JOIN_SIZE=1000; INSERT INTO mytable VALUES(@MAX_JOIN_SIZE);` no insertará los mismos datos en el maestro y el esclavo. Esto no se aplica a `SET TIME_ZONE=...; INSERT INTO mytable VALUES(CONVERT_TZ(...,...,@time_zone))`, que se arregla en MySQL 5.0.4.

- Es posible replicar tablas transaccionales en el maestro usando tablas no transaccionales en el esclavo. Por ejemplo, puede replicar una tabla maestra `InnoDB` como una tabla esclava `MyISAM` . Sin embargo, si lo hace, hay problemas si el esclavo se para en medio de un bloque `BEGIN`/`COMMIT` , ya que el esclavo reinicia al principio del bloque `BEGIN` . Este tema se encuentra en la lista de temas pendientes y se arreglará próximamente.

- Los comandos de actualización que se refieren a variables de usuario (esto es, variables de la forma *`@var_name`*) se replican correctamente en MySQL 5.0; sin embargo esto no es cierto para versiones prévias a 4.1. Tenga en cuenta que los nombres de las variables de usuario no son sensibles a mayúsculas desde MySQL 5.0; debe tener esto en cuenta cuando prepare una replicación entre 5.0 y versiones antiguas.

- Los esclavos MySQL 5.0 pueden conectar a maestros 5.0 usando SSL.

- En MYSQL 5.0 (desde 5.0.3), hay una variable de sistema global `slave_transaction_retries`: Si el flujo SQL del esclavo de replicación falla al ejecutar una transacción debido a un deadlock `InnoDB` o excede el `innodb_lock_wait_timeout` de InnoDB o `TransactionDeadlockDetectionTimeout` o `TransactionInactiveTimeout` de NDB, automáticamente reintenta `slave_transaction_retries` veces antes de parar con un error. El valor por defecto en MySQL 5.0 es 10. A partir de MySQL 5.0.4, el número total de reintentos pueden verse en la salida de `SHOW STATUS`; consulte Sección 5.3.4, “Variables de estado del servidor”.

- Si `DATA DIRECTORY` o `INDEX DIRECTORY` se usa en un comando `CREATE TABLE` en el maestro, la cláusula se usa en el esclavo. Esto puede causar problemas si no existe el directorio correspondiente en el sistema de ficheros del esclavo o existe pero no es accesible en el esclavo. MySQL 5.0 soporta una opción `sql_mode` llamada `NO_DIR_IN_CREATE`. Si el esclavo se ejecuta con este modo SQL , ignora estas cláusulas al replicar el comando `CREATE TABLE` . El resultado es que los datos `MyISAM` y ficheros índice se crean en el directorio de la base de datos de la tabla.

- Es posible que los datos del maestro y el esclavo diverjan si se diseña una consulta tal que la modificación de los datos no sea determinista; esto es, si se deja a criterio del optimizador de consultas. (Esto no es generalmente una buena práctica en ningún caso, incluso fuera de la replicación.) Para una explicación detallada de este tema consulte Sección A.8.4, “Cuestiones abiertas en MySQL”.

- *Lo siguiente se aplica sólo si el maestro o el esclavo están ejecutando la versión 5.0.3 o anterior*: Si se interrumpe un `LOAD DATA INFILE` en el maestro (violación de integridad, conexión muerta, o así), el esclavo ignora el `LOAD DATA INFILE` totalmente. Esto significa que si este comando inserta o actualiza registros en tablas de forma permanente antes de interrumpirse, estas modificaciones no se replican en el esclavo.

- `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, y `FLUSH TABLES WITH READ LOCK` no se loguean ya que cualquiera de ellos puede causar problemas al replicarse en un esclavo.) Para un ejemplo de sintaxis, consulte Sección 13.5.5.2, “Sintaxis de FLUSH”. En MySQL 5.0, `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, y `REPAIR TABLE` se escriben en el log binario y por lo tanto se replican en los esclavos. Esto no es un problema normalmente porque estos comandos no modifican los datos de las tablas. Sin embaargo, esto puede causar dificultades bajo ciertas circunstancias. Si replica las tablas

de privilegios en la base de datos `mysql` y actualiza estas tablas directamente sin usar `GRANT`, debe ejecutar un comando `FLUSH PRIVILEGES` en los esclavos para poner los nuevos privilegios en efecto. Además, si usa `FLUSH TABLES` cuando queda una tabla `MyISAM` que es parte de una tabla `MERGE` , debe ejecutar un `FLUSH TABLES` manualmente en los esclavos. En MySQL 5.0, estos comandos se escriben en el log binario a no ser que especifique `NO_WRITE_TO_BINLOG` o su alias `LOCAL`.

- MySQL sólo soporta un maestro y varios esclavos. En el futuro planeamos añadir un algoritmo de voto para cambiar el maestro automáticamente en caso de problemas con el maestro actual. También planeamos introducir procesos agentes para ayudar a realizar balanceo de carga mandando consultas `SELECT` a distintos esclavos.

- Cuando un servidor para y reinicia, sus tablas `MEMORY` (`HEAP`) se vacían . En MySQL 5.0, el maestro replica este efecto como sigue: La primera vez que el maestro usa cada tabla `MEMORY` tras arrancar, lo notifica a los esclavos que la tabla necesita vacíar escribiendo un comando `DELETE FROM` para esa tabla en el log binario. Consulte Sección 14.3, “El motor de almacenamiento `MEMORY` (`HEAP`)” para más información.

- Las tablas temporales se replican excepto en el caso donde para el esclavo (no sólo los flujos esclavos) y ha replicado tablas temporales que se usan en actualizaciones que no se han ejecutado todavía en el esclavo. Si para el esclavo, las tablas temporales necesitadas por estas actualizaciones no están disponibles cuando el esclavo se reinicia. Para evitar este problema, no pare el esclavo mientras tiene tablas temporales abiertas. En lugar de eso, use el siguiente procedimiento:

  1. Ejecute un comando `STOP SLAVE` .
  2. Use `SHOW STATUS` para chequear el valor de la variable `Slave_open_temp_tables` .
  3. Si el valor es 0, ejecute un comando `mysqladmin shutdown` para parar el esclavo.
  4. Si el valor no es 0, reinicie los flujos esclavos con `START SLAVE`.
  5. Repita el procedimiento posteriormente para comprobar si tiene mejor suerte la próxima vez.

  Planeamos arreglar este problema en el futuro cercano.

- Es correcto conectar servidores de modo circular en una relación maestro/esclavo con la opción `--log-slave-updates`. Tenga en cuenta, sin embargo, que varios comandos no funcionan correctamente en esta clase de inicialización a no ser que su código cliente esté escrito para tener en cuenta que pueden ocurrir actualizaciones en distintas secuencias de diferentes servidores.

  Esto significa que puede crear una inicialización como esta:

  ```
  A -> B -> C -> A
  ```

  Los IDs de los servidores se codifican en los logs binarios de eventos, así que el servidor A conoce cuando un evento que lee fue creado originalmente por sí mismo y no ejecuta el evento (a no ser que el servidor A se iniciara con la opción `--replicate-same-server-id`, que tiene significado sólo en inicializaciones raras). Por lo tanto, no hay bucles infinitos. Sin embargo, esta inicialización circular funciona sólo si no realiza actualizaciones conflictivas entre tablas. En otras palabras, si inserta datos tanto en A y C, no debe insertar un registro en A que pueda tener una clave que entre en conflicto con un registro insertado en C. Tampoco debe actualizar el mismo registro en dos servidores si el orden de las actualizaciones es significativo.

- Si un comando en el esclavo produce un error, el flujo esclavo SQL termina, y el esclavo escribe un mensaje en su log de errores. Luego debe conectar al esclavo manualmente, arreglar el problema (por ejemplo, una tabla no existente), y luego ejecutar `START SLAVE`.

- Es correcto parar un maestro y reiniciarlo posteriormente. Si un esclavo pierde su conexión con el maestro, el esclavo trata de reconectar inmediatamente. Si falla, el esclavo reintenta periódicamente. (Por defecto reinicia cada 60 segundos. Esto puede cambiarse con la opción `--master-connect-retry` .) El esclavo también es capaz de tratar con problemas de conectividad de red. Sin embargo, el esclavo se da cuenta del problema de red sólo tras no recibir datos del maestro durante `slave_net_timeout` segundos. Si los problemas son cortos, puede decrementar `slave_net_timeout`. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

- Parar el esclavo (correctamente) es seguro, ya que toma nota de dónde lo dejó. Las paradas no correctas pueden producir problemas, especialmente si la caché de disco no se volcó a disco antes que parara el sistema. La tolerancia a fallos del sistema se incrementa generalmente si tiene proveedores de corriente ininterrumpidos. Las paradas no correctas del maestro pueden causar inconsistencias entre los contenidos de tablas y el log binario en el maestro; esto puede evitarse usando tablas `InnoDB` y la opción `--innodb-safe-binlog` en el maestro. Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

- Debido a la naturaleza no transaccional de las tablas `MyISAM`, es posible tener un comando que actualice parcialmente una tabla y retorne un código de error. Esto puede ocurrir, por ejemplo, en una inserción de varios registros que tenga un registro que viole una clave, o si una actualización larga se mata tras actualizar algunos de los registros. Si esto ocurre en el maestro, el flujo esclavo sale y para hasta que el administrador de base de datos decida qué hacer acerca de ello a no ser que el código de error se legitime y la ejecución del comando resulte en el mismo código de error. Si este comportamiento de validación de código de error no es deseable, algunos o todos los errores pueden ser ignorados con la opción `--slave-skip-errors` .

- Si actualiza tablas transaccionales para tablas no transaccionales dentro de una secuencia `BEGIN`/`COMMIT` , las actualizaciones del log binario pueden desincronizarse si la tabla no transaccional se actualiza antes de que acabe la transacción. Esto se debe a que la transacción se escribe en el log binario sólo cuando acaba.

- En situaciones donde las transacciones mezclan actualizaciones transaccionales y no transaccionales, el orden de los comandso en el log binario es correcto , y todos los comandos necesarios se escriben en el log binario incluso en caso de un `ROLLBACK`). Sin embargo, cuando una segunda conexión actualiza la tabla no transaccional antes que la primera transacción se complete, los comandos pueden loguearse fuera de orden, ya que la actualización de la segunda conexión se escribe inmediatamente al ejectarse, sin tener en cuenta el estado de la transacción que se ejecuta en la primera conexión.

6.8. Opciones de arranque de replicación

Tanto en el maestro como el esclavo, debe usar la opción `server-id` para establecer un ID de replicación único para cada servidor. Debe elegir un entero positivo único en el rango de 1 a 2^32 - 1 para cada maestro y esclavo. Ejemplo: `server-id=3`

Las opciones que puede usar en el maestro para controlar el logueo binario se describen en Sección 5.10.3, “El registro binario (Binary Log)”.

La siguiente tabla describe las opciones que puede usar en servidores esclavos de replicación MySQL 5.0. Puede especificar estas opciones por línea de comandos o en un fichero de opciones.

Algunas opciones de replicación del esclavo se tratan de forma especial, en el sentido que se ignoran si existe un fichero `master.info` cuando el esclavo arranca y contiene valores para las opciones. Las siguientes opciones se tratan de este modo:

- `--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`

El fichero `master.info` en formato 5.0 incluye valores correspondientes a las opciones SSL. Además, el formato del fichero incluye como primer línea el número de líneas en el fichero. Si actualiza de un servidor antiguo a uno nuevo, el nuevo servidor actualiza el fichero `master.info` al nuevo formato automáticamente cuando arranca. Sin embargo, si baja de versión un servidor nuevo a una versión antigua, debe borrar la primera línea manualmente antes de arrancar el servidor antiguo la primera vez.

Si no existe un fichero `master.info` cuando arranca el esclavo, usa los valores para estas opciones que se especifican en el fichero de opciones o en línea de comandos. Esto ocurre cuando arranca el servidor como un esclavo de replicación la primera vez, o cuando ha ejecutado `RESET SLAVE` y luego ha parado y rearrancado el esclavo.

Si el fichero `master.info` existe cuando el esclavo arranca, el servidor ignora estas opciones. En su lugar, usa los valores encontrados en el fichero `master.info` .

Si reinicia el esclavo con valores distintos de las opciones de arranque que corresponden a los valores en el fichero `master.info` , los valores diferentes no tienen efecto, ya que el servidor continúa usando el fichero `master.info` . Para usar valores distintos, debe reiniciar tras borrar el fichero `master.info` o (preferiblemente) use el comando `CHANGE MASTER TO` para reiniciar los valores mientras el esclavo está corriendo.

Soponga que especifica estas opciones en su fichero `my.cnf` :

```
[mysqld]
master-host=some_host
```

La primera vez que arranca el servidor como esclavo de replicación, lee y usa esta opción del fichero `my.cnf`. El servidor guarda el valor en el fichero `master.info` . La siguiente vez que arranque el servidor, lee el valor de la máquina maestra sólo desde el fichero `master.info` e ignora el valor en el fichero de opciones. Si modifica el fichero `my.cnf` para especificar un equipo maestro distinto a `some_other_host`, el cambio todavía no tiene efecto. Debe usar `CHANGE MASTER TO` en su lugar.

Debido a que el servidor da una precedencia al fichero `master.info` sobre las opciones de arranque descritas, puede no usar las opciones de arranque para estos valores, y en su lugar especificarlos usando el comando `CHANGE MASTER TO` . Consulte Sección 13.6.2.1, “Sintaxis de `CHANGE MASTER TO`”.

Este ejemplo muestra un uso más extensivo de las opciones de arranque para configurar un esclavo:

```
[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
```

La siguiente lista describe opciones de arranque para controlar la replicación: Muchas de estas opciones pueden cambiarse con el servidor en ejecución usando el comando `CHANGE MASTER TO`. Otras, tales como las opciones `--replicate-*`, sólo pueden cambiarse cuando arranca el esclavo. Planeamos arreglar esto.

- `--log-slave-updates`

  Normalmente, las actualizaciones recibidas por un servidor maestro por un esclavo no se loguean en el log binario. Esta opción le dice al esclavo que loguee las actualizaciones realizadas por el flujo SQL en el log binario del esclavo. Para que esta opción tenga efecto, el esclavo debe arrancarse con la opción `--log-bin` para permitir logueo binario. `--log-slave-updates` se usa cuando quiere encadenar servidores de replicación . Por ejemplo, puede hacer una inicialización como esta:

  ```
  A -> B -> C
  ```

  Esto es , A sirve como maestro para el esclavo B, y B sirve como maestro para el esclavo C. Para que funcione, B debe ser tanto maestro *como* esclavo. Debe arrancar tanto A como B con `--log-bin` para permitir logueo binario, y B con la opción `--log-slave-updates` .

- `--log-warnings`

  Hace que el esclavo muestre más mensajes en el log de errores acerca de qué está haciendo. Por ejemplo, le advierte que ha podido reconectar tras un error de red, y le informa cada vez que arrancan los flujos del esclavo. Esta opción está activada por defecto en MySQL 5.0; para desactivarla, use `--skip-log-warnings`. En MySQL 5.0, las conexiones abortadas no se loguean en el log de errores a no ser que el valor sea mayor que 1.

  Tenga en cuenta que los efectos de esta opción no están limitados a replicación. Produce advertencias a través de un espectro de actividades de servidor.

- `--master-connect-retry=`*`seconds`*

  Número de segundos que el flujo esclavo duerme antes de reintentar conectar al maestro en caso que caiga el maestro o se pierda la conexión. El valor en el fichero `master.info` tiene precedencia si puede leerse. Si no está especificado, por defecto es 60.

- `--master-host=`*`host`*

  El nombre de equipo o número IP del maestro. Si no se da esta opción, el flujo esclavo no arranca. El valor en `master.info` tiene precedencia si puede leerse.

- `--master-info-file=`*`file_name`*

  Nombre a usar para el fichero en que el esclavo guarda información acerca del maestro. El nombre por defecto es `mysql.info` en el directorio de datos.

- `--master-password=`*`password`*

  Contraseña de la cuenta que el flujo esclavo usa para autenticar al conectar al maestro. El valor en el fichero `master.info` tiene precedencia si puede leerse. Si no está asignado, se asume la contraseña vacía.

- `--master-port=port_number`

  El puerto TCP/IP en que está escuchando el maestro. El valor en el fichero `master.info` tiene precedencia si puede leerse. Si no está asignado, se usa la especificación compilada. Si no ha cambiado las opciones de `configure` debería ser 3306.

- `--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`

  Estas opciones se usan para preparar una conexión de replicación segura al maestro usando SSL. Sus significados son los mismos que los de las opciones correspondientes `--ssl`, `--ssl-ca`, `--ssl-capath`, `--ssl-cert`, `--ssl-cipher`, `--ssl-key` descritas en Sección 5.7.7.5, “Opciones relativas a SSL”. Los valores en el fichero `master.info` tienen precedencia si pueden leerse.

- `--master-user=username`

  El nombre de usuario que el flujo esclavo usa para autenticar al conectar con el maestro. En MySQL 5.0, esta cuenta debe tener el privilegio `REPLICATION SLAVE` . El valor en el fichero `master.info` , si puede leerse, tiene precedencia. Si el usuario maestro no está inicializado, se asume el usuario `test` .

- `--max-relay-log-size=#`

  Para rotar el log retardado automáticamente. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

- `--read-only`

  Esta opción hace que el esclavo no permita ninguna actualización excepto de flujos esclavos o de usuarios con el privilegio `SUPER` . Esto puede ser útil para asegurar que un servidor esclavo no acepta actualizaciones de los clientes.

- `--relay-log=file_name`

  El nombre para el log retardado. El nombre por defecto es *`host_name`*`-relay-bin.`*`nnnnnn`*, donde *`host_name`* es el nombre del esclavo y *`nnnnnn`* indica que los logs retardados se crean en secuencia enumerada. Puede especificar la opción para crear nombres de logs retardados independientes del nombre de la máquina, o si sus logs retardados tieneden a ser grandes (y no quiere decrementar `max_relay_log_size`) y necesita ponerlos en algún área distinta del directorio de datos, o si quiere incrementar la velocidad balanceando carga entre discos.

- `--relay-log-index=file_name`

  Localización y nombre que deben usarse para el fichero índice del log retardado. El nombre por defecto es *`host_name`*`-relay-bin.index`, donde *`host_name`* es el nombre del esclavo.

- `--relay-log-info-file=file_name`

  El nombre a usar por el fichero en que el esclavo guarda información acerca del log retardado. El nombre por defecto es `relay-log.info` en el directorio de datos.

- `--relay-log-purge={0|1}`

  Desactiva o activa la purga automática del log retardado en cuanto no se necesitan más. El valor por defecto es 1 (activado). Esta es una variable global que puede cambiarse dinámicamente con `SET GLOBAL relay_log_purge`.

- `--relay-log-space-limit=#`

Especifica un límite superior del tamaño total de todos los logs retardados en el esclavo (un valor de 0 significa que no hay limitación de tamaño). Esto es útil para un esclavo con espacio de disco limitado. Cuando se alcanza el límite, el flujo de entrada/salida para de leer eventos del log binario del maestro hasta que el flujo SQL borra algunos logs retardados no usados. Tenga en cuenta que este límite no es absoluto: Hay casos donde el flujo SQL necesita más eventos antes que pueda borrar logs retardados. En tal caso, el flujo de entrada/salida excede el límite hasta que es posible para el flujo SQL borrar algunos logs retardados. (El no hacerlo causaría un deadlock.) No debe asignar `--relay-log-space-limit` un valor menor al doble del valor de `--max-relay-log-size` (o `--max-binlog-size` si `--max-relay-log-size` es 0). En tal caso, hay una oportunidad que el flujo de entrada/salida espere espacio libre debido a que se excede `--relay-log-space-limit`, pero el flujo SQL no tiene log retardado para purgar y es incapaz de satisfacer el flujo de entrada/salida. Esto fuerza al flujo de entrada/salida a que ignore temporalmente `--relay-log-space-limit`.

- `--replicate-do-db=`*`db_name`*

  Le dice al esclavo que restrinja replicación a comandos donde la base de datos por defecto (esto es, la seleccionada por `USE`) es *`db_name`*. Para especificar más de una base de datos, use esta opción múltiples veces, una para cada base de datos. Tenga en cuenta que esto no replica comandos entre bases de datos tales como `UPDATE` *`some_db.some_table`* `SET foo='bar'` mientras se haya seleccionado una base de datos distinta o ninguna base de datos. Si necesita actualizaciones entre bases de datos distintas use `--replicate-wild-do-table=`*`db_name`*`.%`. Por favor lea las notas que hay a continuación de esta lista de opciones.

  Un ejemplo de lo que no funciona como podría esperar: Si el esclavo arranca con `--replicate-do-db=sales` y realiza el siguiente comando en el maestro, el comando `UPDATE` *no* se replica:

  ```
  USE prices;
  UPDATE sales.january SET amount=amount+1000;
  ```

  Si necesita que funcionen actualizaciones entre varias bases de datos, use `--replicate-wild-do-table=`*`db_name`*`.%` en su lugar.

  La razón principal para este comportamiento “sólo chequee la base de datos por defecto ” es que es difícil para el comando conocer si debe o no debe ser replicado (por ejemplo, si está usando comandos `DELETE` de múltiples tablas o comandos `UPDATE` de múltiples tablas que actúan entre múltiples bases de datos). También es más rápido chequear sólo la base de datos por defecto en lugar de todas las bases de datos si no hay necesidad.

- `--replicate-do-table=`*`db_name.tbl_name`*

  Le dice al flujo esclavo que restrinja replicación a la tabla especificada. Para especificar más de una tabla, use esta opción múltiples veces, una para cada tabla. Esto funciona para actualizaciones entre bases de datos, en contraste con `--replicate-do-db`. Lea los comentarios a continuación de esta lista de opciones.

- `--replicate-ignore-db=`*`db_name`*

  Le dice al esclavo que no replique ningún comando donde la base de datos por defecto (esto es, la seleccionada por `USE`) es *`db_name`*. Para especificar más de una base de datos a ignorar, use esta opción múltiples veces, una para cada base de datos. No debe usar esta opción si está usando actualizaciones entre bases de datos y no quiere que estas acutalizaciones se repliquen. Lea las notas después de esta lista de opciones.

Une ejemplo de lo que no funciona como podría esperar: Si el esclavo arranca con `--replicate-ignore-db=sales` y ejecuta el siguiente comando en el maestro, el comando `UPDATE` se replica *not* :

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

Si necesita que funcionen actualizaciones entre bases de datos, use `--replicate-wild-ignore-table=db_name.%` en su lugar.

- `--replicate-ignore-table=db_name.tbl_name`

  Le dice al esclavo que no replique ningún comando que actualice la tabla especificada (incluso si otras tablas se actualizan en el mismo comando). Para especificar más de una tabla a ignorar, use esta opción múltiples veces, una para cada tabla. Esto funciona para actualizaciones entre bases de datos, en contraste con `--replicate-ignore-db`. Lea los comentarios a continuación de esta lista de opciones.

- `--replicate-wild-do-table=db_name.tbl_name`

  Le dice al esclavo que restrinja la replicación a comandos donde cualquiera de las tablas actualizadas coincida con el patrón de base de datos y tabla. Los patrones pueden contener el comodín '`%`' and '`_`' , que tiene el mismo significado que para el operador `LIKE` . Para especificar más de una tabla, use esta opción múltiples veces, una para cada tabla. Esto funciona para actualizaciones entre bases de datos. Lea los comentarios a continuación de esta lista de opciones.

  Ejemplo: `--replicate-wild-do-table=foo%.bar%` replica sólo actualizaciones que usen una tabla donde el nombre de la base de datos comience con `foo` y el nombre de la tabla comienza con `bar`.

  Si el patrón del nombre de la tabla es `%`, coincide con cualquier nombre de tabla y la opción se aplica a commandos a nivel de base de datos (`CREATE DATABASE`, `DROP DATABASE`, y `ALTER DATABASE`). Por ejemplo, si usa `--replicate-wild-do-table=foo%.%`, comandos a nivel de base de datos se replican si el nombre de la base de datos coinciden con el patrón `foo%`.

  Para incluir caracteres comodín literales en el nombre de la base de datos o de tabla, debe introducir un carácter de antibarra de escape. Por ejemplo, para replicar todas las tablas de una base de datos que se llama `my_own%db`, pero no replicar tablas de la base de datos `my1ownAABCdb` , debe escapar los caracteres '`_`' y '`%`' así: `--replicate-wild-do-table=my\_own\%db`. Si usa la opción en la línea de comandos, puede necesitar doblar las antibarras o poner el valor de la opción entre comillas, dependiendo del intérprete de comandos. Por ejemplo, con el shell `bash` , tendría que escribir `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=db_name.tbl_name`

  Le dice al esclavo que no replique un comando donde cualquier tabla coincida con el patrón dado. Para especificar más de una tabla a ignorar, use esta opción múltiples veces, una para cada tabla. Esto funciona para actualizaciones entre bases de datos. Lea los comentarios a continuación de esta lista de opciones.

  Ejemplo: `--replicate-wild-ignore-table=foo%.bar%` no replica actualizaciones que use una tabla donde el nombre de la base de datos comience con `foo` y el nombre de tabla comience con `bar`.

  Para información acerca de cómo funcionan las coincidencias, consulte la descripción de la opción `--replicate-wild-do-table` . Las reglas para incluir caracteres comodín en la opción son las mismas que para `--replicate-wild-ignore-table` .

- `--replicate-rewrite-db=from_name->to_name`

  Le dice al esclavo que traduzca la base de datos por defecto (esto es, la seleccionada por `USE`) a *`to_name`* si era *`from_name`* en el maestro. Sólo los comandos que tengan que ver con tablas están afectados (no comandos tales como `CREATE DATABASE`, `DROP DATABASE`, y `ALTER DATABASE`), si y sólo si *`from_name`* era la base de datos por defecto en el maestro. Esto no funciona para actualizaciones entre bases de datos. Tenga en cuenta que la traducción del nombre de la base de datos se hace antes que se testeen las reglas de `--replicate-*`.

  Si usa esta opción en la línea de comandos y el carácter '>' es especial en su intérprete de comandos, ponga entre comillas el valor de la opción. Por ejemplo:

  ```
  shell> mysqld --replicate-rewrite-db="olddb->newdb"
  ```

- `--replicate-same-server-id`

  Para usar en esclavos. Usualmente puede usar el valor por defecto de 0, para evitar bucles infinitos en replicación circular. Si se pone a 1, este esclavo no evita eventos que tengan su propio id de servidor; normalmente esto es útil sólo en configuraciones raras. No puede ponerse a 1 si se usa `--log-slave-updates` . Tenga en cuenta que por defecto el flujo de entrada/salida del esclavo no escribe eventos en el log binario si tiene el id del esclavo (esta optimización ayuda a ahorrar espacio de disco). Así que si quiere usar `--replicate-same-server-id`, asegúrese de arrancar el esclavo con esta opción antes de hacer que el esclavo lea sus propios eventos que quiera que ejecute el flujo SQL del esclavo.

- `--report-host=slave_name`

  El nombre de máquina o número IP del esclavo que debe reportar al maestro durante el registro del esclavo. Este valor aparece en la salida de `SHOW SLAVE HOSTS` en el maestro. No ponga ningún valor si no quiere que el esclavo se registre él mismo en el maestro. Tenga en cuenta que no es suficiente para el maestro leer el número IP del esclavo en el socket TCP/IP una vez que el esclavo conecte. Debido a `NAT` y otros elementos de enrutamiento, esa IP puede no ser válida para conectar del esclavo al maestro y otros equipos.

- `--report-port=slave_port`

  El número de puerto TCP/IP para conectar al esclavo, a ser reportado al maestro durante el registro del esclavo. Asigne un valor sólo si el esclavo está escuchando en un puerto no estándar o si tiene un túnel especial desde el maestro u otros clientes al esclavo. Si no está seguro, deje esta opción sin asignar ningún valor.

- `--skip-slave-start`

  Le dice a un esclavo que no arranque el flujo del esclavo cuando el servidor arranque. Para arrancar el flujo posteriormente, use `START SLAVE` .

- `--slave_compressed_protocol={0|1}`

  Si se pone esta opción a 1, use compresión para el protocolo esclavo/maestro si ambos lo soportan.

- `--slave-load-tmpdir=file_name`

  Nombre del directorio en el que el esclavo crea ficheros temporales. Esta opción por defecto es el valor de la variable de sistema `tmpdir` . Cuando el flujo SQL del esclavo replica un comando `LOAD DATA INFILE`, extrae el fichero a ser cargado del log retardado en ficheros temporales, luego los carga en la tabla .Si el fichero cargado en el maestro es muy grande, los ficheros temporales en el esclavo también lo son. Por lo tanto, es una buena idea usar esta opción para decirle al esclavo que ponga los ficheros

temporales en un sistema de ficheros con mucho espacio disponible. En tal caso, puede usar la opción `--relay-log` en ese sistema de ficheros, debido a que los logs retardados también son muy grandes. `--slave-load-tmpdir` debe apuntar un sistema de ficheros en disco, no en memoria. El esclavo necesita los ficheros temporales usados para replicar `LOAD DATA INFILE` para sobrevivir a un reinicio de la máquina. El directorio no debe ser uno que limpie el sistema operativo durante el arranque.

- `--slave-net-timeout=`*`seconds`*

  El número de segundos a esperar para más datos del maestro antes de abortar la lectura, considerando la conexión rota y tratando de reconectar. El primer reintento se hace inmediatamente tras el timeout. El intervalo entre reintentos se controla mediante la opción `--master-connect-retry` .

- `--slave-skip-errors= [`*`err_code1,err_code2,... | all`*`]`

  Normalmente, la replicación para cuando ocurre un error, lo que le da la oportunidad de resolver la inconsistencia en los datos manualmente. Esta opción le dice al flujo SQL esclavo que continue la replicación cuando un comando retorna cualquiera de los errores listados en la opción.

  No use esta opción a no ser que entienda porqué esta obteniendo opciones. Si no hay bugs en la preparación de la replicación ni en los programas cliente, y no hay bugs en el propio MySQL, nunca ocurre un error que pare la replicación. El uso indiscriminado de esta opción provoca que el esclavo pueda quedar desincronizado con el maestro, sin que usted sepa porqué ha ocurrido.

  Para códigos de error, debe usar los números proporcionados por el mensaje de error en su log de errores del esclavo y en la salida de `SHOW SLAVE STATUS`. Los códigos de error del servidor se listan en Capítulo 26, *Manejo de errores en MySQL*.

  También puede (pero no debería) usar el valor no recomendable de `all` que ignora todos los mensajes de error y continúa funcionando sin tener en cuenta qué ocurre. No hace falta decir que, si lo usa, no podemos garantizar la integridad de los datos. Por favor no proteste (ni envíe reportes de bug) en este caso si los datos del esclavo no se parecen a lo que hay en el maestro. *Le hemos advertido*.

  Ejemplos:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

Las reglas `--replicate-*` se evalúan como se explica para determinar si un comando se ejecuta por el esclavo o se ignora:

1. ¿Hay algunas reglas `--replicate-do-db` o `--replicate-ignore-db` ?
   - *Sí*: Puede testearlas como `--binlog-do-db` y `--binlog-ignore-db` (consulte Sección 5.10.3, “El registro binario (Binary Log)”). ¿Cuál es el resultado del test?
     - Ignora el comando: Lo ignora y sale.
     - Ejecuta el comando: No lo ejecuta inmediatamente; difiere la decisión; continúa con el siguiente paso.
   - *No*: Continúa con el siguiente paso.
2. ¿Estamos ejecutando ahora una función o procedimiento almacenado?
   - *Sí*: Ejecuta la consulta y sale.
   - *No*: Continúa con el siguiente paso.

3. ¿Hay alguna regla `--replicate-*-table` ?
   - *No*: Continúa con el siguiente paso.
   - *Sí*: Continúa con el siguiente paso. Sólo las tablas que van a ser actualizadas se comparan con las reglas (`INSERT INTO sales SELECT * FROM prices`: sólo `sales` se compara con las reglas). Si varias tablas van a actualizarse (comando de múltiples tablas), la primera tabla coincidente (coincidencia “do” o “ignore”) gana. Esto es, la primera tabla se compara con las reglas. A continuación, si no se puede tomar ninguna decisión, la segunda tabla se compara con las reglas, y así.
4. ¿Hay algunas tablas `--replicate-do-table` ?
   - *Sí*: ¿Coincide la tabla con alguna de ellas?
     - *Sí*: Ejecuta la consulta y sale.
     - *No*: Sigue con el siguiente paso.
   - *No*: Sigue con el siguiente paso.
5. ¿Hay alguna regla `--replicate-ignore-table`?
   - *Sí*: ¿Coincide la tabla con alguna de ellas?
     - *Sí*: Ignora la consulta y sale.
     - *No*: Continúa con el siguiente paso.
   - *No*: Continúa con el siguiente paso.
6. ¿Hay alguna regla `--replicate-wild-do-table`?
   - *Sí*: ¿Coincide la tabla con alguna de ellas?
     - *Sí*: Ejecuta la consulta y sale.
     - *No*: Continúa con el siguiente paso.
   - *No*: Continúa con el siguiente paso.
7. ¿Hay alguna regla `--replicate-wild-ignore-table`?
   - *Sí*: ¿Coincide la tabla con alguna de ellas?
     - *Sí*: Ignora la consulta y sale.
     - *No*: Continúa con el siguiente paso.
   - *No*: Continúa con el siguiente paso.
8. No coincide ninguna regla `--replicate-*-table` . ¿Hay alguna otra tabla para comprobar con las reglas?
   - *Sí*: Bucle.
   - *No*: Hemos testeados todas las tablas a actualizar y no podemos encontrar ninguna coincidencia con ninguna regla. ¿Hay alguna regla `--replicate-do-table` o `--replicate-wild-do-table` ?

- *Sí*: Ignora la consulta y sale.
- *No*: Ejecuta la consulta y sale.

6.9. Preguntas y respuestas sobre replicación

P: ¿Cómo configuro el esclavo si el maestro está en ejecución y no quiero pararlo?

R: Hay varias opciones. Si ha hecho una copia de seguridad del maestro en algún punto y ha guardado el nombre del log binario y el desplazamiento (de la salida de `SHOW MASTER STATUS`) correspondiente a la muestra de datos, use el siguiente procedimiento:

1. Asegúrese que el esclavo tiene asignado un ID de servidor único.
2. Ejecute el siguiente comando en el esclavo, rellenando los valores apropiados en cada opción:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='master_user_name',
    ->     MASTER_PASSWORD='master_pass',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

3. Ejecute `START SLAVE` en el esclavo.

Si no tiene una copia de seguridad del maestro, aquí hay un procedimiento rápido para crear uno. Todos los pasos deben realizarse en el maestro.

1. Ejecute este comando:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

2. Con el bloqueo todavía activo, ejecute este comando (o una variación de él):

```
shell> tar zcf /tmp/backup.tar.gz /var/lib/mysql
```

3. Ejecute este comando y asegúrese de guardar la salida, que necesitará posteriormente:

```
mysql> SHOW MASTER STATUS;
```

4. Libere el bloqueo:

```
mysql> UNLOCK TABLES;
```

Una alternativa es hacer un volcado SQL del maestro en lugar de una copia binaria como en el procedimiento precedente. Para hacerlo, puede usar `mysqldump --master-data` en su maestro y cargar el volcado SQL posteriormetne en su esclavo. Sin embargo, esto es más lento que hacer una copia binaria.

No importa cuál de los dos métodos use, luego siga las instrucciones para el caso en que tenga una muestra de los datos y haya guardado el nombre del log y el desplazamiento. Puede usar la misma muestra de datos para preparar varios esclavos. Una vez que tenga la muesta de datos del maestro, puede esperar a preparar un esclavo mientras los logs binarios en el maestro queden intactos. Hay dos limitaciones prácticas en la cantidad de tiempo que puede esperar que son la cantidad de espacio disponible para guardar los logs binarios y el tiempo que tarda el esclavo en leerlos.

También puede usar `LOAD DATA FROM MASTER`. Este es un comando apropiado que transfiere una muestra de los datos al esclavo y ajusta el nombre del log y el desplazamiento a la vez. En un futuro `LOAD DATA FROM MASTER` será el método recomendado para preparar un esclavo. Tenga en cuenta, sin embargo, que sólo funciona para tablas `MyISAM` y que puede mantener un bloqueo de lectura durante un largo periodo de tiempo. Todavía no está implementado tan eficientemente como querríamos. Si tiene tablas muy grandes, el mejor método por ahora es tomar una muesta de datos binaria en el maestro tras ejecutar `FLUSH TABLES WITH READ LOCK`.

P: ¿Necesita el esclavo estar conectado todo el tiempo al maestro?

R: No, no hace falta. El esclavo puede caer o quedar desconectado durante horas o días, luego reconectar y leer las actualizaciones. Por ejemplo, puede preparar una relación maestro/servidor mediante un enlace telefónico que sólo esté disponible esporádicamente y durante cortos periodos de tiempo. La implicación de esto es, que en un momento dado, el esclavo no garantiza estar sincronizado con el maestro a no ser que tome medidas especiales. En el futuro, tendremos la opción de bloquear el maestro hasta que al menos un esclavo esté sincronizado.

P: ¿Cómo se qué retardo tiene esclavo comparado con el maestro? En otras palabras, ¿cómo se la fecha de la última consulta replicada por el esclavo?

R: Para un esclavo MySQL 5.0, puede leer la columna `Seconds_Behind_Master` en `SHOW SLAVE STATUS`. Consulte Sección 6.3, “Detalles de la implementación de la replicación”.

Cuando un flujo SQL esclavo ejecuta un evento leído del maestro, modifica su propia hora a la hora del evento (esto es el porqué `TIMESTAMP` se replica bien). En la columna `Time` en la salida de `SHOW PROCESSLIST`, el número de segundos mostrados por el flujo SQL del esclavo es el número de segundos entre la hora del último evento replicado y la hora real de la máquina esclava. Puede usar esto para determinar la fecha del último evento replicado. Tenga en cuenta que si su esclavo se ha desconectado del maestro durante una hora, y luego reconecta, puede ver de forma inmediata valores `Time` como 3600 para el flujo SQL esclavo en `SHOW PROCESSLIST`. Esto ocurriría debido a que el esclavo está ejecutando comandos que tienen una hora.

P: ¿Cómo fuerzo al maestro a bloquear actualizaciones hasta que el esclavo las lee?

R: Use el siguiente procedimiento:

1. En el maestro, ejecute estos 2 comandos:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SHOW MASTER STATUS;
   ```

   Registra el nombre y desplazamiento del log de la salida del comando `SHOW` . Estas són las coordinaciones de replicación.

2. En el esclavo, ejecute el siguiente comando, donde el argumento de la función `MASTER_POS_WAIT()` son las coordenadas de replicación obtenidos en el prévio paso:

   ```
   mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
   ```

   El comando `SELECT` bloquea hasta que el esclavo alcanza el fichero y desplazamiento de log especificado. En este punto, el esclavo está sincronizado con el maestro y el comando retorna.

3. En el maestro, ejectue el siguiente comando para permitir al maestro comenzar a ejecutar actualizaciones otra vez:

   ```
   ```

```
mysql> UNLOCK TABLES;
```

P: ¿Qué cuestiones debo considerar al preparar una replicación bidireccional?

R: La replicación MySQL actualmente no sopota ningún protocolo de bloqueo entre maestro y servidor para garantizar la atomicidad de una actualización distribuida (entre servidores). En otras palabras, es posible para el cliente A hacer una actualización del co-maestro 1, y mientras tanto, antes de propagar al co-maestro 2, el cliente B puede hacer una actualización en el co-maesto 2 que haga que la actualización del cliente A funcione de forma distinta que haría en el co-maestro 1. Por lo tanto, cuando la actualización del cliente A se hace en el co-maestro 2, produce tablas que son distintas que las que tiene en el co-maestro 1, incluso tras todas las actualizaciones del co-maestro 2 se hayan propagado. Esto significa que no debe encadenar dos servidores en una relación de replicación bidireccional a no ser que esté seguro que sus actualizaciones pueden hacerse en cualquier orden, o a no ser que se encarge de actualizaciones desordenadas de algún modo en el código del cliente.

Debe tener en cuenta que replicación bidireccional no mejora mucho el rendimiento (o nada), en lo que se refiere a actualizaciones. Ambos servidores tienen que hacer el mismo número de actualizaciones, lo mismo que haría un servidor. La única diferencia es que hay un poco menos de bloqueos, ya que las actualizaciones originadas en otro servidor se serializan en un flojo esclavo. Incluso este beneficio puede verse penalizado por retardos de red.

P: ¿Cómo puedo usar replicación para mejorar rendimiento en mi sistema?

R: Debe preparar un servidor como maestro y dirigir todas las escrituras al mismo. Luego configure tantos esclavos como quiera, y distribuya las lecturas entre los esclavos y maestro. También puede arrancar esclavos con las opciones `--skip-innodb`, `--skip-bdb`, `--low-priority-updates`, y `--delay-key-write=ALL` para mejorar la velocidad en el esclavo. En este caso, el esclavo usa tablas no transaccionales `MyISAM` en lugar de `InnoDB` y `BDB` para obtener más velocidad.

P: ¿Qué debo hacer para preparar código cliente en mis propias aplicaciones para usar replicación que mejore el rendimiento?

R: Si la parte de su código que es responsable de acceso a bases de datos se ha modularizado correctamente, convertiéndola para correr con un entorno de replicación debe ser algo sencillo. Cambie la implementación de su acceso a base de datos para enviar todas las escrituras al maestro, y enviar lecturas al maestro o al esclavo. Si su código no tiene este nivel de abstacción, preparar un sistema de replicación le da la oportunidad de limpiarlo. Debe comenzar creando una biblioteca o módulo con las siguientes funciones:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` en cada nombre de función significa que la función se encarga de tratar todas las condiciones de error. Puede usar distintos nombres para las funciones. Lo importante es tener una interfaz unificada para conectar para lecturas, conectar para escrituras, hacer una lectura y hacer una escritura.

Luego debe convertir su código de cliente para usar la biblioteca. Este es un proceso largo y complicado en principio, pero vale la pena a la larga. Todas las aplicaciones que usen la aproximación descrita son capaces de aprovechar al configuración maestro/esclavo, incluso uno en el que intervengan varios esclavos. El código es mucho más fácil de mantener, y añadiendo opciones para tratar problemas es trivial. Necesita modificar una o dos funciones sólo; por ejemplo, para loguear cuánto tarda cada comando, o qué comando provoca un error.

Si ha escrito mucho código, puede querer automatizar la tarea de conversión usando la utilidad `replace` que viene con la distribución estándar MySQL, o escribir su propio script de conversión. Idealmente, su código usa convenciones de estilo. Si no, probablemente es mejor reescribirlo, o al menos regularizarlo manualmente para usar un estilo consistente.

P: ¿Cuándo y cómo puede mejorar la replicación MySQL el rendimiento de mi sistema?

R: La replicación MySQL es más benéfica para sistemas con lecturas frecuentes y escrituras infrecuentes. En teoría, usando una inicialización un maestro/múltiples esclavos, puede escalar el sistema añadiendo más esclavos hasta que se queda sin ancho de banda, o la carga de actualización crece hasta el punto que el maestro no puede tratarla.

Para determinar cuántos esclavos puede tener antes que los beneficios empiecen a notarse, necesita conocer el patrón de consultas, y determinar empíricamente con pruebas la relación entre las lecturas (lecturas por segundo , o `max_reads`) y las escrituras (`max_writes`) en un típico maestro y típico esclavo. El ejemplo muestra un cálculo simplificado de qué puede obtener con replicación para un sistema hipotético.

Digamos que la carga del sistema consiste en 10% de escrituras y 90% de lecturas, y hemos determinado con pruebas que `max_reads` es 1200 - 2 * `max_writes`. En otras palabras, el sistema puede hacer 1,200 lecturas por segundo sin escrituras, la escritura media es el doble de lenta que la lectura media, y la relación es linear. Supongamos que el maestro y cada esclavo tienen la misma capacidad, y que tienen un maestro y *N* esclavos. Tenemos para cada servidor (maestro o esclavo):

```
lecturas = 1200 - 2 * escrituras
```

`lecturas = 9 * escrituras / (N + 1)` (las lecturas se dividen, pero las escrituras van a todos los servidores)

```
9 * escrituras / (N + 1) + 2 * escrituras = 1200
```

```
escrituras = 1200 / (2 + 9/(N+1))
```

La última ecuación indica que el número máximo de escrituras para *N* esclavos, dado el ratio máximo de lecturas de 1,200 por minuto y un rato de nueve lecturas por escritura.

Este análisis lleva a las siguientes conclusiones:

- Si *N* = 0 (que significa que no tenemos replicación), nuestro sistema puede tratar unas 1200/11 = 109 escrituras por segundo.
- Si *N* = 1, tenemos 184 escrituras por segundo.
- Si *N* = 8,tenemos 400 escrituras por segundo.
- Si *N* = 17, tenemos 480 escrituras por segundo.
- Eventualmente, mientras *N* se aproxima al infinito (y el presupuesto a infinito negativo), podemos llegar carca de 600 escrituras por segundo, incrementando el rendimiento del sistema 5.5 veces. Sin embargo, con sólo ocho servidores, lo incrementamos cerca de cuatro veces.

Tenga en cuenta que estos cálculos asumen ancho de banda infinito y no tiene en cuenta muchos otros factores que pueden ser significativos en el sistema. En muchos casos, puede no ser capaz de realizar una computación similar al mostrado que prediga que ocurre en su sistema si añade *N* esclavos de replicación. Sin embargo, responder las siguientes cuestiones deben ayudarle a decidir si y cuánto la replicación mejorará el rendimiento de su sistema:

- ¿Cuál es el ratio de lectura/escritura en su sistema?

- ¿Cúanta carga de escritura puede tratar un servidor si reduce las lecturas?
- ¿Para cuántos esclavos tiene ancho de banda disponible?

P: ¿Cómo puedo usar replicación para proporcionar redundancia/alta disponibilidad?

R: Con las características actuales, puede preparar un maestro y un esclavo (o varios esclavos), y escribir un scrip que monitorice el maestro para ver si está en marcha. Luego enseñe a su aplicación y a los esclavos a cambiar el maestro en caso de fallo. Algunas sugerencias:

- Para decir al esclavo que cambie el maestro, use el comando `CHANGE MASTER TO` .
- Una buena forma de mantener a las aplicaciones informadas de la localización del maestro es con una entrada de DNS dinámica para el maestro. Con `bind` puede usar `nsupdate` para actualizar dinámicamente el DNS.
- Debe ejecutar los esclavos con la opción `--log-bin` y sin `--log-slave-updates`. De este modo, el esclavo está preparado para ser un maestro en cuanto ejecute `STOP SLAVE`; `RESET MASTER`, y `CHANGE MASTER TO` en los otros esclavos. Por ejemplo, asuma que tiene la siguiente configuración:

```
      WC
        \
         v
WC----> M
       / | \
      /  |  \
     v   v   v
    S1   S2  S3
```

M es el maestro, **S** los esclavos, **WC** los clientes realizando escrituras y lecturas; los clientes que ejecutan sólo lecturas de bases de datos no se representan , ya que no necesitan cambiar. **S1**, **S2**, y **S3** son esclavos ejecutándose con `--log-bin` y sin `--log-slave-updates`. Como las actualizaciones recibidas por un esclavo del maestro no se loguean en el log binario a no ser que se especifique `--log-slave-updates`, el log binario en cada esclavo está vacío. Si por alguna razón **M** no está disponible puede hacer que uno de los esclavos sea el nuevo maestro. Por ejemplo, si elige **S1**, todos los **WC** deben redirigirse a **S1**, y **S2** y **S3** deben replicar de **S1**.

Asegúrese que todos los esclavos han procesado cualquier comando en su log retardado. En cada esclavo, ejecute `STOP SLAVE IO_THREAD`, luego chequee la salida de `SHOW PROCESSLIST` hasta que vea `Has read all relay log`. Mientras que esto es cierto para todos los esclavos, pueden reconfigurarse para una nueva configuración. En el esclavo **S1** se promociona a ser el maestro, ejecute `STOP SLAVE` y `RESET MASTER`.

En los otros esclavos **S2** y **S3**, use `STOP SLAVE` y `CHANGE MASTER TO MASTER_HOST='S1'` (donde `'S1'` representa el nombre de equipo de **S1**). Para `CHANGE MASTER`, añada toda la información acerca de cómo conectar a **S1** desde **S2** o **S3** (*`user`*, *`password`*, *`port`*). En `CHANGE MASTER`, no es necesario especificar el nombre del log binario de **S1** o la posición del log desde la que hay que leer: Sabemos que es el primer log binario y la posición 4, que son los defectos para `CHANGE MASTER`. Finalmente, use `START SLAVE` en **S2** y **S3**.

Luego enseñe a todos los **WC** a dirigir sus comandos a **S1**. Desde ese punto, todos los comandos de actualización enviados de **WC** a **S1** se escriben en el log binario de **S1**, que contiene cada comando de actualización enviado **S1** desde que **M** murió.

El resultado es esta configuración:

```
      WC
     /
     |
WC   |  M(unavailable)
 \   |
  \  |
   v v
    S1<--S2  S3
     ^       |
     +-------+
```

Cuando **M** está activo otra vez, debe ejecutar en la misma máquina `CHANGE MASTER` como el ejecutado en **S2** y **S3**, así que **M** llega a ser un esclavo de **S1** y recoge todos las escrituras de **WC** que se ha perdido mientras no estaba activo. Para hacer que **M** sea un maestro de nuevo (debido a que es la máquina más rápida, por ejemplo), use el anterior protocolo como si **S1** no estuviera disponible y **M** fuera a ser el nuevo maestro. Durante este procedimiento, no olvide ejecutar `RESET MASTER` en **M** antes de hacer **S1**, **S2**, y **S3** esclavos de **M**. De otro modo, ellos podrían recober antiguas escrituras de **WC** desde el punto en que **M** dejó de estar disponible.

Estamos trabajando en integrar un sistema de elección de maestro automático en MySQL, pero hasta que esté preparado, debe crear sus propias herramientas de monitoreo.

6.10. Resolución de problemas de replicación

Si ha seguido las instrucciones, y su configuración de replicación no funciona, chequee lo siguiente:

- **Chequee el log de errores en busca de mensajes**. Muchos usuarios han perdido tiempo por no hacer esto lo primero tras encontrar problemas.
- ¿Está logueando el maestro en el log binario? Chequee con `SHOW MASTER STATUS`. Si lo está, `Position` no es cero. Si no, verifique que está ejecutando el maestro con las opciones `log-bin` y `server-id` .
- ¿Está corriendo el esclavo? Use `SHOW SLAVE STATUS` para chequear si los valores `Slave_IO_Running` y `Slave_SQL_Running` son `Yes`. Si no , verfique las opciones que se usaron para arrancar el esclavo.
- Si el esclavo está corriendo, ¿estableció una conexión con el maestro? Use `SHOW PROCESSLIST`, encuentre los flujos de entrada/salida y SQL y chequee su columna `State` para ver qué muestran .Consulte Sección 6.3, “Detalles de la implementación de la replicación”. Si el estado flujo de entrada/salida dice `Connecting to master`, verifique los permisos para los usuarios de replicación en el maestro, nombre de equeipo, la configuración del DNS, si el maestro está corriendo y si se puede acceder desde el esclavo.
- Si el esclavo estaba corriendo préviamente pero ha parado, la razón habitual es que algunos comandos que se han ejectuado en el maestro han fallado en el esclavo. Esto nunca debe ocurrir si ha tomado muestras de datos apropiadas del maestro, y no ha modificado los datos en el esclavo fuera del flujo del esclavo. Si lo ha hecho, es un bug o ha encontrado una de las limitaciones de replicación descritas en Sección 6.7, “Características de la replicación y problemas conocidos”. Si es un bug, consulte Sección 6.11, “Reportar bugs de replicación” para instrucciones de cómo reportarlo.
- Si un comando que ha tenido exito en el maestro no se ejecuta en el esclavo, y no es posible resincronizar la base de datos completa (esto es, borrar la base de datos del esclavo y copiar una nueva muestra del maestro), pruebe lo siguiente:

  1. Determine si la tabla del esclavo es distinta a la del maestro. Trate de entender cómo ha ocurrido. Luego haga que la tabla del esclavo sea idéntica a la del maestro y ejecute `START SLAVE`.

2. Si el paso precedente no funciona o no se aplica, trate de entender si sería seguro hacer una actualización manual (si es necesario) y luego ignorar el siguiente comando del maestro.

3. Si decide que puede evitar el siguiente comando del maestro, ejecute los siguientes comandos:

```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n;
mysql> START SLAVE;
```

   El valor de *n* debe ser 1 si el siguiente comando del maestro no usa `AUTO_INCREMENT` o `LAST_INSERT_ID()`. De otro modo, el valor debe ser 2. La razón para usar un valor de 2 para comandos que usen `AUTO_INCREMENT` or¡ `LAST_INSERT_ID()` es que toman dos eventos en el log binario del maestro.

4. Si está seguro que el esclavo arrancó perfectamente sincronizado con el maestro, y que nadie ha actualizado las tablas involucradas fuera del flujo esclavo, entonces presumiblemente la discrepancia es producto de un bug. Si está ejecutando la versión más reciente, reporte el problema. Si está ejecutando una versión antigua de MySQL, trate de actualizar a la última versión.

6.11. Reportar bugs de replicación

Cuando determine que no hay un error de usuario involucrado, y la replicación todavía no funciona o es inestable, es el momento de enviarnos un reporte de bug. Necesitamos obtener toda la infomación posible que nos pueda proporcionar. Por favor dedique algo de tiempo y esfuerzo para preparar un buen reporte de bug.

Si tiene un caso de test repetible que demuestra el bug, por favor introdúzcalo en nuestra base de datos de bugs en http://bugs.mysql.com/. Si tiene un problema “fantasma” (uno que no pueda duplicar a voluntad), use el siguiente procedimiento:

1. Verifique que no hay ningún error de usuario involucrado. Por ejemplo, si actualiza el esclavo fuera de un flujo esclavo, los datos se desincronizan, y puede tener violaciones de claves únicas en las actualizaciones. En este caso, el flujo esclavo para y espera a que limpie las tablas manualmente para sincronizar. *Este no es un problema de replicación. Es un problema de interferencias externas que provocan problemas de replicación.*

2. Ejecute el esclavo con las opciones `--log-slave-updates` y `--log-bin` . Estas opciones provocan que el esclavo loguee las actualizaciones que recibe del maestro en sus propios logs binarios.

3. Guarde toda evidencia antes de resetear el estado de replicación. Si no tenemos información o sólo parcial, es difícil o imposible para nosotros encontrar el problema. la evidencia que debe recolectar es:

   - Todos los logs binarios del maestro
   - Todos los logs binarios del esclavo
   - La salida de `SHOW MASTER STATUS` del maestro cuando descubrió el problema
   - La salida de `SHOW SLAVE STATUS` del maestro cuando descubrió el problema
   - Logs de error del maestro y esclavo

4. Use `mysqlbinlog` para examinar los logs binarios. Lo siguiente debería ser útil para encontrar la consulta problemática, por ejemplo:

```
```

```
shell> mysqlbinlog -j pos_from_slave_status \
           /path/to/log_from_slave_status | head
```

Una vez que ha recolectado la evidencia del problema, trate de isolarlo como un caso de test separado. Luego introduzca el problema en nuestra base de datos de bugs en http://bugs.mysql.com/ que es tanta información como sea posible.

Capítulo 7. Optimización de MySQL

Tabla de contenidos

La optimización es una tarea compleja, porque requiere un conocimiento de todo el sistema a optimizar. Se podría optimizar sólo algunos aspectos teniendo poco conocimiento del sistema o aplicación, pero cuanto más óptimo se quiera el sistema, más se tiene que conocer acerca del mismo.

Este capítulo intenta explicar y da algunos ejemplos de las diferentes maneras de optimizar MySQL. Sin embargo se debe recordar que siempre existen vías de hacer todavía más rápido el sistema, aunque pudieran requerir un notable incremento de esfuerzos.

7.1. Panorámica sobre optimización

El factor más importante para hacer un sistema rápido es su diseño básico. Ademas, es necesario conocer los procesos que cumple el sistema y cuáles son sus cuellos de botella. En la mayoría de casos los cuellos de botella nacen de los siguientes factores:

- Búsqueda en Disco. El disco necesita cierto tiempo para encontrar un paquete de datos. Con discos modernos, el tiempo medio para esto es usualmente menor a 10ms, así que en teoría se pueden hacer 100 búsquedas por segundo. Este tiempo mejora lentamente con los discos nuevos y es muy difícil optimizarlo para una sola tabla. La manera de optimizar el tiempo de búsqueda es distribuir los datos dentro de más de un disco.

- Lectura y escritura en disco. Cuando el disco se encuentra en la posición correcta, necesitamos leer los datos. Los discos modernos transfieren al menos 10-20MB/s. Esto es mucho más fácil de optimizar que las búsquedas, puesto que podemos leer en paralelo desde múltiples discos.

- Ciclos de CPU. Cuando tenemos datos en la memoria principal, necesitamos procesarlos para obtener algún resultado. Tener tablas pequeñas en comparación con la cantidad de memoria es el factor mas común de limitación. Pero con tablas pequeñas, la rapidez no es usualmente el problema.

- Ancho de banda de Memoria. Cuando el CPU necesita más datos de los que puede almacenar en la cache de la CPU, el ancho principal de la memoria se convierte en un cuello de botella. Es poco común en la mayoría de casos, pero debe tenerse en cuenta.

7.1.1. Limitaciones y soluciones de compromiso en el diseño de MySQL

Al utilizar el motor de almacenamiento `MyISAM`, MySQL usa un bloqueo (lock) extremadamente rápido de tablas, que permite múltiples lecturas o una sola escritura. El mayor problema con este motor de almacenamiento ocurre cuando se tiene un flujo constante de actualizaciones y selecciones lentas de una sola tabla. Si éste es el problema para algunas tablas, puede usar otro motor de almacenamiento para ellas. Ver Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

MySQL puede trabajar con tablas transaccionales y no transaccionales. Para hacer el trabajo más facil con tablas no transaccionales (donde no se puede deshacer una transacción si algo va mal), MySQL tiene las siguientes reglas. Obsérvese que estas reglas **sólo** se aplican cuando no se está corriendo en modo SQL estricto o si se usa el especificador `IGNORE` para un `INSERT` o un `UPDATE`.

- Todas las columnas tienen valores por defecto. Obsérvese que cuando se ejecuta en modo SQL estricto (incluyendo modo SQL `TRADITIONAL`), se debe especificar cualquier valor para una columna `NOT NULL`.

- Si se inserta un valor inapropiado o fuera de rango dentro de una columna, MySQL atribuye a la columna el “ mejor valor posible ” en vez de reportar un error. Para valores numéricos, éste es el 0, el valor más pequeño posible o el valor más grande posible. Para cadenas de texto, puede ser una cadena vacía o el trozo de la cadena más grande que quepa en la columna. Este comportamiento no se aplica cuando se ejecuta en modo SQL estricto `TRADITIONAL`.

- Todas las expresiones calculadas retornan un valor que puede ser usado en vez de señalar una condición de error. Por ejemplo, 1/0 devuelve `NULL`. (Este comportamiento puede ser cambiado usando el modo SQL `ERROR_FOR_DIVISION_BY_ZERO`).

Si se usa una tabla no transaccional, no debería usar MySQL para comprobar el contenido de la columna. En general, la manera más segura (y generalmente más rápida) es usar la aplicacion para asegurarse que se están pasando sólo valores legales a la base de datos.

Para más informacion acerca de esto, ver Sección 1.7.6, “Cómo trata MySQL las restricciones (Constraints)” y Sección 13.2.4, “Sintaxis de `INSERT`” o Sección 5.3.2, “El modo SQL del servidor”.

7.1.2. Diseñar aplicaciones pensando en la portabilidad

Debido a que todos los servidores SQL implementan diferentes partes del estándar SQL, toma trabajo escribir aplicaciones SQL portables. Es muy fácil obtener portabilidad para selects muy simples y para inserts, pero es más difícil cuantas más funcionalidades se requieran. Obtener una aplicación que sea rápida en varios sistemas de bases de datos, es una tarea muy difícil.

Para hacer aplicaciones complejas portables, necesita determinar para cuáles servidores SQL debe trabajar, después determinar qué características soportan esos servidores.

Todos los sistemas de bases de datos tienen algunos puntos débiles. Esto es, tienen diferentes concesiones de diseño que conducen a comportamientos diferentes.

Puede usar el programa de MySQL `crash-me` para encontrar funciones, tipos y límites que puede usar con una selección de servidores de bases de datos. `crash-me` no comprueba cada posible característica, pero es razonablemente completo, puesto que hace cerca de 450 pruebas.

Un ejemplo de un tipo de información de la que el programa `crash-me` puede proveer es que no debería usar nombres de columnas que sean mayores de 18 caracteres si quiere que la aplicación funcione en Informix o DB2.

El programa `crash-me` y MySQL benchmarks son independientes de la base de datos. Mirando por encima cómo están escritos, puede hacerse una idea de qué debe hacer que sus propias aplicaciones sean independientes de bases de datos. Los programas se encuentran en el directorio `sql-bench` dentro del código fuente de la distribución de MySQL. Estan escritos en Perl y usan la interfaz para bases de datos DBI. Usar DBI ya soluciona una parte de los problemas de portabilidad puesto que provee de métodos de acceso independientes a las bases de datos.

Ver http://dev.mysql.com/tech-resources/benchmarks/ para los resultados de las mediciones.

Esforzarse para ser independiente del motor de bases de datos, implica prestar atención a los cuellos de botella de cada servidor SQL. Por ejemplo, MySQL es muy rápido obteniendo y actualizando registros para tablas `MyISAM`, pero tiene un problema mezclando lecturas y escrituras lentas sobre la misma tabla. Por otra parte, Oracle tiene un enorme problema cuando intenta acceder a registros que se han actualizado recientemente (hasta que se escriben en el disco). Las bases de datos transaccionales en general no son muy buenas generando resúmenes de tablas desde las tablas de registro (log), puesto que en este caso el bloqueo (lock) de registros es casi inútil.

Para hacer una aplicación *realmente* independiente de la base de datos, se necesita definir una interfaz fácilmente extendible a través de la cual se manipularán los datos. Puesto que C++ está disponible en la mayoría de los sistemas, tiene sentido usar una interfaz basada en clases de C++ hacia la base de datos.

Si usa alguna característica que es específica de algún sistema de base de datos (por ejemplo la sentencia `REPLACE`, que es específica en MySQL), debería implementar la misma característica para

otros servidores SQL codificando un método alternativo. Aunque la alternativa sea más lenta, permite que la misma tarea se haga en otros servidores.

Con MySQL, puede usar la sintaxis `/*! */` para agregar algunas palabras claves de MySQL a una consulta. El código dentro de `/**/` es tratado como un comentario (e ignorado) por la mayoría de los otros servidores SQL.

Si el alto rendimiento es más importante que la exactitud, como en algunas aplicaciones Web, es posible crear una capa de la aplicación que almacene en una cache todos los resultados para obtener un rendimiento mejor. Dejando que los resultados viejos expiren en determinado tiempo, puede mantener la cache razonablemente actualizado. Éste es un método para soportar picos de carga, al que se añade la posibilidad de implementar un incremento dinámico de la cache y un aumento del tiempo de expiración, hasta que las cosas regresen a la normalidad.

En este caso, la información de creación de la tabla debe contener información del tamaño inicial de la cache y de la frecuencia de refresco de la tabla.

Una alternativa para implementar una aplicación en cache es usar la cache para consultas de MySQL. Habilitando la cache para consultas, el servidor toma los detalles de las consultas determinando qué resultados pueden ser reutilizados. Esto simplifica la aplicación. Ver Sección 5.12, “La caché de consultas de MySQL”.

7.1.3. Para qué hemos usado MySQL

Esta sección describe un uso inicial de MySQL.

Durante el inicio del desarrollo de MySQL, las características de MySQL fueron pensadas para ajustarse a nuestro cliente más grande, que hacía data-warehouse para un par de los mayoristas más grandes de Suecia.

Desde todas las tiendas obteníamos resúmenes semanales de todas las transacciones con tarjetas de bonos, y se esperaba de nosotros información útil para que los dueños de las tiendas entendieran cómo sus campañas publicitarias afectaban a sus propios clientes.

El volumen de datos era enorme (cerca de siete millones de transacciones resumidas por mes), y teníamos datos de 4-10 años que debíamos presentar a los usuarios. Teníamos peticiones semanales de nuestros clientes, quienes querían acceso instantáneo a los nuevos reportes de esta información.

Solucionamos este problema almacenando toda la información por mes en unas “tablas de transacciones” comprimidas. Teníamos una serie de sencillas macros que generaban resúmenes de tablas agrupados por diferentes criterios (grupo de productos, id del cliente, tienda, etc) a partir de las tablas en las que fueron almacenadas las transacciones. Los reportes eran páginas Web que eran generadas dinámicamente por un pequeño script escrito en Perl. Este script analizaba la página Web, ejecutaba una sentencia SQL y escribía los resultados. Debimos usar PHP o `mod_perl` en vez de eso, pero no estaban disponibles en aquel entonces.

Para datos gráficos, escribimos una sencilla herramienta en C que podía procesar los resultados de una consulta SQL y generar una imagen GIF basada en los resultados. Esta herramienta también era dinámicamente invocada desde el script en Perl que analizaba las páginas Web.

En la mayoría de casos, se podía crear un nuevo reporte simplemente copiando un script existente y modificando la consulta SQL que utilizaba. En algunos casos, necesitábamos agregar más columnas a una tabla de resumen existente o generar una nueva. Esto también fue sencillo puesto que guardábamos todas las tablas con las transacciones en disco. (El tamaño era de alrededor de 50GB de tablas de transacciones y 200GB de otros datos del cliente.)

También permitimos a nuestros clientes acceder a las tablas de resumen directamente con ODBC, para que los usuarios avanzados pudieran experimentar con los datos por sí mismos.

Este sistema trabajó bien y no tuvimos problemas elaborando los datos en un modesto servidor Sun Ultra SPARCstation (2x200Mhx). Finalmente el sistema fue migrado a Linux.

7.1.4. El paquete de pruebas de rendimiento (benchmarks) de MySQL

Esta sección debería contener una descripción técnica del paquete de pruebas de rendimiento de MySQL (así como del `crash-me`), pero esa descripción aún no ha sido escrita. Sin embargo, puede hacerse una buena idea de cómo hacer pruebas de rendimiento viendo el código y los resultados dentro del directorio `sql-bench` en el código fuente de la distribución de MySQL.

La finalidad de este paquete de pruebas de rendimiento es visualizar qué operaciones se realizan bien y cuáles lo hacen pobremente en cada implementación de SQL.

Estas pruebas de rendimiento no son multi hilo, así que miden el tiempo mínimo para las operaciones realizadas. Se planea agregar en un futuro pruebas multi hilo al paquete.

Para usar el paquete, deben satisfacerse los siguientes requisitos:

- El paquete de pruebas de rendimiento se proporciona con el código fuente de la distribución de MySQL. También puede descargar una distribución liberada de http://dev.mysql.com/downloads/, o usar nuestro repositorio de código fuente(ver Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”).
- Los scripts de las pruebas de rendimiento están escritos en Perl y usan el módulo de Perl DBI para acceder a los servidores de bases de datos, así que DBI debe estar instalado. También es necesario el controlador DBI específico para cada servidor al que se quiere realizar las pruebas. Por ejemplo, para probar MySQL, PostgreSQL, y DB2, debe tener los módulos `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` instalados. Ver Sección 2.13, “Notas sobre la instalación de Perl”.

Una vez obtenido el código fuente de la distribución de MySQL, el paquete de pruebas de rendimiento se encuentra en el directorio `sql-bench`. Para ejecutar las pruebas de rendimiento, compílese MySQL, váyase al directorio `sql-bench` y ejecútese el script `run-all-tests`:

```
shell> cd sql-bench
shell> perl run-all-tests --server=nombre_servidor
```

`nombre_servidor` debe ser uno de los servidores soportados. Para obtener la lista completa de opciones y servidores soportados, invóquese el comando:

```
shell> perl run-all-tests --help
```

El script `crash-me` también está situado dentro del directorio `sql-bench`. `crash-me` intenta determinar qué características soporta una base de datos y cuáles son sus capacidades y limitaciones. Esto lo consigue ejecutando consultas. Determina por ejemplo:

- Cuáles tipos de columnas se soportan
- Cuántos índicies se soportan
- Qué funciones se soportan
- Qué tamaño puede alcanzar una consulta
- Que tamaño puede alcanzar una columna `VARCHAR`

Para más información acerca de resultados de pruebas de rendimiento, visítese http://dev.mysql.com/tech-resources/benchmarks/.

7.1.5. Usar pruebas de rendimiento (benchmarks) propios

Decididamente debería realizar pruebas de rendimiento sobre su aplicación y su base de datos para determinar dónde se encuentran los cuellos de botella. Eliminando un cuello de botella (o reemplazándolo con un módulo “tonto”) puede fácilmente identificar el siguiente cuello de botella. Aunque el rendimiento global de su aplicación sea aceptable, debería al menos hacer un plan para cada cuello de botella, y decidir cómo eliminarlo si algún día realmente necesita un rendimiento extra.

Para ejemplos de programas de pruebas de rendimiento portables, mire el paquete de pruebas de rendimiento MySQL. Ver Sección 7.1.4, “El paquete de pruebas de rendimiento (benchmarks) de MySQL”. Puede tomar cualquier programa de este paquete y modificarlo en base a sus necesidades. Así puede probar diferentes soluciones para su problema y probar cuál realmente es más rápida para usted.

Otro paquete de pruebas de rendimiento es Open Source Database Benchmark, disponible en http://osdb.sourceforge.net/.

Es muy común que un problema ocurra sólo cuando el sistema está bajo mucha carga. Muchos de nuestros clientes nos contactan cuando tienen un sistema (probado) en producción y encuentran problemas de carga. En la mayoría de casos, los problemas de rendimiento se deben a cuestiones relacionadas con el diseño básico de la base de datos (por ejemplo, lecturas completas de las tablas no son buenas bajo alta carga) o problemas con el sistema operativo o las bibliotecas. En la mayoría de casos, esos problemas se solucionarían mucho más fácilmente si los sistemas no estuvieran en producción.

Para evitar este tipo de problemas, es conveniente hacer pruebas de rendimiento a las aplicaciones enteras bajo la mayor carga de trabajo posible. Para ello, puede utilizarse Super Smack, disponible en http://jeremy.zawodny.com/mysql/super-smack/. Como su nombre indica, puede poner de rodillas a un sistema, así que asegúrese de usarlo sólo en sistemas de desarrollo.

7.2. Optimizar sentencias `SELECT` y otras consultas

Ante todo, un factor afecta a todas las sentencias: cuanto más compleja es la configuración de los permisos, mayor es la carga.

Usar permisos simples cuando se ejecuta una sentencia `GRANT` permite a MySQL reducir la carga en la verificación de permisos cuando los clientes ejecutan sentencias. Por ejemplo, si no se concede (grant) ningún privilegio a nivel de tabla o de columna, el servidor no necesita verificar nunca el contenido de las tablas `tables_priv` y `columns_priv`. Similarmente, si no se establece límites de recursos sobre ninguna cuenta, el servidor no tiene que realizar conteos de recursos. Cuando se tiene un alto volumen de consultas, puede valer la pena utilizar una estructura de permisos simplificada para reducir la carga de verificación de los mismos.

Si el problema es con una expresión o función específica de MySQL, se puede utilizar la función `BENCHMARK()` en el cliente `mysql` para realizar una prueba de velocidad. Su sintaxis es `BENCHMARK(`*`número_iteraciones`*`,`*`expresion`*`)`. El valor devuelto siempre es cero, pero `mysql` muestra cuánto tardó en ejecutarse la sentencia. Por ejemplo:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
```

```
1 row in set (0.32 sec)
```

Este resultado se obtuvo en un sistema Pentium II 400MHz. Muestra que MySQL puede ejecutar 1,000,000 de expresiones simples de suma en 0.32 segundo sobre ese sistema.

Todas las funciones de MySQL deberían estar altamente optimizadas, pero pueden existir algunas excepciones. `BENCHMARK()` es una excelente herramienta para determinar si alguna función es un problema en una consulta.

7.2.1. Sintaxis de `EXPLAIN` (Obtener información acerca de un `SELECT`)

```
EXPLAIN nombre_de_tabla
```

O:

```
EXPLAIN SELECT opciones_de_select
```

La sentencia `EXPLAIN` puede utilizarse como un sinónimo de `DESCRIBE` o también como una manera para obtener información acerca de cómo MySQL ejecuta una sentencia `SELECT`:

- `EXPLAIN nombre_de_tabla` es sinónimo de `DESCRIBE nombre_de_tabla` o `SHOW COLUMNS FROM nombre_de_tabla`.
- Cuando se precede una sentencia `SELECT` con la palabra `EXPLAIN`, MySQL muestra información del optimizador sobre el plan de ejecución de la sentencia. Es decir, MySQL explica cómo procesaría el `SELECT`, proporcionando también información acerca de cómo y en qué orden están unidas (join) las tablas.

Esta sección trata sobre el segundo uso de `EXPLAIN`.

`EXPLAIN` es una ayuda para decidir qué índices agregar a las tablas, con el fin de que las sentencias `SELECT` encuentren registros más rápidamente. `EXPLAIN` puede utilizarse también para verificar si el optimizador une (join) las tablas en el orden óptimo. Si no fuera así, se puede forzar al optimizador a unir las tablas en el orden en el que se especifican en la sentencia `SELECT` empezando la sentencia con `SELECT STRAIGHT_JOIN` en vez de simplemente `SELECT`.

Si un índice no está siendo utilizado por las sentencias `SELECT` cuando debiera, debe ejecutarse el comando `ANALYZE TABLE`, a fin de actualizar las estadísticas de la tabla como la cardinalidad de sus claves, que pueden afectar a la decisiones que el optimizador toma. Ver Sección 13.5.2.1, “Sintaxis de ANALYZE TABLE”.

`EXPLAIN` muestra una línea de información para cada tabla utilizada en la sentencia `SELECT`. Las tablas se muestran en el mismo orden en el que MySQL las leería al procesar la consulta. MySQL resuelve todas las uniones (joins) usando un método de *single-sweep multi-join*. Esto significa que MySQL lee un registro de la primera tabla; encuentra su correspondiente en la segunda tabla, en la tercera, y así sucesivamente. Cuando todas las tablas han sido procesadas, MySQL muestra las columnas seleccionadas y recorre a la inversa la lista de tablas hasta que encuentra aquélla para la que la sentencia devuelve más registros. Se lee entonces el siguiente registro de esta tabla y el proceso continúa con la siguiente tabla.

`EXPLAIN` retorna una tabla; cada línea de esta tabla muestra información acerca de una tabla, y tiene las siguientes columnas:

- `id`

  The `SELECT` identifier. This is the sequential number of the `SELECT` within the query.

- `select_type`

The type of SELECT, which can be any of the following:

- 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)

- table

  The table to which the row of output refers.

- type

  The join type. The different join types are listed here, 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 with 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 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 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*:

```
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;
```

- 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 Sección 7.2.7, “Cómo optimiza MySQL IS NULL”.

- index_merge

This join type indicates that the Index Merge optimization is used. In this case, the key column 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 Sección 7.2.6, “Index Merge Optimization”.

- unique_subquery

  This type replaces 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 non-unique 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 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 for when a key column is compared to a constant using any of the =, <>, >, >=, <, <=, IS NULL, <=>, BETWEEN, or IN operators:

  ```
  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

  This join type is the same as ALL, except that only the index tree is scanned. This usually is faster than ALL, because the index file usually is smaller than the data file.

  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 allow row retrieval from the table based on constant values or column values from earlier tables.

- possible_keys

The possible_keys column indicates which indexes MySQL could 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 see 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 Sección 13.1.2, “Sintaxis de ALTER TABLE”.

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. The key is NULL if no index was chosen. 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 Sección 13.2.7, “Sintaxis de SELECT”.

  For MyISAM and BDB tables, running ANALYZE TABLE helps the optimizer choose better indexes. For MyISAM tables, myisamchk --analyze does the same. See Sección 13.5.2.1, “Sintaxis de ANALYZE TABLE” and Sección 5.8.3, “Mantenimiento de tablas y recuperación de un fallo catastrófico (crash)”.

- 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 allows you to determine how many parts of a multiple-part key MySQL actually uses.

- ref

  The ref column shows which columns or constants are used with the key to select rows from the table.

- rows

  The rows column indicates the number of rows MySQL believes it must examine to execute the query.

- Extra

  This column contains additional information about how MySQL resolves the query. Here is an explanation of the different text strings that can appear in this column:

  - Distinct

    MySQL stops searching for more rows for the current row combination after it has found the first matching row.

  - 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: #)

  MySQL found no good index to use, but found that some of indexes might be used once 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. The applicability criteria are as described in Sección 7.2.5, “Optimización de rango” and Sección 7.2.6, “Index Merge Optimization”, with the exception that all column values for the preceding table are known and considered to be constants.

  This is not very fast, but is faster than performing a join with no index at all.

- Using filesort

  MySQL needs to 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 Sección 7.2.10, “Cómo optimiza MySQL ORDER BY”.

- 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 temporary

  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.

  If you want to make your queries as fast as possible, you should look out for Extra values of Using filesort and Using temporary.

- Using sort_union(...), Using union(...), Using intersect(...)

  These indicate how index scans are merged for the index_merge join type. See Sección 7.2.6, “Index Merge Optimization” for more information.

- Using index for group-by

  Similar to the Using index way of accessing a table, 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 Sección 7.2.11, “Cómo optimiza MySQL los GROUP BY”.

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 product also is used to determine which multiple-table SELECT statements to execute. See Sección 7.5.2, “Afinar parámetros del servidor”.

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 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	Column Type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
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 (key map: 35)
```

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. Since `tt.ActualPC` is declared as `CHAR(10)` and `et.EMPLOYID` is `CHAR(15)`, 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);
```

`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 (key map: 1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      range checked for each record (key map: 1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better: The product of the `rows` values is less by a factor of 74. This version is executed 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);
```

`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
```

This is almost as good as it can get.

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;
```

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
```

Note that 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. If not, you may 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.

7.2.2. Estimar el renidimiento de una consulta

In most cases, you can estimate the 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 1024 bytes and the data pointer is usually 4 bytes. For a 500,000-row table with an index length of 3 bytes (medium integer), 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 (as above) to find where to place the new index and normally two seeks to update the index and write the row.

Note that the preceding discussion doesn't 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 Sección 7.5.2, “Afinar parámetros del servidor”.

7.2.3. Velocidad de las consultas SELECT

In general, when you want to make a slow `SELECT ... WHERE` query faster, the first thing to check is whether you can add an index. All references between different tables should usually be done with indexes. You can use the `EXPLAIN` statement to determine which indexes are used for a `SELECT`. See Sección 7.4.5, “Cómo utiliza MySQL los índices” and Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)”.

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 non-constant 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 (if you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all records in order according to the index. Note that the first time you sort a large table this way, it may take a long time.

7.2.4. Optimización de las cláusulas WHERE por parte de MySQL

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.

Note that work on the MySQL optimizer is ongoing, so this section is incomplete. MySQL does many optimizations, not all of which are documented here.

Some of the optimizations performed by MySQL are listed here:

- 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 HEAP 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 don't use GROUP BY or group 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 records as soon as possible.
-
- 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 `SQL_SMALL_RESULT`, 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. The optimizer is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size, so a fixed percentage no longer determines the choice between using an index or a scan.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.
- Before each record 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;
```

The following queries are resolved 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, ... ;
```

7.2.5. Optimización de rango

The `range` access method uses a single index to retrieve a subset of table records that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. A detailed description of how intervals are extracted from the `WHERE` clause is given in the following sections.

7.2.5.1. Método de acceso a rango para índices simples

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, so we'll talk about *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.
- For `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>`, `<`, `>=`, `<=`, `BETWEEN`, `!=`, or `<>` operators, or `LIKE 'pattern'` (where `'pattern'` doesn't start with a wildcard).
- For all types of indexes, 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';
```

Note that some non-constant values may be converted to constants during the 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 can't be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

For example, 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:

   ```
   (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 right way to remove them is to replace them with `TRUE`, so that we don't miss any matching records 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 in the 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 doesn't depend on the order in which conditions appear in `WHERE` clause.

7.2.5.2. 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 records 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'
   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.)

  See Sección 7.2.5.1, “Método de acceso a rango para índices simples” for the definition of what is considered to be a constant.

  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 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 records that match the condition (or two intervals if `<>` or `!=` is used). For example, for this condition:

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

  The single interval is:

```
('foo', 10, 10)
   < (key_part1, key_part2, key_part3)
      < ('foo', +inf, +inf)
```

  It is possible that the created interval contains more records 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 records contained within intervals are combined with `OR`, they form a condition that covers a set of records contained within the union of their intervals. If the conditions are combined with `AND`, they form a condition that covers a set of records 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 is:

```
(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
```

Sección 7.2.5.1, “Método de acceso a rango para índices simples” describes how optimizations are performed to combine or eliminate intervals for range conditions on single-part index. Analogous steps are performed for range conditions on multiple-part keys.

7.2.6. Index Merge Optimization

The Index Merge (`index_merge`) method is used to retrieve rows with several `ref`, `ref_or_null`, or `range` scans and merge the results into one. This method is employed when the table condition is a disjunction of conditions for which `ref`, `ref_or_null`, or `range` could be used with different keys.

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, this 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 key_part1 = 10 OR key_part2 = 20;

SELECT * FROM tbl_name
    WHERE (key_part1 = 10 OR key_part2 = 20) AND non_key_part=30;

SELECT * FROM t1, t2
    WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
    AND t2.key1=t1.some_col;

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):

- intersection
- union
- sort-union

The following sections describe these methods in greater detail.

Note: The Index Merge optimization algorithm has the following known deficiencies:

- If a range scan is possible on some key, an Index Merge is not considered. For example, consider this query:

```
SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

  For this query, two plans are possible:

  1. An Index Merge scan using the `(goodkey1 < 10 OR goodkey2 < 20)` condition.
  2. A range scan using the `badkey < 30` condition.

  However, the optimizer only considers the second plan. If that is not what you want, you can make the optimizer consider `index_merge` by using `IGNORE INDEX` or `FORCE INDEX`. The following queries are executed using Index Merge:

```
SELECT * FROM t1 FORCE INDEX(goodkey1,goodkey2)
WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

SELECT * FROM t1 IGNORE INDEX(badkey)
WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

- If your query has a complex `WHERE` clause with deep `AND`/`OR` nesting and MySQL doesn't 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)
```

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.

7.2.6.1. 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.

Here are some 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 records are not retrieved and (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes don't cover all columns used in the query, full records 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 record retrieval, but is used to filter out records retrieved using other conditions.

7.2.6.2. Algoritmo de acceso a 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.

Here are some 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;
```

7.2.6.3. 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.

Here are some 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 records and sort them before returning any records.

7.2.7. Cómo optimiza MySQL IS NULL

MySQL can perform the same optimization on *col_name* IS NULL that it can use with *col_name* = *constant_value*. For example, MySQL can use indexes and ranges to search for NULL with IS NULL.

```
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 5.0 can also optimize the combination *col_name = expr* AND *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.

Note that 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);
```

7.2.8. Cómo MySQL optimiza DISTINCT

DISTINCT combined with ORDER BY needs a temporary table in many cases.

Note that because DISTINCT may use GROUP BY, you should be aware of how MySQL works with columns in ORDER BY or HAVING clauses that are not part of the selected columns. See Sección 12.10.3, “GROUP BY con campos escondidos”.

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 Sección 7.2.11, “Cómo optimiza MySQL los GROUP BY”.

When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.

If you don't use columns from all tables named in a query, MySQL stops scanning the not-used 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 the first row in `t2` is found:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

7.2.9. Cómo optimiza MySQL los LEFT JOIN y RIGHT JOIN

An *A* `LEFT JOIN` *B* `join_condition` is implemented in MySQL 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 done, 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.
- All standard `WHERE` optimizations are done.
- 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 don't 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.

`RIGHT JOIN` is implemented analogously to `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` and `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,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 `a` and `b` are listed in the `FROM` clause:

```
SELECT *
    FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
    WHERE b.key=d.key;
```

MySQL 5.0 performs the following `LEFT JOIN` optimization: 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 this would result in a better query plan. To force a specific table order, use `STRAIGHT_JOIN`.

7.2.10. Cómo optimiza MySQL ORDER BY

In some cases, MySQL can use an index to satisfy an `ORDER BY` clause without doing any extra sorting.

The index can also be used even if the `ORDER BY` does not match the index exactly, as long as all of the unused portions of the index and all the 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 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;
```

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:

- You use `ORDER BY` on different keys:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- You use `ORDER BY` on non-consecutive parts of a key:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
  ```

- You mix `ASC` and `DESC`:

  ```
  SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
  ```

- The key used to fetch the rows is not the same as the one used in the `ORDER BY`:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- You are joining many tables, and the columns in the `ORDER BY` are not all from the first non-constant table that is used to retrieve rows. (This is the first table in the `EXPLAIN` output that doesn't have a `const` join type.)
- You have different `ORDER BY` and `GROUP BY` expressions.
- The type of table index used doesn't store rows in order. For example, this is true for a `HASH` index in a `HEAP` table.

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 Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)”.

In MySQL 5.0, a `filesort` optimization is used that records not only the sort key value and row position, but the columns required for the query as well. This avoids reading the rows twice. The `filesort` algorithm works like this:

1. Read the rows that match the `WHERE` clause, as before.

2. For each row, record a tuple of values consisting of the sort key value and row position, and also the columns required for the query.
3. Sort the tuples by sort key value
4. Retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

This algorithm represents an improvement over that used in some older versions of MySQL.

To avoid a slowdown, this optimization is used 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 that you see high disk activity and low CPU activity.)

If you want to increase `ORDER BY` speed, first see 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 size of the `sort_buffer_size` variable.
- Increase the size of the `read_rnd_buffer_size` variable.
- Change `tmpdir` to point to a dedicated filesystem with large amounts of empty space. In MySQL 5.0, 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. You can use this feature to spread the load across several directories. *Note:* The paths should be for directories in filesystems that are located on different *physical* disks, not different partitions on the same disk.

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 `ORDER BY` clause explicitly 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;
```

7.2.11. Cómo optimiza MySQL los GROUP BY

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 the index stores its keys in order (for example, this is a B-Tree index, and not a HASH index). Whether usage 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 via 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.

7.2.11.1. Loose index scan

The most efficient way is when the index is used to directly retrieve the group fields. With this access method, MySQL uses the property of some index types (for example, B-Trees) that the keys are ordered.

This property allows use of lookup groups in an index without having to consider all keys in the index that satisfy all WHERE conditions. Since this access method considers only a fraction of the keys in an index, 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 in Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)” of the range join type), a loose index scan looks up the first key of each group 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 includes the first consecutive parts of the index (if instead of GROUP BY, the query has a DISTINCT clause, then all distinct attributes refer to the beginning of the index).
- The only aggregate functions used (if any) are MIN() and MAX(), and all of them refer to the same column.
- 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.

The EXPLAIN output for such queries shows Using index for group-by in the Extra column.

The following queries provide several examples that fall into this category, assuming there is an index idx(c1, c2, c3) on table t1(c1,c2,c3,c4):

```
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 other aggregate functions than MIN() or MAX(), for example:

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

- The fields in the GROUP BY clause do not refer to the beginning of the index, as shown here:

```
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, an example being:

```
SELECT c1,c3 FROM t1 GROUP BY c1, c2;
```

7.2.11.2. Tight index scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it is still 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. Since 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* . Notice that 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, for all columns in a query referring to parts of the key coming before or in between parts of the GROUP BY key, there is a constant equality condition. 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 can be then 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.

The following queries do not work with the first method above, but still work with the second index access method (assuming we have the aforementioned index idx on table t1):

- 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;
```

7.2.12. Cómo optimiza MySQL las cláusulas LIMIT

In some cases, MySQL handles a query differently when you are using LIMIT *row_count* and not using HAVING:

- If you are selecting 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 use 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, once the rows have been found, there is no need to sort any remainder of the result set, and MySQL does not do so.
- When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.
- In some cases, a GROUP BY can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key 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.
- 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 trick does not work in the MySQL Monitor, which merely displays Empty set in such cases; you should instead use SHOW COLUMNS or DESCRIBE for this purpose.)
- When the server uses temporary tables to resolve the query, the LIMIT *row_count* clause is used to calculate how much space is required.

7.2.13. Cómo evitar lecturas completas de tablas

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 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 Sección 7.2.4, “Optimización de las cláusulas `WHERE` por parte de MySQL”.
- 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 does a lot of key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate. 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 Sección 13.5.2.1, “Sintaxis de `ANALYZE TABLE`”.
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index. See Sección 13.2.7, “Sintaxis de `SELECT`”.

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

- 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 Sección 5.3.3, “Variables de sistema del servidor”.

7.2.14. Velocidad de la sentencia `INSERT`

The time required for inserting a record is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting record: (1 x size of record)
- Inserting indexes: (1 x 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 non-empty table, you may tune the bulk_insert_buffer_size variable to make data insertion even faster. See Sección 5.3.3, “Variables de sistema del servidor”.
- If you are inserting a lot of rows from different clients, you can get higher speed by using the INSERT DELAYED statement. See Sección 13.2.4, “Sintaxis de INSERT”.
- With MyISAM tables you can insert rows at the same time that SELECT statements are running if there are no deleted rows in the tables.
- When loading a table from a text file, use LOAD DATA INFILE. This is usually 20 times faster than using a lot of INSERT statements. See Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.
- With some extra work, it is possible to make LOAD DATA INFILE run even faster when the table has many indexes. Use the following procedure:

  1. Optionally create the table with CREATE TABLE.
  2. Execute a FLUSH TABLES statement or a mysqladmin flush-tables command.
  3. Use myisamchk --keys-used=0 -rq */path/to/db/tbl_name*. This removes all use of indexes for the table.
  4. Insert data into the table with LOAD DATA INFILE. This does not update any indexes and therefore is very fast.
  5. If you intend only to read from the table in the future, use myisampack to compress it. See Sección 14.1.3.3, “Características de las tablas comprimidas”.
  6. Re-create the indexes with myisamchk -r -q */path/to/db/tbl_name*. This creates the index tree in memory before writing it to disk, which is much faster because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
  7. Execute a FLUSH TABLES statement or a mysqladmin flush-tables command.

  Note that LOAD DATA INFILE also performs the preceding optimization if you insert into an empty MyISAM table; the main difference 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.

  In MySQL 5.0, you can also use ALTER TABLE *tbl_name* DISABLE KEYS instead of myisamchk --keys-used=0 -rq */path/to/db/tbl_name* and ALTER TABLE *tbl_name* ENABLE KEYS instead of myisamchk -r -q */path/to/db/tbl_name*. In this way, you can also skip the FLUSH TABLES steps.

- You can speed up INSERT operations that are done with multiple statements by locking 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 statement.

For transactional tables, you should use BEGIN and COMMIT instead of LOCK TABLES to obtain faster insertions.

Locking also lowers the total time of multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. For example:

```
Connection 1 does 1000 inserts
Connections 2, 3, and 4 do 1 insert
Connection 5 does 1000 inserts
```

If you don't 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 inserts or updates in a row. If you do very many inserts in a row, you could do a LOCK TABLES followed by an UNLOCK TABLES once in a while (about each 1,000 rows) to allow other threads access to the 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 get some more speed for MyISAM tables, for both LOAD DATA INFILE and INSERT, enlarge the key cache by increasing the key_buffer_size system variable. See Sección 7.5.2, “Afinar parámetros del servidor”.

7.2.15. Velocidad de las sentencias UPDATE

Update statements are optimized as 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 does not get updated.

Another way to get fast updates is to delay updates and then do many updates in a batch later. This is much quicker than doing one at a time if you lock the table.

Note that for a MyISAM table that uses dynamic record format, updating a record to a longer total length may split the record. If you do this often, it is very important to use OPTIMIZE TABLE occasionally. See Sección 13.5.2.5, “Sintaxis de OPTIMIZE TABLE”.

7.2.16. Velocidad de sentencias DELETE

The time to delete individual records is exactly proportional to the number of indexes. To delete records more quickly, you can increase the size of the key cache. See Sección 7.5.2, “Afinar parámetros del servidor”.

If you want to delete all rows from a table, use TRUNCATE TABLE tbl_name rather than DELETE FROM *tbl_name*. See Sección 13.2.9, “Sintaxis de TRUNCATE”.

7.2.17. Otros consejos sobre optimización

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid connection overhead. If you can't use persistent connections and you are initiating many new connections to the database, you may want to change the value of the thread_cache_size variable. See Sección 7.5.2, “Afinar parámetros del servidor”.

- Always check whether all your queries really use the indexes you have created in the tables. In MySQL, you can do this with the `EXPLAIN` statement. See Sección 7.2.1, “Sintaxis de `EXPLAIN` (Obtener información acerca de un `SELECT`)”.
- 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.
- With `MyISAM` tables that have no deleted rows, you can insert rows at the end at the same time that another query is reading from the table. If this is important for you, you should consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` after you have deleted a lot of rows.
- Use `ALTER TABLE ... ORDER BY` *`expr1, expr2, ...`* if you mostly retrieve rows in *`expr1, expr2, ...`* order. By using this option after extensive changes to the table, you may be able to get higher performance.
- In some cases, it may make sense to introduce a column that is “hashed” based on information from other columns. If this column is short and reasonably unique, it may be much faster than a big index on many columns. In MySQL, it is very easy to use this extra column:

  ```
  SELECT * FROM tbl_name
      WHERE hash_col=MD5(CONCAT(col1,col2))
      AND col1='constant' AND col2='constant';
  ```

- For `MyISAM` tables that change a lot, you should try to avoid all variable-length columns (`VARCHAR`, `BLOB`, and `TEXT`). The table uses dynamic record format if it includes even a single variable-length column. See Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.
- It is normally not useful to split a table into different tables just because rows become large. To access a row, the biggest performance hit is the disk seek to find the first byte of the row. After finding the data, most modern disks can read the entire row quickly enough for most applications. The only case in which it is advantageous to split up a table is if the table is a `MyISAM` table with dynamic record format (see above) that you can change to a fixed record size, or if you very often need to scan the table but do not need most of the columns. See Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.
- 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 important when you use a MySQL storage engine such as `MyISAM` that has only table-level locking (multiple readers / single writers). This gives better performance with most databases, because the row locking manager in this case has less to do.
- If you need to collect statistics from large log tables, use summary tables instead of scanning the entire log table. Maintaining the summaries should be much faster than trying to calculate statistics “live”. Regenerating new summary tables from the logs when things change (depending on business decisions) is faster than changing the running application.
- If possible, you should 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.
- When possible, take advantage columns' default values to insert values explicitly only when the value to be inserted differs from the default. This reduces the amount parsing required of MySQL and so improves the speed of insert operations.

- In some cases, it is convenient to 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 may save a lot of accesses at some stage. This is practical when you have data that does not conform well to a rows-and-columns table structure.
- Normally, you should try to keep all data non-redundant (observing what is referred to in database theory as *third normal form*). However, there may be situations in which it can be advantageous to duplicate information or create summary tables in order to gain more speed.
- Stored procedures or UDFs (user-defined functions) may be a good way to gain performance for some tasks. See Capítulo 19, *Procedimientos almacenados y funciones* and Sección 27.2, “Añadir nuevas funciones a MySQL” for more information about these.
- You can always gain something by caching queries or answers in your application and then performing many inserts or updates together. If your database supports table locks (like MySQL and Oracle), this should help to ensure 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 Sección 5.12, “La caché de consultas de MySQL”.
- Use `INSERT DELAYED` when you do not need to know when your data is written. This speeds things up because many records 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.
- Use multiple-row `INSERT` statements to store many rows with one SQL statement (many SQL servers support this, including MySQL).
- Use `LOAD DATA INFILE` to load large amounts of data. This is faster than using `INSERT` statements.
- Use `AUTO_INCREMENT` columns to generate unique values.
- Use `OPTIMIZE TABLE` once in a while to avoid fragmentation with `MyISAM` tables when using a dynamic table format. See Sección 14.1.3, “Formatos de almacenamiento de tablas `MyISAM`”.
- Use `MEMORY` tables when possible to get more speed. See Sección 14.3, “El motor de almacenamiento `MEMORY` (`HEAP`)”.
- With Web servers, images and other binary assets should normally be stored as files. That is, store only a reference to the file rather than the file itself in the database. Most Web servers are better at caching files than database contents, so using files is generally faster.
- Use in-memory tables for non-critical data that is accessed often, such as information about the last displayed banner for users who don't have cookies enabled in their Web browser. User sessions are another alternative available in many Web application environments for handling volatile state data.
- Columns with identical information in different tables should be declared to have identical data types.

  Try to keep column names simple. 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, you should keep them shorter than 18 characters.
- If you need really high speed, you should take a look at the low-level interfaces for data storage that the different SQL servers support. For example, by accessing the MySQL `MyISAM` storage engine directly,

you could get a speed increase of two to five times compared to using the SQL interface. To be able to do this, the data must be on the same server as the application, and usually it should only be accessed by one process (because external file locking is really slow). One could eliminate these problems by introducing low-level `MyISAM` commands in the MySQL server (this could be one easy way to get more performance if needed). By carefully designing the database interface, it should be quite easy to support this types of optimization.

- If you are using numerical 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 don't have to parse your text files to find line and column boundaries.
- Replication can provide performance benefits 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 Capítulo 6, *Replicación en MySQL*.
- 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 should ensure that they are okay by running the server with the `--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.)

7.3. Temas relacionados con el bloqueo

7.3.1. Métodos de bloqueo

MySQL 5.0 supports table-level locking for `MyISAM` and `MEMORY` 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 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.

The table-locking method MySQL uses for `WRITE` locks works as follows:

- If there are no locks on the table, put a write lock on it.
- Otherwise, put the lock request in the write lock queue.

The table-locking method MySQL uses for `READ` locks works as follows:

- If there are no write locks on the table, put a read lock on it.
- Otherwise, put the lock request in the read lock queue.

When a lock is released, the lock is made available to the threads in the write lock queue, then to the threads in the read lock queue.

This means that if you have many updates for a table, `SELECT` statements wait until there are no more updates.

You can analyze the table lock contention on your system by checking the `Table_locks_waited` and `Table_locks_immediate` status variables:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

You can freely mix concurrent `INSERT` and `SELECT` statements for a `MyISAM` table without locks if the `INSERT` statements are non-conflicting. That is, you can insert rows into a `MyISAM` table at the same time other clients are reading from it. No conflict occurs if the data file contains no free blocks in the middle, because in that case, records always are inserted at the end of the data file. (Holes can result from rows having been deleted from or updated in the middle of the table.) If there are holes, concurrent inserts are re-enabled automatically when all holes have been filled with new data.

If you want to do many `INSERT` and `SELECT` operations on a table when concurrent inserts are not possible, you can insert rows in a temporary table and update the real table with the records from the temporary table once in a while. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, insert_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM insert_table;
mysql> TRUNCATE TABLE insert_table;
mysql> UNLOCK TABLES;
```

`InnoDB` uses row locks and `BDB` uses page locks. For these two storage engines, deadlocks are possible. This is because `InnoDB` automatically acquires row locks and `BDB` acquires page locks during the processing of SQL statements, not at the start of the transaction.

Advantages of row-level locking:

- Fewer lock conflicts when accessing different rows in many threads.
- Fewer changes for rollbacks.
- Makes it possible to lock a single row a long time.

Disadvantages of row-level locking:

- Takes more memory than page-level or table-level locks.
- Is slower than page-level or table-level locks when used on a large part of the table because you must acquire many more locks.
- Is definitely much worse than other locks if you often do `GROUP BY` operations on a large part of the data or if you often must scan the entire table.
- With higher-level locks, you can also more easily support locks of different types to tune the application, because the lock overhead is less than for row-level locks.

Table locks are superior to page-level or row-level locks in the following cases:

- Most statements for the table are reads.

- Reads and updates on strict keys, where you update or delete a row that can be fetched with a single 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.

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 `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only in well-behaved applications.

7.3.2. Cuestiones relacionadas con el bloqueo (locking) de tablas

To achieve a very high lock speed, MySQL uses table locking (instead of page, row, or column locking) for all storage engines except `InnoDB` and `BDB`.

For `InnoDB` and `BDB` tables, MySQL uses only table locking if you explicitly lock the table with `LOCK TABLES`. For these table types, we recommend that you not use `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 much better than row locking for most applications, but there are some pitfalls.

Table locking enables many threads to read from a table at the same time, but if a thread wants to write to a table, it must first get exclusive access. During the update, all other threads that want to access this particular table must wait until the update is done.

Table updates normally are considered to be more important than table retrievals, so they are given higher priority. This should ensure that updates to a table are not “starved” even if there is heavy `SELECT` activity for the table.

Table locking causes problems in cases such as when a thread is waiting because the disk is full and free space needs to become available before the thread can proceed. In this case, all threads 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 client issues a `SELECT` that takes a long time to run.
- Another client then issues an `UPDATE` on the same table. This client waits until the `SELECT` is finished.
- Another client issues another `SELECT` statement on the same table. Because `UPDATE` has higher priority than `SELECT`, this `SELECT` waits for the `UPDATE` to finish, *and* for the first `SELECT` to finish.

The following list describes some ways to avoid or reduce contention caused by table locking:

- Try to get the `SELECT` statements to run faster. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates`. 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.
- You can specify that all updates issued in a specific connection should be done with low priority by using the `SET LOW_PRIORITY_UPDATES=1` statement. See Sección 13.5.3, “Sintaxis de `SET`”.
- You can give a specific `INSERT`, `UPDATE`, or `DELETE` statement lower priority with the `LOW_PRIORITY` attribute.
- You can give a specific `SELECT` statement higher priority with the `HIGH_PRIORITY` attribute. See Sección 13.2.7, “Sintaxis de `SELECT`”.
- You can 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 allows `READ` locks after a certain number of `WRITE` locks.
- If you have problems with `INSERT` combined with `SELECT`, you might want to consider switching `MyISAM` tables, which support concurrent `SELECT` and `INSERT` statements.
- If you mix inserts and deletes on the same table, `INSERT DELAYED` may be of great help. See Sección 13.2.4.2, “Sintaxis de `INSERT DELAYED`”.
- If you have problems with mixed `SELECT` and `DELETE` statements, the `LIMIT` option to `DELETE` may help. See Sección 13.2.1, “Sintaxis de `DELETE`”.
- Using `SQL_BUFFER_RESULT` with `SELECT` statements can help to make the duration of table locks shorter. See Sección 13.2.7, “Sintaxis de `SELECT`”.
- 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:

- Concurrent users are not a problem if you don't mix updates with selects that need to examine many rows in the same table.
- You can use `LOCK TABLES` to increase speed, as many updating 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 Capítulo 15, *El motor de almacenamiento `InnoDB`*. See Sección 14.4, “El motor de almacenamiento `BDB` (`BerkeleyDB`)”.

7.4. Optimizar la estructura de una base de datos

7.4.1. Elecciones de diseño

MySQL keeps row data and index data in separate files. Many (almost all) other databases mix row and index data in the same file. We believe that the MySQL choice is better for a very wide range of modern systems.

Another way to store the row data is to keep the information for each column in a separate area (examples are SDBM and Focus). This causes a performance hit for every query that accesses more than one column. Because this degenerates so quickly when more than one column is accessed, we believe that this model is not good for general-purpose databases.

The more common case is that the index and data are stored together (as in Oracle/Sybase, et al). In this case, you find the row information at the leaf page of the index. The good thing with this layout is that it, in many cases, depending on how well the index is cached, saves a disk read. The bad things with this layout are:

- Table scanning is much slower because you have to read through the indexes to get at the data.
- You can't use only the index table to retrieve data for a query.
- You use more space because you must duplicate indexes from the nodes (you can't store the row in the nodes).
- Deletes degenerate the table over time (because indexes in nodes are usually not updated on delete).
- It's harder to cache only the index data.

7.4.2. Haga sus datos lo más pequeños posibles

One of the most basic optimizations is to design your tables to take as little space on the disk as possible. This can give huge improvements because disk reads are faster, and smaller tables normally require less main memory while their contents are being actively processed during query execution. Indexing also is a lesser resource burden if done on smaller columns.

MySQL supports a lot of different table types and row formats. For each table, you can decide which storage/index method to use. Choosing the right table format for your application may give you a big performance gain. See Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

You can get better performance on a table and minimize storage space using the techniques listed here:

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory.
- Use the smaller integer types if possible to get smaller tables. For example, `MEDIUMINT` is often a better choice than `INT` since a `MEDIUMINT` column uses 25% less space.
- Declare columns to be `NOT NULL` if possible. It makes everything faster and you save one bit per column. If you really need `NULL` in your application, you should definitely use it. Just avoid having it on all columns by default.
- For `MyISAM` tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size record format is used. This is faster but unfortunately may waste some space. See Sección 14.1.3, “Formatos de almacenamiento de tablas `MyISAM`”. You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE` option `ROW_FORMAT=fixed`.
- Starting with MySQL/InnoDB 5.0.3, `InnoDB` tables use a more compact storage format. In earlier versions of MySQL, InnoDB records 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/InnoDB, you can request the old format with `ROW_FORMAT=REDUNDANT`.

  The compact InnoDB format also changes the way how `CHAR` columns containing UTF-8 data are stored. In the `ROW_FORMAT=REDUNDANT` format, a UTF-8 `CHAR(`*n*`)` occupies 3**n* bytes, given that the maximum length of a UTF-8 encoded character is 3 bytes. Since many languages can be

written mostly with single-byte UTF-8 characters, a fixed storage length often wastes space. The `ROW_FORMAT=COMPACT` format allocates a variable amount of *n*..3**n* bytes for these columns by stripping trailing spaces if necessary. The minimum storage length is kept as *n* bytes in order to facilitate in-place updates in typical cases.

- 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 really need. Indexes are good for retrieval but bad when you need to store data quickly. If you access a table mostly by searching on a combination of columns, make an index on them. The first index part should be the most used column. If you *always* use many columns when selecting from the table, you should use the column with more duplicates first to obtain better compression of the index.
- If it is very likely that a column has a unique prefix on the first number of characters, it is better to index only this prefix. MySQL supports an index on the leftmost part of a character column. Shorter indexes are faster not only because they take less disk space, but also because they give you more hits in the index cache and thus fewer disk seeks. See Sección 7.5.2, “Afinar parámetros del servidor”.
- 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.

7.4.3. Índices de columna

All MySQL column 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 Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*. 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.

With *`col_name(length)`* syntax in an index specification, you can create an index that uses only the first *`length`* characters of a `CHAR` or `VARCHAR` column. Indexing only a prefix of column values like this can make the index file much smaller.

The `MyISAM` and `InnoDB` storage engines also support indexing on `BLOB` and `TEXT` columns. When indexing 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)));
```

Im MySQL 5.0, prefixes can be up to 1000 bytes long for `MyISAM` and `InnoDB` tables. Note that prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE` statements is interpreted as number of characters. *Be sure to take this into account when specifying a prefix length for a column that uses a multi-byte character set*.

You can also create `FULLTEXT` indexes. These are used for full-text searches. In MySQL 5.0, 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 partial (prefix) indexing is not supported. See Sección 12.7, “Funciones de búsqueda de texto completo (Full-Text)” for details.

In MySQL 5.0, you can also create indexes on spatial column types. Spatial types are supported only by the `MyISAM` storage engine. Spatial indexes use R-trees.

The `MEMORY` (`HEAP`) storage engine uses hash indexes by default, but also supports B-tree indexes in MySQL 5.0.

7.4.4. Índices de múltiples columnas

MySQL can create indexes on multiple columns. An index may consist of up to 15 columns. For certain column types, you can index a prefix of the column (see Sección 7.4.3, “Índices de columna”).

A multiple-column index can be considered a sorted array containing values that are created by concatenating the values of the indexed columns.

MySQL uses multiple-column indexes in such a way that queries are fast when you specify a known quantity for the first column of the index in a `WHERE` clause, even if you don't specify values for the other columns.

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 `last_name` and `first_name`. The index can be used for queries that specify values in a known range for `last_name`, or for both `last_name` and `first_name`. Therefore, the `name` index is used 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 in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

The manner in which MySQL uses indexes to improve query performance is discussed further in the next section.

7.4.5. Cómo utiliza MySQL los índices

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first record 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. If a table has 1,000 rows, then this is at least 100 times faster than reading sequentially. Note that if you need to access most of the rows, it is faster to read sequentially, because this minimizes disk seeks.

Most MySQL indexes (`PRIMARY KEY`, `UNIQUE`, `INDEX`, and `FULLTEXT`) are stored in B-trees. Exceptions are that indexes on spatial column types use R-trees, and that `MEMORY` tables also support hash indexes.

Strings are automatically prefix- and end-space compressed. See Sección 13.1.4, “Sintaxis de `CREATE INDEX`”.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in `MEMORY` tables) are described at the end of this section.

Indexes are used 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.
- To retrieve rows from other tables when performing joins.
- 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_# = 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 replace 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 key (for example, `ORDER BY` *`key_part1, key_part2`*). If all key parts are followed by `DESC`, the key is read in reverse order. See Sección 7.2.10, “Cómo optimiza MySQL `ORDER BY`”.
- In some cases, a query can be optimized to retrieve values without consulting the data rows. If a query uses only columns from a table that are numeric and that form a leftmost prefix for some key, the selected values may be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
  WHERE key_part1=1
```

Suppose that you issue the following `SELECT` statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

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 tries to find the most restrictive index by deciding which index finds fewer 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 find 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 a partial index 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 col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first of the preceding queries uses the index. The second and third queries do involve indexed columns, but `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

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 doesn't 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.

MySQL 5.0 performs an additional `LIKE` optimization. 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 employs 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 AND parts */
... 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, since it requires fewer seeks.) However, if such a query uses `LIMIT` to only retrieve 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 indexes have somewhat different characteristics than 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.
- 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 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.)

7.4.6. La caché de claves de MyISAM

To minimize disk I/O, the MyISAM storage engine employs a strategy that is used by many database management systems. It exploits a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (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 filesystem cache.

This section first describes the basic operation of the MyISAM key cache. Then it discusses recent changes (made in MySQL 4.1) 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 threads can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

You can control the size of the key cache by means of 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 filesystem 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 non-leaf nodes.)

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, the least recently used index block is selected. To make such a choice easier, the key cache module maintains a special queue (known as an *LRU chain*) of all used blocks. When a block is accessed, it is placed at the end of the queue. When blocks need to be replaced, blocks at the beginning of the queue are the least recently used and become the first candidates for eviction.

7.4.6.1. Acceso compartido a la caché de claves

In older versions of MySQL, access to the key cache was serialized, and no two threads could access key cache buffers simultaneously. However, in MySQL 5.0, the server supports shared access to the key cache:

- A buffer that is not being updated can be accessed by multiple threads.
- A buffer that is being updated causes threads that need to use it to wait until the update is complete.
- Multiple threads 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 allows the server to improve throughput significantly.

7.4.6.2. Múltiples cachés de claves

Shared access to the key cache improves performance but does not eliminate contention among threads entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL 5.0 also provides multiple key caches, which allow 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 Sección 13.5.5.1, “Sintaxis de CACHE INDEX”).

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       |
+---------+--------------------+----------+----------+
```

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;
```

Note that you cannot destroy the default key cache. An attempt to do this will be 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 Sección 9.4.1, “Variables estructuradas de sistema” 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, we recommend a strategy that uses 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. Queries that access tables assigned to one cache do not compete with queries 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 non-leaf 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.

`CACHE INDEX` 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 a.t1, a.t2, b.t3 IN hot_cache
```

```
CACHE INDEX a.t4, b.t5, b.t6 IN cold_cache
```

7.4.6.3. Midpoint Insertion Strategy

By default, the key cache management system in MySQL 5.0 uses the 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 sub-chain and a warm sub-chain. 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 sub-chain. After a certain number of hits (accesses of the block), it is promoted to the hot sub-chain. 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 sub-chain is placed at the end of the chain. The block then circulates within this sub-chain. If the block stays at the beginning of the sub-chain for a long enough time, it is demoted to the warm chain. 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 sub-chain not accessed within the last *`N`*`*key_cache_age_threshold/100` hits is to be moved to the beginning of the warm sub-chain. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sub-chain.

The midpoint insertion strategy allows 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 sub-chain during an index scan operation as well.

7.4.6.4. Precarga de índices

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its non-leaf nodes, then it makes sense to preload the key cache with index blocks before starting to use it. Preloading allows 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 a random order, 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 non-leaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the non-leaf 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.

7.4.6.5. Tamaño de bloque de la caché de claves

In MySQL 5.0, 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.

Currently, you cannot control the size of the index blocks in a table. This size is set by the server when the `.MYI` index file is created, depending on the size of the keys in the indexes present in the table definition. In most cases, it is set equal to the I/O buffer size.

7.4.6.6. Reestructurar la caché de claves

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 set 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 filesystem caching. Filesystem caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. Once the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of filesystem caching for the indexes ceases.

7.4.7. Cómo cuenta MySQL las tablas abiertas

When you execute a `mysqladmin status` command, you 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 threads having different states on the same table, the table is opened independently by each concurrent thread. 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 threads.)

You can read more about this topic in the next section, Sección 7.4.8, “Cómo abre y cierra tablas MySQL”.

7.4.8. Cómo abre y cierra tablas MySQL

The `table_cache`, `max_connections`, and `max_tmp_tables` system variables affect the maximum number of files the server keeps open. If you increase one or more 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 allow 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 also need to 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_safe`. See Sección A.2.17, “No se encontró el fichero”.

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 in order to execute queries.

An unused table is closed and removed 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.
- 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 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 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 with other threads and is not closed until the thread calls `HANDLER *tbl_name* CLOSE` or until the thread terminates. When this happens, the table is returned to the table cache (if the cache isn't full). See Sección 13.2.3, “Sintaxis de HANDLER”.

You can determine whether your table cache is too small by checking the `mysqld` status variable `Opened_tables`:

```
mysql> SHOW STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

If the value is large, even when you have not issued a lot of `FLUSH TABLES` statements, you should increase the table cache size. See Sección 5.3.3, “Variables de sistema del servidor” and Sección 5.3.4, “Variables de estado del servidor”.

7.4.9. Desventajas de crear muchas tablas en la misma base de datos

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 making the table cache larger.

7.5. Optimización del servidor MySQL

7.5.1. Factores de sistema y afinamientos de parámetros de arranque

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 things at this level.

The default operating system to use is very important! To get the best use of multiple-CPU machines, you should use Solaris (because its threads implementation works well) or Linux (because the 2.4 and later kernels have good SMP support). Note that older Linux kernels have a 2GB filesize limit by default. If you have such a kernel and a need for files larger than 2GB, you should get the Large File Support (LFS) patch for the ext2 filesystem. Other filesystems such as ReiserFS and XFS do not have this 2GB limitation.

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.
- Use the `--skip-external-locking` MySQL option to avoid external locking. This option is on by default in MySQL 5.0.

  Note that the `--skip-external-locking` option 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 this option is mandatory, because the external locking does not work in any case.

  The only case where you cannot use `--skip-external-locking` is if 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.

  You can still use `LOCK TABLES` and `UNLOCK TABLES` even if you are using `--skip-external-locking`.

7.5.2. Afinar parámetros del servidor

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:

```
back_log                          50
binlog_cache_size                 32768
bulk_insert_buffer_size           8388608
connect_timeout                   5
date_format                       (No default value)
datetime_format                   (No default value)
default_week_format               0
delayed_insert_limit              100
delayed_insert_timeout            300
delayed_queue_size                1000
expire_logs_days                  0
flush_time                        1800
ft_max_word_len                   84
ft_min_word_len                   4
ft_query_expansion_limit          20
ft_stopword_file                  (No default value)
group_concat_max_len              1024
innodb_additional_mem_pool_size   1048576
innodb_autoextend_increment       8
innodb_buffer_pool_awe_mem_mb     0
innodb_buffer_pool_size           8388608
innodb_concurrency_tickets        500
innodb_file_io_threads            4
innodb_force_recovery             0
innodb_lock_wait_timeout          50
innodb_log_buffer_size            1048576
innodb_log_file_size              5242880
innodb_log_files_in_group         2
innodb_mirrored_log_groups        1
innodb_open_files                 300
innodb_sync_spin_loops            20
innodb_thread_concurrency         8
innodb_thread_sleep_delay         10000
interactive_timeout               28800
join_buffer_size                  131072
key_buffer_size                   8388600
key_cache_age_threshold           300
key_cache_block_size              1024
key_cache_division_limit          100
long_query_time                   10
lower_case_table_names            1
max_allowed_packet                1048576
max_binlog_cache_size             4294967295
max_binlog_size                   1073741824
max_connect_errors                10
max_connections                   100
max_delayed_threads               20
max_error_count                   64
max_heap_table_size               16777216
max_join_size                     4294967295
max_length_for_sort_data          1024
max_relay_log_size                0
max_seeks_for_key                 4294967295
max_sort_length                   1024
max_tmp_tables                    32
max_user_connections              0
max_write_lock_count              4294967295
```

```
multi_range_count                 256
myisam_block_size                 1024
myisam_data_pointer_size          6
myisam_max_extra_sort_file_size   2147483648
myisam_max_sort_file_size         2147483647
myisam_repair_threads             1
myisam_sort_buffer_size           8388608
net_buffer_length                 16384
net_read_timeout                  30
net_retry_count                   10
net_write_timeout                 60
open_files_limit                  0
optimizer_prune_level             1
optimizer_search_depth            62
preload_buffer_size               32768
query_alloc_block_size            8192
query_cache_limit                 1048576
query_cache_min_res_unit          4096
query_cache_size                  0
query_cache_type                  1
query_cache_wlock_invalidate      FALSE
query_prealloc_size               8192
range_alloc_block_size            2048
read_buffer_size                  131072
read_only                         FALSE
read_rnd_buffer_size              262144
div_precision_increment           4
record_buffer                     131072
relay_log_purge                   TRUE
relay_log_space_limit             0
slave_compressed_protocol         FALSE
slave_net_timeout                 3600
slave_transaction_retries         10
slow_launch_time                  2
sort_buffer_size                  2097144
sync-binlog                       0
sync-frm                          TRUE
sync-replication                  0
sync-replication-slave-id         0
sync-replication-timeout          10
table_cache                       64
thread_cache_size                 0
thread_concurrency                10
thread_stack                      196608
time_format                       (No default value)
tmp_table_size                    33554432
transaction_alloc_block_size      8192
transaction_prealloc_size         4096
updatable_views_with_limit        1
wait_timeout                      28800
```

If there is a `mysqld` server currently running, you can see what values it actually is using for the system variables by connecting to it and issuing this statement:

```
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
```

You can find a full description for all system and status variables in Sección 5.3.3, “Variables de sistema del servidor” and Sección 5.3.4, “Variables de estado del servidor”.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by 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 256MB of memory and many tables and want maximum performance with a moderate number of clients, you should use something like this:

```
shell> mysqld_safe --key_buffer_size=64M --table_cache=256 \
           --sort_buffer_size=4M --read_buffer_size=1M &
```

- If you have only 128MB 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=16M --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 doing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, you should increase the value of `read_rnd_buffer_size` to speed up the reading of rows after sorting operations.

When you have installed MySQL, the `support-files` directory contains some different `my.cnf` sample files: `my-huge.cnf`, `my-large.cnf`, `my-medium.cnf`, and `my-small.cnf`. You can use these as a basis for optimizing your system.

Note that 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=32M --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 Sección 15.11, “Consejos de afinamiento del rendimiento de `InnoDB`”.

7.5.3. Vigilar el rendimiento del optimizador de consultas

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 7-10) this is not a problem. However, when bigger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server's performance.

MySQL 5.0.1 introduces a more flexible method for query optimization that allows 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 via 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, then this option can be switched off (`optimizer_prune_level`=0) with the risk that query compilation may take much longer. Notice 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 in order 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 compiler 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.

7.5.4. Efectos de la compilación y del enlace en la velocidad de MySQL

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`.

On Linux, it is best to compile the server with `pgcc` and `-O3`. You need about 200MB memory to compile `sql_yacc.cc` with these options, because `gcc` or `pgcc` needs a great deal of memory to make all functions inline. You should also set `CXX=gcc` when configuring MySQL to avoid inclusion of the `libstdc++` library, which is not needed. Note that with some versions of `pgcc`, the resulting binary runs only on true Pentium processors, even if you use the compiler option indicating that you want the resulting code to work on all x586-type processors (such as AMD).

By using a better compiler and compilation options, you can obtain a 10-30% 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 allow 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 use `pgcc` and compile everything with `-O6`, the `mysqld` server is 1% faster than with `gcc` 2.95.2.
- 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.
- If you strip your `mysqld` binary with `strip mysqld`, the resulting binary can be up to 4% faster.
- 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 hostname `localhost`, MySQL uses a socket file by default.)
- For TCP/IP connections from a client to a server, connecting to a remote server on another host is 8-11% slower than connecting to a server on the same host, even for connections over 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 options `-mcpu=v8 -Wa,-xarch=v8plusa` gives 4% better performance.
- On Solaris 2.5.1, MIT-pthreads is 8-12% slower than Solaris native threads on a single processor. With more load or CPUs, the difference should be larger.
- Compiling on Linux-x86 using `gcc` without frame pointers (`-fomit-frame-pointer` or `-fomit-frame-pointer -ffixed-ebp`) makes `mysqld` 1-4% faster.

Binary MySQL distributions for Linux that are provided by MySQL AB used to be compiled with `pgcc`. We had to go back to regular `gcc` due to a bug in `pgcc` that would generate binaries that do not run on AMD. We will continue using `gcc` until that bug is resolved. In the meantime, if you have a non-AMD machine, you can build a faster binary by compiling with `pgcc`. The standard MySQL Linux binary is linked statically to make it faster and more portable.

7.5.5. Cómo utiliza MySQL la memoria

The following list indicates some of the ways that the `mysqld` server uses memory. Where applicable, the name of the system variable relevant to the memory use is given:

- The key buffer (variable `key_buffer_size`) is shared by all threads; other buffers used by the server are allocated as needed. See Sección 7.5.2, “Afinar parámetros del servidor”.
- Each connection uses some thread-specific space:
  - A stack (default 64KB, variable `thread_stack`)
  - A connection buffer (variable `net_buffer_length`)
  - A result buffer (variable `net_buffer_length`)

  The connection buffer and result buffer are dynamically enlarged up to `max_allowed_packet` when needed. While a query is running, a copy of the current query string is also allocated.
- All threads share the same base memory.
- 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 in order 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 (`HEAP`) tables. Temporary tables with a large record length (calculated as the sum of all column lengths) or that contain `BLOB` columns are stored on disk.

  If an internal heap table exceeds the size of `tmp_table_size`, MySQL 5.0 handles this automatically by changing the in-memory heap table to a disk-based `MyISAM` table as necessary. You can also increase the temporary table size by setting the `tmp_table_size` option to `mysqld`, or by setting the SQL option `SQL_BIG_TABLES` in the client program. See Sección 13.5.3, “Sintaxis de SET”.
- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Sección A.4.4, “Dónde almacena MySQL los archivos temporales”.
- Almost all parsing and calculating is done in a local memory store. 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; this is done with `malloc()` and `free()`.
- 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.
- 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.
- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. By default, the cache has 64 entries. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See Sección 7.4.8, “Cómo abre y cierra tablas MySQL”.

- 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.

`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. You can verify this by checking 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.

7.5.6. Cómo usa MySQL las DNS

When a new client connects to `mysqld`, `mysqld` spawns a new thread to handle the request. This thread first checks whether the hostname is in the hostname cache. If not, the thread attempts to resolve the hostname:

- If the operating system supports the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls, the thread uses them to perform hostname resolution.
- If the operating system doesn't support the thread-safe calls, the thread locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no other thread can resolve hostnames that are not in the hostname cache until the first thread unlocks the mutex.

You can disable DNS hostname lookups by starting `mysqld` with the `--skip-name-resolve` option. However, in this case, you can use only IP numbers in the MySQL grant tables.

If you have a very slow DNS and many hosts, you can get more performance by either disabling DNS lookups with `--skip-name-resolve` or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling `mysqld`.

You can disable the hostname cache by starting the server with the `--skip-host-cache` option. To clear the hostname cache, issue a `FLUSH HOSTS` statement or execute the `mysqladmin flush-hosts` command.

If you want to disallow TCP/IP connections entirely, start `mysqld` with the `--skip-networking` option.

7.6. Cuestiones relacionadas con el disco

- Disk seeks are a major 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/or data file 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 Sección 7.6.1, “Utilizar enlaces simbólicos”.

  - 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 Nth 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 Sección 7.1.5, “Usar pruebas de rendimiento (benchmarks) propios”.

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 more 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:

```
hdparm -m 16 -d 1
```

Note that 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, filesystem corruption may result, so back up everything before experimenting!

- You can also set the parameters for the filesystem that the database uses:

If you don't need to know when files were last accessed (which is not really useful on a database server), you can mount your filesystems with the `-o noatime` option. That skips updates to the last access time in inodes on the filesystem, which avoids some disk seeks.

On many operating systems, you can set a filesystem to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you more performance without sacrificing too much reliability. (This flag is on by default on Linux.)

7.6.1. Utilizar enlaces simbólicos

You can move tables and databases 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 disk.

The recommended way to do this is simply to symlink databases to a different disk. Symlink tables only as a last resort.

7.6.1.1. Utilización de enlaces simbólicos para bases de datos en 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 symlink 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
```

For 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 result.

However, if you really need to do this, it is possible by altering the source file `mysys/my_symlink.c`, in which you should look for the following statement:

```
if (!(MyFlags & MY_RESOLVE_LINK) ||
    (!lstat(filename,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Change the statement to this:

```
if (1)
```

Note that, in MySQL 5.0, symbolic link support is enabled by default for all Windows servers.

7.6.1.2. Utilización de enlaces simbólicos para tablas en Unix

You should not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). You can check whether your system supports symbolic links by issuing a `SHOW VARIABLES LIKE 'have_symlink'` statement.

In MySQL 5.0, symlinks are fully supported only for `MyISAM` tables. For other table types, you may get strange problems if you try to use symbolic links on files in the operating system with any of the preceding statements.

The handling of symbolic links for `MyISAM` tables in MySQL 5.0 works the following way:

- In the data directory, you always have the table definition file, the data file, and the index file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The definition file cannot.
- You can symlink the data file and the index file independently to different directories.
- Symlinking can be accomplished manually from the command line using `ln -s` if `mysqld` is not running. Aternativly, you can instruct a running MySQL server to perform the symlinking by using the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See Sección 13.1.5, “Sintaxis de CREATE TABLE”.
- `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.
- **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 why you should *not* run `mysqld` as the system `root` or allow system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` 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` to move a table to another database, the table is moved to the other database directory. The old symlinks and the files to which they pointed are deleted. In other words, the new table is not symlinked.
- If you are not using symlinks, you should use the `--skip-symbolic-links` option to `mysqld` to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

Table symlink operations that are not yet supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.
- `BACKUP TABLE` and `RESTORE TABLE` don't respect symbolic links.
- The `.frm` file must *never* be a symbolic link (as indicated previously, only the data and index files can be symbolic links). Attempting to do this (for example, to make synonyms) 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`:

```
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` also fail.

7.6.1.3. Usar enlaces simbólicos para bases de datos en Windows

The `mysqld-max` and `mysql-max-nt` servers for Windows are compiled with the `-DUSE_SYMDIR` option. This allows you to put a database directory on a different disk by setting up a symbolic link to it. This is similar to the way that symbolic links work on Unix, although the procedure for setting up the link is different.

In MySQL 5.0, symbolic links are enabled by default. If you do not need them, you can disable them with the `skip-symbolic-links` option:

```
[mysqld]
skip-symbolic-links
```

On Windows, you create a symbolic link to a MySQL database by creating a 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.

Suppose that the MySQL data directory is `C:\mysql\data` and you want to have database `foo` located at `D:\data\foo`. Set up a symlink as shown here:

1. Make sure that the `D:\data\foo` directory exists by creating it if necessary. If you have a database directory named `foo` in the data directory, you should move it to `D:\data`. Otherwise, the symbolic

link is ineffective. To avoid problems, the server should not be running when you move the database directory.

2. Create a text file `C:\mysql\data\foo.sym` that contains the pathname `D:\data\foo\`.

After this, all tables created in the database `foo` are created in `D:\data\foo`. Note that the symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.

Capítulo 8. Programas cliente y utilidades MySQL

Tabla de contenidos

Hay muchos programas clientes MySQL distintos que conectan con el servidor para acceder a la base de datos o realizar tareas administrativas. Existen también otras utilidades, que no comunican con el servidor, pero que realizan operaciones relacionadas con MySQL.

Este capítulo proporciona un breve resumen de estos programas y luego una descripción más detallada de cada uno. La descripción indica cómo invocar cada programa y las opciones que entiende. Consulte Capítulo 4, *Usar los programas MySQL* para información general sobre la invocación de los programas y sus opciones específicas.

8.1. Panorámica de scripts y utilidades del lado del cliente

La siguiente lista describe brevemente los programas clientes y las utilidades MySQL:

- `myisampack`

  Utilidad que comprime tablas `MyISAM` para producir tablas más pequeñas de sólo lectura. Consulte Sección 8.2, “`myisampack`, el generador de tablas comprimidas de sólo lectura de MySQL”.

- `mysql`

  La herramienta de línea de comando para introducir comandos SQL interactivamente o ejecutarlos desde un fichero en modo batch. Consulte Sección 8.3, “La herramienta intérprete de comandos `mysql`”.

- `mysqlaccess`

  Script que verifica los permisos de acceso para una combinación de máquina, usuario y base de datos.

- `mysqladmin`

  Cliente que realiza tareas administrativas, tales como crear y borrar bases de datos, recargar las tablas de permisos, volcar tablas a disco y reabrir ficheros de log. `mysqladmin` también puede utilizarse para consultar la versión, información de procesos, e información de estado del servidor. Consulte Sección 8.4, “Administrar un servidor MySQL con `mysqladmin`”.

- `mysqlbinlog`

  Utilidad para leer comados de un log binario. El log de comandos ejecutados contenidos en el fichero de log binario puede utilizarse para ayudar en la recuperación de una fallida. Consulte Sección 8.5, “La utilidad `mysqlbinlog` para registros binarios”.

- MySQL AB proporciona tres programas con interfícies gráficas para usar con el servidor MySQL:

  - MySQL Administrator: Esta herramienta se usa para administrar servidores MySQL, bases de datos, tablas y usuarios.

  - MySQL Query Browser: MySQL AB proporciona esta herramienta gráfica para crear, ejecutar y optimizar consultas en bases de datos MySQL.

  - MySQL Migration Toolkit: Esta herramienta ayuda en la migración de esquemas y datos de otros sistemas de gestión de bases de datos realcionales a MySQL.

-

- `mysqlcheck`

  Cliente de mantenimiento de tablas que verifica, repara, analiza y optimiza tablas. Consulte Sección 8.6, “El programa `mysqlcheck` para mantener y reparar tablas”.

- `mysqldump`

  Cliente que vuelca una base de datos MySQL en un fichero como comandos SQL o como ficheros separados por tabuladores. Originalmente fue creado como freeware por Igor Romanenko. Consulte Sección 8.7, “El programa de copia de seguridad de base de datos `mysqldump`”.

- `mysqlhotcopy`

  Utilidad que realiza copias de seguridad rápidas de tablas `MyISAM` o `ISAM` mientras el servidor está en ejecución .Consulte Sección 8.8, “El programa de copias de seguridad de base de datos `mysqlhotcopy`”.

- `mysqlimport`

Cliente que importa ficheros de texto en sus respectivas tablas usando `LOAD DATA INFILE`. Consulte Sección 8.9, “El programa para importar datos `mysqlimport`”.

- `mysqlshow`

  Cliente que muestra información de bases de datos, tablas, columnas, e índices. Consulte Sección 8.10, “Mostrar bases de datos, tablas y columnas con `mysqlshow`”.

- `perror`

  Utilidad que muestra el significado de los errores de sistema de MySQL. Consulte Sección 8.11, “`perror`, explicación de códigos de error”.

- `replace`

  Programa que cambia cadenas de caracteres en ficheros o en la entrada estándar. Consulte Sección 8.12, “La utilidad `replace` de cambio de cadenas de caracteres”.

Cada programa MySQL tienen varias opciones, pero todos ellos proporcionan una opción `--help` que puede utilizarse para obtener una descripción completa de las distintas opciones del programa. Por ejemplo, `mysql --help`.

Los clientes MySQL que comunican con el servidor usando la biblioteca `mysqlclient` usan las siguientes variables de entorno:

`MYSQL_UNIX_PORT`	El fichero socket de Unix; se utiliza para conexiones al `localhost`
`MYSQL_TCP_PORT`	Puerto por defecto; se utiliza para conexiones TCP/IP
`MYSQL_PWD`	Contraseña por defecto
`MYSQL_DEBUG`	Opciones de traza para depurar código
`TMPDIR`	Directorio donde se crean las tablas y ficheros temporales

El uso de `MYSQL_PWD` no es seguro. Consulte Sección 5.7.6, “Guardar una contraseña de forma segura”.

Puede cambiar las opciones por defecto o valores especificados en las variables de entorno para todos los programas estándar, especificando las opciones en un fichero de opciones o en la línea de comandos. Sección 4.3, “Especificar opciones de programa”.

8.2. `myisampack`, el generador de tablas comprimidas de sólo lectura de MySQL

La utilidad `myisampack` comprime tablas `MyISAM` . `myisampack` funciona comprimiendo cada columna de la tabla separadamente. Normalmente, `myisampack` comprime el fichero de datos un 40%-70%.

Cuando la tabla se utiliza posteriormente, el servidor lee en la memoria la información que se necesita para descomprimir las columnas. Esto da un rendimiento mucho mejor al acceder a registros individuales, ya que sólo se tiene que descomprimir un registro.

Siempre que es posible, MySQL utiliza `mmap()` para realizar operaciones de mapeo en memoria de tablas comprimidas. Si `mmap()` no funciona, MySQL vuelve a operaciones normales de escritura/lectura en ficheros.

Tenga en cuenta lo siguiente:

- Si el servidor `mysqld` se invoca con la opción `--skip-external-locking`, no es buena idea invocar `myisampack` si la tabla tiene que actualizarse durante el proceso de compresión.
- Tras su compresión, una tabla será de sólo lectura. Esto, normalmente, es lo deseable (como cuando se accede a tablas comprimidas en un CD). Permitir escrituras en tablas comprimidas está en nuestra lista de cosas pendientes, pero con poca prioridad.
- `myisampack` comprime columnas `BLOB` o `TEXT` . El antiguo programa `pack_isam` para tablas `ISAM` no puede.

Invoque `myisampack` de la siguiente forma:

```
shell> myisampack [opciones] nombre_fichero ...
```

Cada nombre de fichero debe ser el nombre de un fichero índice (`.MYI`) . Si no se encuentra en el directorio de la base de datos, debe especificar la ruta al fichero. Se permite omitir la extensión `.MYI` .

`myisampack` soporta las siguientes opciones:

- `--help`, `-?`

  Muestra mensaje de ayuda y acaba.

- `--backup`, `-b`

  Realiza una copia de seguridad de los datos de la tabla usando el nombre *`nombre_tabla`*`.OLD`.

- `--debug[=`*`opciones_de_depuración`*`]`, `-# [`*`opciones_de_depuración`*`]`

  Escribe un log de depuración. La cadena de caracteres *`opciones_de_depuración`* a menudo es `'d:t:o,`*`nombre_fichero`*`'`.

- `--force`, `-f`

  Crea una tabla comprimida incluso si llega a ser mayor que la original o si el fichero temporal de una invocación anterior de `myisampack` ya existe. (`myisampack` crea un fichero temporal llamado *`nombre_tabla`*`.TMD` cuando comprime la tabla. Si se mata `myisampack`, el fichero `.TMD` puede no borrarse.) Normalmente, `myisampack` acaba con un error si encuentra que *`nombre_tabla`*`.TMD` existe. Con `--force`, se fuerza `myisampack` a comprimir la tabla.

- `--join=`*`nombre_de_tabla_grande`*, `-j` *`nombre_de_tabla_grande`*

  Junta todas las tablas nombradas en la línea de comandos en una única tabla *`nombre_de_tabla_grande`*. Todas las tablas que hay que combinar *deben* tener una estructura idéntica (los mismos nombres de columna y tipos, los mismos índices, etc).

- `--packlength=#`, `-p #`

  Especifica el tamaño de almacenamiento del registro, en bytes. El valor debe ser 1, 2, o 3. `myisampack` almacena todos los registros con punteros de 1, 2, o 3 bytes. En la mayoría de casos normales, `myisampack` puede determinar el valor de longitud óptimo antes de empezar a comprimir el fichero, pero puede darse cuenta durante el proceso de compresión de que podría haber usado una longitud

inferior. En ese caso, myisampack muestra una nota diciendo que en la siguente compresión del mismo fichero, puede usarse una longitud de fichero inferior.

- --silent, -s

  Modo silencioso. Sólo muestra mensajes de error.

- --test, -t

  No comprime la tabla, sólo prueba la compresión.

- --tmpdir=*ruta*, -T *ruta*

  myisampack crea en ese directorio los ficheros temporales.

- --verbose, -v

  Modo explícito. Muestra información sobre el proceso de compresión y su resultado.

- --version, -V

  Muestra información de la versión y sale.

- --wait, -w

  Espera y vuelve a empezar si la tabla está en uso. Si el servidor mysqld se invocó con la opción --skip-external-locking, no es buena idea invocar myisampack si existe la posibilidad que el servidor actualice la tabla durante el proceso de compresión.

La siguiente secuencia de comandos ilustra una típica sesión de compresión de tablas:

```
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
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> 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
```

```
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
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` muestra la siguiente información:

- `normal`

  Número de columnas para el que no se necesita compresión extra.

- `empty-space`

  Número de columnas que contienen valores que son sólo espacios, que ocupan un bit.

- `empty-zero`

  Número de columnas que contienen valores que son sólo ceros binarios; ocupan un bit.

- `empty-fill`

  Número de columnas de tipo entero que no ocupan el rango completo de bytes de su tipo; se cambian a un tipo inferior. Por ejemplo, una columna `BIGINT` (ocho bytes) puede almacenarse como una columna `TINYINT` (un byte) si todos sus valores están en el rango de `-128` a `127`.

- `pre-space`

  Número de columnas decimales que se almacenan con espacios iniciales. En este caso, cada valor contiene un contador para el número de espacios precedentes.

- `end-space`

  Número de columnas que tienen muchos espacios al final. En este caso, cada valor contiene un contador para el número de espacios al final.

- `table-lookup`

  La columna tiene pocos valores distintos, que se convierten a `ENUM` antes de la compresión Huffman.

- `zero`

  Número de columnas cuyos valores son todos cero.

- `Original trees`

  Número inicial de árboles Huffman.

- `After join`

  Número de árboles Huffman distintos que quedan tras unir árboles para tener algún espacio de cabecera.

Tras comprimir una tabla, `myisamchk -dvv` muestra información adicional acerca de cada columna:

- `Type`

El tipo de la columna. El valor puede contener cualquiera de los siguientes descriptores:

- constant

  Todos los registros tienen el mismo valor.

- no endspace

  No guarda el espacio final.

- no endspace, not_always

  No guarda el espacio final y no realiza compresión en caso de que haya espacio final en todos los valores.

- no endspace, no empty

  No guarda el espacio final. No guarda valores vacíos.

- table-lookup

  La columna se convirtió a ENUM.

- zerofill(*n*)

  Los *n* bytes más significativos en el valor son siempre 0 y no se guardan.

- no zeros

  No almacena ceros.

- always zero

  Los valores cero se guardan usando un bit.

- Huff tree

  Número de árboles Huffman asociados con cada columna.

- Bits

  Número de bits usados en el árbol Huffman.

Tras ejecutar myisampack, es necesario lanzar myisamchk para volver a crear algunos índices. Al mismo tiempo, se puede ordenar los bloques de índices y crear estadísticas, que el optimizador MySQL necesita para trabajar de forma más eficiente:

```
shell> myisamchk -rq --sort-index --analyze nombre_de_tabla.MYI
```

Una vez instalada la tabla comprimida en el directorio de la base de datos MySQL, debe ejecutar mysqladmin flush-tables para forzar mysqld a comenzar a usar la nueva tabla.

Para descomprimir una tabla comprimida, use la opción --unpack con myisamchk o isamchk.

8.3. La herramienta intérprete de comandos mysql

mysql es un simple shell SQL (con capacidades GNU readline). Soporta uso interactivo y no interactivo. Cuando se usa interactivamente, los resultados de las consultas se muestran en formato de tabla ASCII. Cuando se usa no interactivamente (por ejemplo, como filtro), el resultado se presenta en formato separado por tabuladores. El formato de salida puede cambiarse usando opciones de línea de comandos.

Si tiene problemas relacionados con memoria insuficiente para conjuntos de resultados grandes, utilice la opción --quick. Esto fuerza mysql a devolver los resultados desde el servidor registro a registro en lugar de recibir todo el conjunto de resultados y almacenarlo en memoria antes de mostrarlo. Esto se hace usando mysql_use_result() en lugar de mysql_store_result() para recibir el conjunto de resultados.

Usar mysql es muy sencillo. Invóquelo desde el prompt de su intérprete de comandos como se muestra a continuación:

```
shell> mysql nombre_base_de_datos
```

O:

```
shell> mysql --user=nombre_usuario --password=contraseña nombre_base_de_datos
```

Cuando introduzca un comando SQL, acábelo con ';', \g, o \G y pulse Enter.

Puede ejecutar un script simplemente así:

```
shell> mysql nombre_base_de_datos < script.sql > output.tab
```

mysql soporta las siguientes opciones:

- --help, -?

  Muestra el mensaje de ayuda y sale.

- --batch, -B

  Muestra los resultados usando tabuladores como separadores de columnas, con cada registro en una nueva línea. Con esta opción mysql no usa el fichero de historial.

- --character-sets-dir=*ruta*

  Directorio donde los conjuntos de caracteres están instalados. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- --compress, -C

  Comprime toda la información enviada entre el cliente y el servidor si ambos soportan compresión.

- --database=*nombre_base_de_datos*, -D *nombre_base_de_datos*

  La base de datos a usar. Esto es útil sobretodo en un fichero de opciones.

- --debug[=*opciones_de_depuración*], -# [*opciones_de_depuración*]

  Escribe un log de depuración. La cadena de caracteres *opciones_de_depuración* a menudo es 'd:t:o,*nombre_de_fichero*'. Por defecto es 'd:t:o,/tmp/mysql.trace'.

- --debug-info, -T

  Muestra alguna información de depuración al salir del programa.

- --default-character-set=*conjunto de caracteres*

  Usa *conjunto de caracteres* como conjunto de caracteres por defecto. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- --execute=*comando*, -e *comando*

  Ejecuta el comando y sale. El formato de salida por defecto es como el producido con --batch. Consulte Sección 4.3.1, “Usar opciones en la línea de comandos” para algunos ejemplos.

- --force, -f

  Continua incluso si ocurre un error SQL.

- --host=*nombre_equipo*, -h *nombre_equipo*

  Conecta al servidor MySQL en el equipo especificado.

- --html, -H

  Produce la salida en HTML.

- --ignore-space, -i

  Ignora espacios tras los nombres de funciones. El efecto de esta opción se describe en la discusión para IGNORE_SPACE en Sección 5.3.2, “El modo SQL del servidor”.

- --local-infile[={0|1}]

Activa o desactiva la función LOCAL para LOAD DATA INFILE. Sin valor, la opción activa LOCAL. Puede darse como --local-infile=0 o --local-infile=1 para desactivar explícitamente o activar LOCAL. Activar LOCAL no tiene efecto si el servidor no lo soporta.

- --named-commands, -G

  Los comandos con nombre están *activados* por defecto. Se permite comandos con formato largo así como comandos abreviados con *. Por ejemplo, se reconoce tanto quit como \q .

- --no-auto-rehash, -A

  No hay un recálculo del hash (rehashing) automático. Esta opción hace que mysql arranque más rápido, pero debe realizar el comando rehash si quiere usar el completado automático de tabla y columna.

- --no-beep, -b

  No hace ningún sonido cuando ocurre un error.

- --no-named-commands, -g

  Desactiva los comandos con nombre. Use la forma * únicamente, o use comandos con nombre sólo al comienzo de una línea, acabando con punto y coma (';'). Desde MySQL 3.23.22, mysql arranca con esta opción *activada* por defecto. De todos modos, incluso con esta opción, los comandos con formato largo funcionan desde la primera línea.

- --no-pager

  No usa paginación para mostrar la salida de una consulta. La paginación en la salida se discute más en Sección 8.3.1, “Comandos mysql”.

- --no-tee

  No copia la salida en un fichero. Los ficheros tee se discuten más en Sección 8.3.1, “Comandos mysql”.

- --one-database, -O

  Ignora comandos excepto aquéllos para la base de datos por defecto, nombrada en la línea de comandos. Esto es útil para evitar actualizaciones de otras bases de datos en el log binario.

- --pager[=*comando*]

Este comando se utiliza para paginar la salida de una consulta. Si el comando se omite, el paginador por defecto es el valor de la variable de entorno PAGER. Paginadores válidos son less, more, cat [> nombre_de_fichero], etcétera. Esta opción sólo funciona en Unix. No funciona en modo batch. La paginación de la salida de datos (output) se discute también en Sección 8.3.1, “Comandos mysql”.

- --password[=*contraseña*], -p[*contraseña*]

  La contraseña a usar al conectar al servidor. Si se usa la forma de opción corta (-p), *no* se deben poner espacios entre la opción y la contraseña. Si se omite el valor de la contraseña, a continuación de --password o -p en la línea de comandos, aparece un prompt pidiéndola. La contraseña debe omitirse en sistemas Unix basados en SysV, ya que debe mostrarse en la salida de ps.

- --port=*puerto_num*, -P *puerto_num*

  El puerto TCP/IP a usar en la conexión.

- --prompt=*formato_cadena*

  Asigna al prompt el formato especificado. Por defecto es mysql>. Las secuencias especiales que soporta el prompt se describen en Sección 8.3.1, “Comandos mysql”.

- --protocol={TCP | SOCKET | PIPE | MEMORY}

  Protocolo de conexión a usar.

- --quick, -q

  No cachea cada resultado de consulta, muestra cada registro según va llegando. Esto puede ralentizar el servidor si la salida se suspende. Con esta opción, mysql no usa el fichero histórico.

- --raw, -r

  Muestra los valores de las columnas sin conversión de escape. Se usa a menudo con la opción --batch.

- --reconnect

  Si se pierde la conexión con el servidor, intenta reconectar automáticamente. Se intenta un único intento de reconexión cada vez que se pierde la conexión. Para eliminar el comportamiento de la reconexión, use --skip-reconnect.

- --safe-updates, --i-am-a-dummy, -U

Sólo permite aquellos comandos UPDATE y DELETE que especifica cuáles registros se deben modificar usando valores clave. Si tiene activada esta opción en un fichero de opciones, puede evitarla usando --safe-updates en la línea de comandos. Consulte Sección 8.3.3, “Sugerencias acerca de mysql” para más información sobre esta opción.

- --secure-auth

  No envía contraseñas al servidor en formato antiguo (pre-4.1.1). Esto evita realizar conexiones con servidores que no usen el formato de contraseña nuevo.

- --show-warnings

  Muestra los mensajes de advertencia tras cada comando, si es necesario. Esta opción se aplica en modo interactivo y batch. Esta opción se añadió en MySQL 5.0.6.

- --sigint-ignore

  Ignora señales SIGINT (típicamente el resultado de pulsar Control-C).

- --silent, -s

  Modo silencioso. Produce menos salida de datos (output). Esta opción puede utilizarse múltiples veces para producir cada vez menos salida de datos.

- --skip-column-names, -N

  No escribe nombres de columnas en los resultados.

- --skip-line-numbers, -L

  No escribe el número de línea en los errores. Es útil cuando se quiere comparar ficheros de resultados que incluyan mensajes de error.

- --socket=*ruta*, -S *ruta*

  Fichero socket a usar para la conexión.

- --table, -t

  Muestra la salida de datos (output) en formato de tabla. Éste es el modo de funcionamiento por defecto para modo interactivo, pero puede usarse para producir la salida de datos de la tabla en modo batch.

- --tee=*nombre_fichero*

  Añade una copia de la salida de datos (output) al fichero dado. Esta opción no funciona en modo batch. Más información sobre ficheros tee en Sección 8.3.1, “Comandos mysql”.

- --unbuffered, -n

  Vuelca el buffer después de cada consulta.

- --user=*nombre_usuario*, -u *nombre_usuario*

  El nombre de usuario MySQL usado al conectar al servidor.

- --verbose, -v

  Modo explícito. Produce más salida de datos (output) acerca de lo que hace el programa. Esta opción puede escribirse múltiples veces para producir más y más salida. (Por ejemplo, -v -v -v produce la salida de datos en formato de tabla incluso en modo batch.)

- --version, -V

  Muestra información de la versión y sale.

- --vertical, -E

  Muestra los registros de salida de la consulta verticalmente (una línea por cada valor). Sin esta opción, se puede solicitar este formato de salida de datos (output) para un comando escribiendo \G al final del mismo.

- --wait, -w

  Si la conexión no puede establecerse, espera y vuelve a intentarlo en lugar de abortar.

- --xml, -X

  Produce salida XML.

Puede asignar las siguientes variables usando las opciones --*nombre_variable*=*valor*:

- connect_timeout

  Número de segundos antes que la conexión dé un timeout. (El valor por defecto es 0).

- max_allowed_packet

  La máxima longitud del paquete para enviar al o recibir del servidor. (El valor por defecto es 16MB).

- max_join_size

  El límite automático de registros en un join al usar --safe-updates. (El valor por defecto es 1.000.000).

- net_buffer_length

  El tamaño de búfer para comunicaciones TPC/IP y socket. (El valor por defecto es 16KB).

- select_limit

  Límite automático para comandos SELECT al usar --safe-updates. (El valor por defecto es 1.000).

Es posible asignar valores a las variables usando la sintaxis --set-variable=*nombre_variable=valor* o -O *nombre_variable=valor*. En MySQL 5.0, esta sintaxis está obsoleta.

En Unix, el cliente mysql escribe un registro de los comandos ejecutados en un fichero histórico. Por defecto, el fichero histórico se llama .mysql_history y se crea en el directorio raíz del usuario. Para especificar un fichero diferente, cámbiese el valor de la variable de entorno MYSQL_HISTFILE.

Si no se quiere mantener el fichero histórico, primero hay que borrar .mysql_history si existe, y luego se utiliza alguna de las siguientes técnicas:

- Se cambia el valor de la variable MYSQL_HISTFILE a /dev/null. Para que este cambio tenga efecto cada vez que se entra, se pone la asignación de esta variable de entorno en uno de los ficheros de arranque de la shell del usuario.
- Se crea .mysql_history como enlace simbólico a /dev/null:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

  Sólo es necesario hacer esto una vez.

8.3.1. Comandos mysql

mysql envía comandos SQL al servidor para que sean ejecutados. También hay un conjunto de comandos que mysql interpreta por sí mismo. Para obtener una lista de estos comandos, se escribe help o \h en el prompt mysql>:

```
mysql> help

MySQL commands:
?         (\h)    Synonym for `help'.
clear     (\c)    Clear command.
connect   (\r)    Reconnect to the server.
                  Optional arguments are db and host.
delimiter (\d)    Set query 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.
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.
warnings  (\W)    Show warnings after every statement.
nowarning (\w)    Don't show warnings after every statement.
```

Cada comando tiene forma corta y larga. La forma larga no es sensible a mayúsculas; la forma corta lo es. La forma larga puede seguirse con un terminador de punto y coma, pero la forma corta no debería.

En el comando `delimiter`, debe evitar el uso del carácter antibarra ('`\`') ya que es un carácter de escape para MySQL.

Los comandos `edit`, `nopager`, `pager`, y `system` sólo funcionan en Unix.

El comando `status` proporciona alguna información sobre la conexión y el servidor que está usando. Si está ejecutando el modo `--safe-updates`, `status` también muestra los valores para las variables `mysql` que afectan a sus consultas.

Para loguear las consultas y su salida, use el comando `tee`. Todos los datos mostrados por pantalla se guardan en un fichero dado. Esto puede ser muy útil para depurar. Puede activar esta característica en la línea de comandos con la opción `--tee`, o interactivamente con el comando `tee`. El fichero `tee` puede desactivarse interactivamente con el comando `notee`. Ejecutando `tee` de nuevo, vuelve a activar el logueo. Sin parámetros, se utiliza el fichero anterior. Tenga en cuenta que `tee` vuelca los resultados de las consultas al fichero después de cada comando, justo antes que `mysql` muestre su siguiente prompt.

Navegar o buscar resultados de consultas en modo interactivo usando programas Unix, tales como `less`, `more`, o cualquier otro programa similar es posible con la opción `--pager`. Si no especifica ningún valor para la opción, `mysql` comprueba el valor de la variable de entorno `PAGER` y configura el paginador al valor de la misma. La paginación de la salida de datos (output) puede activarse interactivamente con el comando `pager` y desactivar con `nopager`. El comando tiene un argumento opcional; si se especifica, el programa de paginación se configura con el mismo. Sin argumento, el paginador se configura con el paginador que se especificó en la línea de comandos, o `stdout` si no se especifica paginador.

La paginación de la salida sólo funciona en Unix ya que usa la función `popen()`, que no existe en Windows. Para Windows, la opción `tee` puede usarse para guardar la salida de las consultas, aunque no es tan conveniente como `pager` para navegar por la salida en algunas situaciones.

Algunos consejos sobre el comando `pager`:

- Puede usarlo para escribir en un fichero y que los resultados sólo vayan al fichero:

  ```
  mysql> pager cat > /tmp/log.txt
  ```

  Puede pasar cualquier opción al programa que quiera usar como paginador:

  ```
  ```

```
mysql> pager less -n -i -S
```

- En el ejemplo precedente, tenga en cuenta la opción `-S`. Puede encontrarla muy útil para consultar una amplia variedad de resultados. En ocasiones un conjunto de resultados muy amplio es difícil de leer en pantalla. La opción `-S` con `less` puede hacer el resultado mucho más legible ya que puede desplazarlo horizontalmente usando las flechas de izquierda y derecha. También puede usar `-S` interactivamente con `less` para activar y desactivar el modo de navegación horizontal. Para más información, lea la página del manual (man) de `less`:

  ```
  shell> man less
  ```

- Puede especificar comandos muy complejos para el paginador, para tratar la salida de las consultas:

  ```
  mysql> pager cat | tee /dr1/tmp/res.txt \
            | tee /dr2/tmp/res2.txt | less -n -i -S
  ```

  En este ejemplo, el comando enviaría los resultados de las consultas a dos ficheros en dos directorios distintos en dos sistemas de ficheros montados en `/dr1` y `/dr2`,y mostraría los resultados en pantalla vía `less`.

Puede combinar las funciones `tee` y `pager`. Puede tener un fichero `tee` activado y tener el `pager` configurado con `less`, siendo capaz de navegar en los resultados con el programa `less` y al mismo tiempo tener todo en un fichero. La diferencia entre `tee` en Unix usado con el comando `pager` y el comando `tee` que implementa `mysql` es que el segundo `tee` funciona incluso si no tiene el `tee` de Unix disponible. El comando `tee` implementado también loguea todo lo que se muestra por pantalla, mientras que `tee` en Unix usado con `pager` no loguea tanto. Adicionalmente, el logueo en fichero de `tee` puede activarse y desactivarse interactivamente desde `mysql`. Esto es útil cuando quiere loguear algunas consultas en un fichero, pero no otras.

En MySQL 5.0, el prompt por defecto `mysql>` puede reconfigurarse. La cadena de caracteres para definir el prompt puede contener las siguientes secuencias especiales:

Opción	Descripción
`\v`	Versión del servidor
`\d`	Base de datos actual
`\h`	Equipo del servidor
`\p`	Puerto TCP/IP usado o fichero socket
`\u`	Nombre de usuario
`\U`	Nombre de cuenta *`nombre_usuario@nombre_equipo`* completo
`\\`	Carácter de barra invertida '`\`' literal
`\n`	Carácter de nueva línea
`\t`	Carácter de tabulador
`\ `	Espacio (hay un espacio a continuación de la barra invertida)
`\_`	Espacio
`\R`	Hora actual, en formato militar de 24 horas (0-23)
`\r`	Hora actual, en formato estándar de 12 horas (1-12)
`\m`	Minutos de la hora actual
`\y`	Año actual, dos dígitos

\Y	Año actual, cuatro dígitos
\D	Fecha actual completa
\s	Segundos de la hora actual
\w	Día actual de la semana en inglés, en formato de tres letras (Mon, Tue, ...)
\P	am/pm
\o	Mes actual en formato numérico
\O	Mes actual en inglés, en formato de tres letras (Jan, Feb, ...)
\c	Contador que incrementa con cada comando ejectuado
\S	Punto y coma
\'	Comilla
\"	Comilla doble

'\' seguido por cualquier otra letra es esa letra.

Si especifica el comando prompt sin argumentos, mysql resetea el prompt al valor por defecto de mysql>.

Puede cambiar el prompt de diversas formas:

- Use una variable de entorno

  Puede cambiar la variable de entorno MYSQL_PS1 con una cadena de caracteres. Por ejemplo:

  ```
  shell> export MYSQL_PS1="(\u@\h) [\d]> "
  ```

- Con un fichero de opciones

  Puede cambiar la opción prompt en el grupo [mysql] de cualquier fichero de opciones MySQL, tal como /etc/my.cnf o el fichero .my.cnf en el directorio raíz. Por ejemplo:

  ```
  [mysql]
  prompt=(\\u@\\h) [\\d]>\\_
  ```

  En este ejemplo, tenga en cuenta que las barras invertidas son dos. Si asigna el valor del prompt usando la opción prompt en un fichero de opciones, se recomienda poner dos barras invertidas al usar las opciones especiales de prompt. Hay algúna redundancia en el conjunto de opciones de prompt disponible y el conjunto de secuencias de caracteres de escape especiales que se reconoce en el fichero de opciones. (Estas secuencias están listadas en Sección 4.3.2, “Usar ficheros de opciones”). La redundancia puede causar problemas si usa sólo una barra invertidas. Por ejemplo, \s se interpreta como un espacio en lugar de los segundos actuales. El siguiente ejemplo muestra cómo definir un prompt dentro de un fichero de opciones que incluya la hora actual en formato HH:MM:SS>:

  ```
  [mysql]
  prompt="\\r:\\m:\\s> "
  ```

- Use una opción por línea de comandos

  Puede cambiar la opción --prompt en la línea de comandos de mysql. Por ejemplo:

  ```
  shell> mysql --prompt="(\u@\h) [\d]> "
  ```

```
(user@host) [database]>
```

- Interactivamente

  Puede cambiar el prompt interactivamente usando el comando `prompt` (o `\R`) . Por ejemplo:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(usuario@equipo) [base_de_datos]>
(usuario@equipo) [base_de_datos]> prompt
Returning to default PROMPT of mysql>
mysql>
```

8.3.2. Ejecutar sentencias SQL desde un fichero de texto

El cliente `mysql` se usa normalmente de forma interactiva, como se muestra a continuación:

```
shell> mysql nombre_base_de_datos
```

Sin embargo, es posible poner comandos SQL en un fichero y decirle a `mysql` que lea las entradas de ese fichero. Para ello, cree un fichero `fichero_de_texto` que contenga el comando que quiera ejecutar. Luego invoque `mysql` como se muestra aquí:

```
shell> mysql nombre_base_de_datos < fichero_de_texto
```

Puede comenzar su fichero de texto con un comando `USE nombre_base_de_datos`. En este caso, no es necesario especificar el nombre de la base de datos en la línea de comandos:

```
shell> mysql < text_file
```

Si está ejecutando `mysql`, puede ejecutar un script SQL en un fichero usando el comando `source` o `\.` :

```
mysql> source nombre_de_fichero
mysql> \. nombre_de_fichero
```

Si desea que el script muestre información de progreso al usuario, puede insertar algunas líneas como

```
SELECT '<información>' AS ' ';
```

que muestra <información>.

Para más información acerca del modo batch, consulte Sección 3.5, “Usar `mysql` en modo batch”.

8.3.3. Sugerencias acerca de `mysql`

Esta sección describe algunas técnicas que pueden ayudarle a usar `mysql` de forma más efectiva.

8.3.3.1. Mostrar resultados de consultas verticalmente

Algunos resultados de consultas son mucho más legibles cuando se muestran verticalmente, en lugar del formato habitual de tabla horizontal. Las consultas pueden mostrarse verticalmente terminando las mismas con \G en lugar de un punto y coma. Por ejemplo, valores de texto largos que incluyan nuevas líneas, a menundo son más fáciles de leer en formato vertical:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 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)
```

8.3.3.2. Usar la opción `--safe-updates`

Una opción de arranque útil para principiantes es `--safe-updates` (o `--i-am-a-dummy`, que tiene el mismo efecto). Es útil para casos donde pueda haber ejecutado un comando `DELETE FROM` *`nombre_tabla`* pero olvidó la cláusula `WHERE`. Normalmente, ese comando borra todos los registros de la tabla. Con `--safe-updates`, puede borrar registros sólo especificando los valores clave que los identifican. Esto ayuda a prevenir accidentes.

Cuando usa la opción `--safe-updates`, `mysql` realiza los siguientes comandos al conectarse al servidor MySQL:

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=1000, SQL_MAX_JOIN_SIZE=1000000;
```

Consulte Sección 13.5.3, “Sintaxis de SET”.

El comando `SET` tiene el siguiente efecto:

- No está permitido ejecutar un comando `UPDATE` o `DELETE` a no ser que especifique una condición clave en la cláusula `WHERE` o proporcione una cláusula `LIMIT` (o ambas). Por ejemplo:

  ```
  UPDATE nombre_tabla SET columna_no_clave=# WHERE columna_clave=#;

  UPDATE nombre_tabla SET columna_no_clave=# LIMIT 1;
  ```

- Todos los resultados grandes de un `SELECT` se limitan automáticamente a 1.000 registros a no ser que el comando incluya una cláusula `LIMIT` .

- Se aborta comandos `SELECT` de múltiples tablas que probablemente necesiten examinar más de 1,000,000 de registros.

Para especificar límites distintos a los anteriores, puede cambiar los valores por defecto con las opciones `--select_limit` y `--max_join_size`:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

8.3.3.3. Deshabilitar las reconexiones automáticas de mysql

Si el cliente mysql pierde la conexión al enviar una consulta, inmediatamente y de forma automática trata de reconectar una vez con el servidor y envía la consulta de nuevo. Sin embargo, incluso si mysql tiene éxito al reconectar, la primera conexión ha terminado y todos los objetos de sesión previos y opciones se pierden: tablas temporales, modo "autocommit", y variables de usuario y de sesión. Este comportamiento puede ser problemático, como en el siguiente ejemplo, donde el servidor se apaga y reinicia sin conocimiento del usuario:

```
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)
```

La variable de usuario @a se ha perdido con la conexión, y tras la reconexión no está definida. Si es importante que mysql termine con un error cuando se pierda la conexión, puede arrancar el cliente mysql con la opción --skip-reconnect .

8.4. Administrar un servidor MySQL con mysqladmin

mysqladmin es un cliente para realizar operaciones administrativas. Se puede usar para comprobar la configuración y el estado actual del servidor, crear y borrar bases de datos, y con más finalidades.

Invoque mysqladmin así:

```
shell> mysqladmin [opciones] comando [opciones_de_comando] comando ...
```

mysqladmin soporta los siguientes comandos:

- create *nombre_base_de_datos*

  Crea una nueva base de datos llamada *nombre_base_de_datos*.

- debug

  Le dice al servidor que escriba información de depuración en el log de error.

- drop *nombre_base_de_datos*

  Borra la base de datos llamada *nombre_base_de_datos* y todas sus tablas.

- extended-status

  Muestra las variables de estado del servidor y sus valores.

- `flush-hosts`

  Vuelca toda la información en la caché del equipo.

- `flush-logs`

  Vuelca todos los logs.

- `flush-privileges`

  Recarga las tablas de permisos (lo mismo que `reload`).

- `flush-status`

  Limpia las variables de estado.

- `flush-tables`

  Vuelca todas las tablas.

- `flush-threads`

  Vuelca la caché de threads.

- `kill id,id,...`

  Mata los threads del servidor.

- `old-password` *`nueva_contraseña`*

  Es como el comando `password` pero guarda la contraseña usando el formato de hash antiguo (pre-4.1). Consulte Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1”.)

- `password` *`nueva_contraseña`*

  Introduce una nueva contraseña. Esto cambia la contraseña a `nueva_contraseña` para la cuenta que usa con `mysqladmin` para conectar con el servidor.

  Si *`nueva_contraseña`* contiene espacios u otros caracteres que son especiales para su intérprete de comandos, debe ponerla entre comillas. En Windows, asegúrese de usar comillas dobles en lugar de simples; comillas simples no se eliminan de la contraseña sino que se interpretan como parte del acontraseña. Por ejemplo:

  shell> mysqladmin password "mi nueva contraseña"

- `ping`

  Comprueba si el servidor está vivo. El estado retornado por `mysqladmin` es 0 si el servidor está en ejecución, 1 si no lo está. En MySQL 5.0, el estado es 0 incluso en caso de un error tal como `Access denied`, ya que esto significa que el servidor está en ejecución pero no ha admitido la conexión, lo que no es lo mismo que el servidor no esté en ejecución.

- `processlist`

  Muestra una lista de los threads activos del servidor. Esto es como la salida del comando `SHOW PROCESSLIST`. Si se da la opción `--verbose`, la salida es como la de `SHOW FULL PROCESSLIST`. (Consulte Sección 13.5.4.16, “Sintaxis de SHOW PROCESSLIST”.)

- `reload`

Recarga las tablas de permisos.

- refresh

  Vuelca todas las tablas y cierra y abre los ficheros de logs.

- shutdown

  Detiene el servidor.

- start-slave

  Comienza la replicación en un servidor esclavo.

- status

  Muestra un mensaje de estado corto del servidor.

- stop-slave

  Detiene la replicación en un servidor esclavo.

- variables

  Muestra las variables de sistema del servidor y sus valores.

- version

  Muestra información de la versión del servidor.

Todos los comandos pueden abreviarse a un prefijo único. Por ejemplo:

```
shell> mysqladmin proc stat
+----+------+-----------+----+---------+------+-------+------------------+
| Id | User | Host      | db | Command | Time | State | Info             |
+----+------+-----------+----+---------+------+-------+------------------+
| 51 | root | 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
Memory in use: 92M  Max memory used: 410M
```

El comando mysqladmin status muestra los siguientes valores:

- Uptime

  Número de segundos que MySQL server ha estado en ejecución.

- Threads

  Número de threads activos (clientes).

- Questions

Número de preguntas (consultas) de los clientes desde el arranque del servidor.

- `Slow queries`

  Número de consultas que han tardado más de `long_query_time` segundos. Consulte Sección 5.10.4, “El registro de consultas lentas (Slow Query Log)”.

- `Opens`

  Número de tablas que el servidor ha abierto.

- `Flush tables`

  Número de comandos `flush ...`, `refresh` y `reload` ejecutados.

- `Open tables`

  Número de tablas abiertas actualmente.

- `Memory in use`

  Cantidad de memoria reservada directamente por el código de `mysqld`. Este valor se muestra sólo cuando MySQL es ha compilado con `--with-debug=full`.

- `Maximum memory used`

  La cantidad máxima de memoria reservada directamente por el código de `mysqld`. Este valor se muestra sólo cuando MySQL se ha compilado con `--with-debug=full`.

Si ejecuta `mysqladmin shutdown` al conectar a un servidor local usando ficheros socket Unix, `mysqladmin` espera hasta que el fichero con el ID del proceso del servidor se haya borrado, para asegurar que el servidor se ha parado correctamente.

`mysqladmin` soporta las siguientes opciones:

- `--help`, `-?`

  Muestra un mensaje de ayuda y sale.

- `--character-sets-dir=`*`ruta`*

  Directorio donde están instalados los conjuntos de caracteres. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--compress`, `-C`

Comprime toda la información enviada entre el cliente y el servidor, si ambos soportan compresión.

- `--count=#`, `-c #`

  Número de iteraciones a realizar en la ejecución de comandos repetidos. Esto funciona sólo con `--sleep` (`-i`).

- `--debug[=opciones_de_depuración]`, `-# [opciones_de_depuración]`

  Escribe un log de depuración. La cadena de caracteres *`opciones_de_depuración`* a menudo es `'d:t:o,nombre_de_fichero'`. Por defecto es `'d:t:o,/tmp/mysqladmin.trace'`.

- `--default-character-set=conjunto_de_caracteres`

  Usa *`conjunto_de_caracteres`* como el conjunto de caracteres por defecto. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--force`, `-f`

  No pide confirmación para el comando `drop database`. Con comandos múltiples, continúa incluso si hay un error.

- `--host=nombre_de_equipo`, `-h nombre_de_equipo`

  Conecta con el servidor MySQL en un equipo dado.

- `--password[=contraseña]`, `-p[contraseña]`

  La contraseña a usar cuando conecta con el servidor. Si usa la forma corta de la opción (`-p`), *no* puede haber un espacio entre la opción y la contraseña. Si omite el valor *`contraseña`* siguiente a la opción `--password` o `-p` en la línea de comando, aparece un prompt pidiéndola.

- `--port=número_de_puerto`, `-P número_de_puerto`

  Puerto TCP/IP para usar en las conexiones.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  Protocolo de conexión en uso. Nuevo en MySQL 4.1.

- `--relative`, `-r`

  Muestra la diferencia entre los valores actuales y anteriores cuando se usa con `-i`. Actualmente, esta opción sólo funciona con el comando `extended-status`.

- `--silent`, `-s`

  Sale silenciosamente si no puede establecerse una conexión con el servidor.

- `--sleep=retraso`, `-i retraso`

  Ejecuta comandos una y otra vez, durmiendo durante *`retraso`* segundos entre ellos.

- `--socket=ruta`, `-S ruta`

  El fichero socket a usar en la conexión.

- `--user=nombre_de_usuario`, `-u nombre_de_usuario`

Nombre de usuario MySQL a usar al conectar con el servidor.

- `--verbose`, `-v`

  Modo explícito. Muestra más información sobre lo que hace el programa.

- `--version`, `-V`

  Muestra información sobre la versión y sale.

- `--vertical`, `-E`

  Muestra la salida (output) verticalmente. Es similar a `--relative`, pero la salida es vertical.

- `--wait[=#]`, `-w[#]`

  Si la conexión no puede establecerse, espera y vuelve a intentarlo en lugar de abortar. Si se da un valor de opción, indica el número de veces a reintentar. El valor por defecto es una vez.

Puede asignar valores a las siguientes variables usando las opciones *`--nombre_de_variable=valor`* :

- `connect_timeout`

  El número de segundos máximos antes que la conexión dé un timeout. El valor por defecto es 43200 (12 horas).

- `shutdown_timeout`

  El número máximo de segundos a esperar para la parada del servidor. El valor por defecto es 3600 (1 hora).

También es posible asignar valores a las variables usando la sintaxis `--set-variable=`*`nombre_de_variable=valor`* o `-O` *`nombre_de_variable=valor`*. Sin embargo, esta sintaxis está obsoleta desde MySQL 4.0.

8.5. La utilidad `mysqlbinlog` para registros binarios

Los ficheros de log binario que el servidor genera se escriben en formato binario. Para examinar estos ficheros en formato de texto, se utiliza la utilidad `mysqlbinlog` .

Invoque `mysqlbinlog` así:

```
shell> mysqlbinlog [opciones] fichero_de_log ...
```

Por ejemplo, para mostrar el contenido del log binario `binlog.000003`, use este comando:

```
shell> mysqlbinlog binlog.0000003
```

La salida incluye todos los comandos contenidos en `binlog.000003`, junto con otra información tal como el tiempo que ha tardado cada comando, el ID del thread del cliente que lo lanzó, la hora en que se ejecutó, etcétera.

Normalmente, `mysqlbinlog` se utiliza para leer ficheros de log binarios directamente y aplicarlos al servidor MySQL local. También es posible leer logs binarios de un servidor remoto usando la opción `--read-from-remote-server` .

Cuando se vaya a leer logs binarios remotos, deben darse los parámetros de conexión para indicar cómo conectar con el servidor, pero se ignoran a no ser que también se especifique la opción --read-from-remote-server . Estas opciones son --host, --password, --port, --protocol, --socket y --user.

También puede utilizarse mysqlbinlog para leer ficheros de logs de relay escritos por un servidor esclavo en el arranque de una replicación. Los logs de relay tienen el mismo formato que los ficheros de log binarios.

El log binario se discute en Sección 5.10.3, “El registro binario (Binary Log)”.

mysqlbinlog soporta las siguientes opciones:

- --help, -?

  Muestra un mensaje de ayuda y sale.

- --database=*nombre_de_base_de_datos*, -d *nombre_de_base_de_datos*

  Muestra las entradas sólo para esta base de datos (sólo log local)

- --force-read, -f

  Con esta opción, si mysqlbinlog lee de un log binario un evento que no reconoce, muestra una advertencia, ignora el comando, y continúa. Sin esta opción, mysqlbinlog se detiene si lee un evento que no reconoce.

- --host=*nombre_de_equipo*, -h *nombre_de_equipo*

  Obtiene el log binario del servidor MySQL del equipo dado.

- --local-load=*ruta*, -l *ruta*

  Prepara ficheros temporales locales para LOAD DATA INFILE en el directorio especificado.

- --offset=*N*, -o *N*

  Ignora las primeras *N* entradas.

- --password[=*contraseña*], -p[*contraseña*]

  La contraseña a usar cuando se conecta al servidor. Si usa la forma corta de opción (-p), *no* puede tener un espacio entre la opción y la contraseña. Si omite el valor *contraseña* siguiente a la opción --password o -p en la línea de comandos, aparece un prompt para que lo introduzca.

- --port=*número_de_puerto*, -P *número_de_puerto*

  El puerto TCP/IP a usar cuando conecta con un servidor remoto.

- --position=*N*, -j *N*

  Obsoleto, use --start-position.

- --protocol={TCP | SOCKET | PIPE | MEMORY}

  Protocolo de conexión a usar.

- --read-from-remote-server, -R

Lee el log binario de un servidor MySQL en vez de leer el local. Cualquier opción de parámetros de conexión se ignora a no ser que esta opción se dé también. Estas opciones son `--host`, `--password`, `--port`, `--protocol`, `--socket`, y `--user`.

- `--result-file=nombre`, `-r nombre`

  Salida directa al fichero dado.

- `--short-form`, `-s`

  Muestra sólo los comandos contenidos en el log, sin ninguna información extra.

- `--socket=ruta`, `-S ruta`

  Fichero socket a usar para la conexión.

- `--start-datetime=datetime`

  Comienza a leer el log binario en el primer evento que tenga una fecha igual o posterior a la del argumento `datetime` . El valor *`datetime`* es relativo a la zona horaria local en la máquina donde se ejecuta `mysqlbinlog`. El valor debe estar en un formato aceptado por los tipos de datos `DATETIME` o `TIMESTAMP` . Por ejemplo:

  ```
  shell> mysqlbinlog --start-datetime="2004-12-25 11:25:56" binlog.000003
  ```

  Esta opción es útil para recuperaciones en un punto temporal concreto.

- `--stop-datetime=datetime`

  Detiene la lectura del log binario en el primer evento que tenga una fecha igual o posterior a la del argumento `datetime`. Consulte la descripción de la opción `--start-datetime` para información acerca del valor de `datetime`.

- `--start-position=N`

  Comienza a leer el log binario en el primer evento que tenga una posición igual al argumento `N`.

- `--stop-position=N`

  Deja de leer el log binario en el primer evento que tenga una posición igual o superior al argumento `N`.

- `--to-last-log`, `-t`

  No se detiene al final del log binario solicitado del servidor MySQL, sino que sigue mostrando información hasta el final del último log binario. Si se envía la salida (output) al mismo servidor MySQL, esto puede conducir a un bucle infinito. Esta opción requiere `--read-from-remote-server`.

- `--disable-log-bin`, `-D`

  Desactiva el log binario. Esto es útil para eliminar un bucle infinito si se utiliza la opción `--to-last-log` y envía la salida (output) al mismo servidor MySQL. Esta opción también es útil cuando se restaura tras un fallo para evitar duplicación de comandos que se han logueado. **Nota**: Esta opción requiere privilegios `SUPER` .

- `--user=nombre_de_usuario`, `-u nombre_de_usuario`

  Nombre de usuario MySQL a usar cuando se conecta a un servidor remoto.

- --version, -V

  Muestra versión de información y sale.

Puede cambiar las siguientes variables con las opciones --*nombre_de_variable*=*valor*:

- open_files_limit

  Especifica el número de descriptores de ficheros abiertos a reservar.

Puede enviar la salida (output) de mysqlbinlog a un cliente mysql para que ejecute los comandos contenidos en el log binario. Esto se usa para recuperar de un fallo cuando tiene una copia de seguridad antigua (consulte Sección 5.8.1, “Copias de seguridad de bases de datos”):

```
shell> mysqlbinlog nombre_de_equipo-bin.000001 | mysql
```

O:

```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

También puede redirigir la salida (output) de mysqlbinlog a un fichero de texto, si necesita modificar el log de comandos primero (por ejemplo, para eliminar comandos que no quiere ejecutar por alguna razón). Tras editar el fichero, ejecute los comandos que contiene usándolo como entrada del programa mysql .

mysqlbinlog tiene la opción --start-position, que muestra sólo aquellos comandos con un desplazamiento en el log binario mayor o igual a una posición dada (la posición dada debe coincidir con el comienzo de un evento). También tiene las opciones de parar o arrancar cuando ve un evento de una fecha y hora dada. Esto permite ejecutar recuperaciones a partir de un punto temporal usando la opción --stop-datetime (que permite decir, por ejemplo, "devuelve mi base de datos al estado en que estaba hoy a las 10:30 AM").

Si tiene más de un log binario para ejecutar en el servidor MySQL, el método seguro es procesarlos todos usando una conexión única con el servidor. El siguiente ejemplo muestra una forma peligrosa de hacerlo:

```
shell> mysqlbinlog nombre_de_equipo-bin.000001 | mysql # PELIGRO!!
shell> mysqlbinlog nombre_de_equipo-bin.000002 | mysql # PELIGRO!!
```

Procesar logs binarios de esta manera, utilizando diferentes conexiones con el servidor provoca problemas si el primer fichero de log contiene un comando CREATE TEMPORARY TABLE y el segundo log contiene un comando que usa la tabla temporal. Cuando el primer proceso mysql termina, el servidor elimina la tabla temporal. Cuando el segundo proceso mysql trata de usar la tabla, el servidor devuelve “tabla desconocida”.

Para evitar problemas como éste, utilice una *única* conexión para ejecutar los contenidos de todos los logs binarios que quiere procesar. Sigue una forma de hacerlo:

```
shell> mysqlbinlog nombre_de_equipo-bin.000001 nombre_de_equipo-bin.000002 | mysql
```

Otra forma de hacerlo es escribir todos los logs a un único fichero y luego procesar el fichero:

```
shell> mysqlbinlog nombre_de_equipo-bin.000001 >  /tmp/statements.sql
shell> mysqlbinlog nombre_de_equipo-bin.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

En MySQL 5.0, mysqlbinlog puede producir la salida (output) que reproduce una operación LOAD DATA INFILE sin el fichero de datos original. mysqlbinlog copia los datos en un fichero temporal y escribe un comando LOAD DATA LOCAL INFILE que se refiere a este fichero. La localización por defecto del directorio donde estos ficheros se escriben es específico del sistema. Para espeficar un directorio explícitamente, use la opción --local-load.

Como mysqlbinlog convierte comandos LOAD DATA INFILE en comandos LOAD DATA LOCAL INFILE (esto es, añade LOCAL). Tanto el cliente como el servidor que se utiliza para procesar los comandos deben estar configurados para permitir LOCAL. Consulte Sección 5.5.4, “Cuestiones relacionadas con la seguridad y LOAD DATA LOCAL”.

Atención: Los ficheros temporales creados por comandos LOAD DATA LOCAL *no* se borran automáticamente ya que se necesitan hasta que los comandos se ejecutan totalmente. Debe borrar los ficheros temporales cuando no necesite el log de comandos. Los ficheros pueden encontrarse en el directorio de ficheros temporales y tienen nombres como original_file_name-#-#.

En el futuro, eliminaremos este problema permitiendo que mysqlbinlog pueda conectarse directamente a un servidor mysqld. Así será posible eliminar los ficheros de log automáticamente cuando los comandos LOAD DATA INFILE se hayan ejecutado.

8.6. El programa mysqlcheck para mantener y reparar tablas

El cliente mysqlcheck comprueba y repara tablas MyISAM. También puede optimizar y analizar tablas.

mysqlcheck es similar a myisamchk, pero funciona de forma distinta. La principal diferencia operacional es que mysqlcheck debe usarse cuando el servidor mysqld está en ejecución, mientras que myisamchk debe usarse cuando no lo está. El beneficio de usar mysqlcheck es que no tiene que parar el servidor para comprobar o reparar las tablas.

mysqlcheck usa los comandos SQL CHECK TABLE, REPAIR TABLE, ANALYZE TABLE, y OPTIMIZE TABLE de forma conveniente para los usuarios. Determina los comandos a usar en función de la operación que quiera realizar, luego envía los comandos al servidor para ejecutarlos.

Hay tres modos generales de invocar mysqlcheck:

```
shell> mysqlcheck [opciones] nombre_de_base_de_datos [tablas]
shell> mysqlcheck [opciones] --databases DB1 [DB2 DB3...]
shell> mysqlcheck [opciones] --all-databases
```

Si no nombra ninguna tabla o usa las opciones --databases o --all-databases, se comprueban todas las bases de datos.

mysqlcheck tiene una característica especial comparado con otros clientes. El comportamiento por defecto de comprobar tablas (--check) puede cambiarse renombrando el binario. Si quiere tener una herramienta que repare las tablas por defecto, debe hacer una copia de mysqlcheck llamada mysqlrepair, o hacer un enlace simbólico a mysqlcheck llamado mysqlrepair. Si invoca mysqlrepair, repara tablas.

Los siguientes nombres pueden usarse para cambiar el comportamiento por defecto de mysqlcheck :

mysqlrepair	La opción por defecto es --repair
mysqlanalyze	La opción por defecto es --analyze

mysqloptimize	La opción por defecto es --optimize

mysqlcheck soporta las siguientes opciones:

- --help, -?

  Muestra el mensaje de ayuda y sale.

- --all-databases, -A

  Comprueba todas las tablas en todas las bases de datos. Esto es lo mismo que usar la opción --databases y llamar todas las bases de datos en la línea de comandos.

- --all-in-1, -1

  En lugar de realizar un comando para cada tabla, ejecuta un único comando para cada base de datos, que nombra todas las tablas de la base de datos a procesar.

- --analyze, -a

  Analiza las tablas.

- --auto-repair

  Si una tabla comprobada está corrupta, la repara automáticamente. Cualquier reparación necesaria se hace tras el chequeo de cada tabla.

- --character-sets-dir=*path*

  El directorio donde los conjuntos de caracteres están instalados. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- --check, -c

  Comprueba las tablas en busca de errores.

- --check-only-changed, -C

  Chequea sólo tablas que han cambiado desde la última comprobación o que no se han cerrado correctamente.

- --compress

  Comprime toda la información enviada entre el cliente y el servidor si ambos soportan compresión.

- --databases, -B

  Procesa todas las tablas en la base de datos especificada. Con esta opción, todos los argumentos nombrados se tratan como nombres de bases de datos, no como nombres de tablas.

- --debug[=*opciones_de_depuración*], -# [*opciones_de_depuración*]

  Escribe un log de depuración. La cadena de caracteres *opciones_de_depuración* a menudo es 'd:t:o,*nombre_de_fichero*'.

- --default-character-set=*conjunto_de_caracteres*

  Usa *conjunto_de_caracteres* como el conjunto de caracteres por defecto. ConsulteSección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- `--extended`, `-e`

  Si usa esta opción para comprobar tablas, se asegura que sean 100% consistentes pero tarda bastante.

  Si usa esta opción para reparar tablas, ejecuta una reparación extendida que puede no sólo tardar bastante tiempo, sino que ¡también puede producir un montón de registros basura!

- `--fast`, `-F`

  Comprueba sólo tablas que no se han cerrado correctamente.

- `--force`, `-f`

  Continúa incluso si se produce un error SQL.

- `--host=`*`nombre_de_equipo`*, `-h` *`nombre_de_equipo`*

  Conecta con el servidor MySQL en el equipo dado.

- `--medium-check`, `-m`

  Realiza un chequeo que es más rápido que la operación `--extended`. Esto encuentra sólo el 99.99% de todos los errores, lo cual debería ser suficiente en la mayoría de casos.

- `--optimize`, `-o`

  Optimiza las tablas.

- `--password[=`*`contraseña`*`]`, `-p[`*`contraseña`*`]`

  La contraseña a usar cuando se conecta con el servidor. Si usa la opción con su forma corta (`-p`), *no* puede haber un espacio entre la opción y la contraseña. Si omite el valor *`contraseña`* a continuación de la opción `--password` o `-p` en la línea de comandos, aparece un prompt pidiéndola.

- `--port=`*`número_de_puerto`*, `-P` *`número_de_puerto`*

  El puerto TCP/IP para usar en la conexión.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  Protocolo de conexión a usar.

- `--quick`, `-q`

  Si usa esta opción para comprobar tablas, evita que el chequeo escanee los registros para buscar enlaces incorrectos. Es el método de chequeo más rápido.

  Si se utiliza esta opción para reparar tablas, el programa intenta reparar sólo el árbol del índice. Este es el método más rápido de reparación.

- `--repair`, `-r`

  Hace una reparación que puede arreglar prácticamente todo excepto claves únicas que no son únicas.

- `--silent`, `-s`

  Modo silencioso. Sólo muestra mensajes de error.

- `--socket=`*`ruta`*, `-S` *`ruta`*

Fichero socket a usar en la conexión.

- `--tables`

  Más prioritaria que la opción `--databases` o `-B`. Todos los argumentos que vienen después de la opción se consideran nombres de tablas.

- `--user=nombre_de_usuario`, `-u nombre_de_usuario`

  El nombre de usuario MySQL a usar cuando se conecta al servidor.

- `--verbose`, `-v`

  Modo explícito. Muestra información acerca de varios estados de las operaciones del programa.

- `--version`, `-V`

  Muestra información de la versión y sale.

8.7. El programa de copia de seguridad de base de datos mysqldump

El cliente `mysqldump` puede utilizarse para volcar una base de datos o colección de bases de datos para copia de seguridad o para transferir datos a otro servidor SQL (no necesariamente un servidor MySQL). EL volcado contiene comandos SQL para crear la tabla y/o rellenarla.

Si está haciendo una copia de seguridad del servidor, y las tablas son todas `MyISAM`, puede considerar usar `mysqlhotcopy` ya que hace copias de seguridad más rápidas y restauraciones más rápidas, que pueden realizarse con el segundo programa. Consulte Sección 8.8, “El programa de copias de seguridad de base de datos mysqlhotcopy”.

Hay tres formas de invocar `mysqldump`:

```
shell> mysqldump [opciones] nombre_de_base_de_datos [tablas]
shell> mysqldump [opciones] --databases DB1 [DB2 DB3...]
shell> mysqldump [opciones] --all-databases
```

Si no se nombra ninguna tabla o se utiliza la opción `--databases` o `--all-databases`, se vuelca bases de datos enteras.

Para obtener una lista de las opciones que soporta su versión de `mysqldump`, ejecute `mysqldump --help`.

Si ejecuta `mysqldump` sin las opciones `--quick` o `--opt`, `mysqldump` carga el resultado entero en memoria antes de volcarlo. Esto puede ser un problema si está volcando una base de datos grande. En MySQL 4.0, `--opt` está activado por defecto, pero puede desactivarse con `--skip-opt`.

Si está utilizando una copia reciente del programa `mysqldump` para generar un volcado que tiene que ser cargado en un servidor MySQL muy viejo, no debe usar las opciones `--opt` ni `-e` .

`mysqldump` soporta las siguientes opciones:

- `--help`, `-?`

  Muestra un mensaje de ayuda y sale.

- --add-drop-table

  Añade un comando DROP TABLE antes de cada comando CREATE TABLE .

- --add-locks

  Rodea cada volcado de tabla con los comandos LOCK TABLES y UNLOCK TABLES. Esto provoca inserciones más rápidas cuando el fichero volcado se recarga. Consulte Sección 7.2.14, “Velocidad de la sentencia INSERT”.

- --all-databases, -A

  Vuelca todas las tablas en todas las bases de datos. Es como utilizar la opción --databases y nombrar todas las bases de datos en la línea de comando.

- --allow-keywords

  Permite la creación de columnas con nombres que son palabras claves. Esto funciona añadiendo un prefijo a cada nombre de columna con el nombre de tabla.

- --comments[={0|1}]

  Si tiene como valor 0, suprime información adicional en el fichero de volcado como la versión del programa, la versión del servidor, y el equipo. --skip-comments tiene el mismo efecto que --comments=0. El valor por defecto es 1, que incluye la información extra.

- --compact

  Produce una salida (output) menos explícita. Esta opción suprime comentarios y activa las opciones --skip-add-drop-table, --no-set-names, --skip-disable-keys, y --skip-add-locks .

- --compatible=*nombre*

  Produce una salida que es compatible con otros sistemas de bases de datos o con servidores MySQL antiguos. El valor de name puede ser ansi, mysql323, mysql40, postgresql, oracle, mssql, db2, maxdb, no_key_options, no_table_options, o no_field_options. Para usar varios valores, sepárelos por comas. Estos valores tienen el mismo significado que las opciones correspondientes para asignar el modo SQL del servidor. Consulte Sección 5.3.2, “El modo SQL del servidor”.

- --complete-insert, -c

  Usa comandos INSERT compuestos que incluyen nombres de columnas.

- --compress, -C

  Comprime toda la información enviada entre el cliente y el servidor si ambos admiten compresión.

- --create-options

  Incluye todas las opciones de tabla específicas de MySQL en el comando CREATE TABLE.

- --databases, -B

  Vuelca varias bases de datos. Normalmente, mysqldump trata el primer argumento de la línea de comandos como un nombre de base de datos y los siguientes argumentos como nombres de tablas. Con esta opción, trata todos los argumentos como nombres de bases de datos. Los comandos CREATE DATABASE IF NOT EXISTS *nombre_de_base_de_datos* y USE *nombre_de_base_de_datos* están incluídos en la salida (output) antes de cada nueva base de datos.

- `--debug[=opciones_de_depuración]`, `-# [opciones_de_depuración]`

  Escribe un log de depuración. La cadena de caracteres *`opciones_de_depuración`* normalmente es `'d:t:o,nombre_de_fichero'`.

- `--default-character-set=conjunto_de_caracteres`

  Usa *`conjunto_de_caracteres`* como el conjunto de caracteres por defecto. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”. Si no se especifica, `mysqldump` desde MySQL 5.0 utiliza `utf8`.

- `--delayed-insert`

  Inserta registros usando comandos `INSERT DELAYED`. Esta opción se eliminió en MySQL 5.0.7.

- `--delete-master-logs`

  En servidores de replicación maestros, borra los logs binarios tras realizar la operación de volcado. En MySQL 5.0, esta opción se activa automáticamente `--master-data`.

- `--disable-keys`, `-K`

  Para cada tabla, rodea el comando `INSERT` con `/*!40000 ALTER TABLE tbl_name DISABLE KEYS */;` y `/*!40000 ALTER TABLE tbl_name ENABLE KEYS */;`. Esto hace que la carga del fichero volcado en MySQL 4.0 o posteriores sea más rápida porque los índices se crean sólo tras la inserción de todos los registros. Esta opción es efectiva sólo para tablas `MyISAM` .

- `--extended-insert`, `-e`

  Usa la sintaxis de `INSERT` de múltiples registros, que incluyen una lista de varios `VALUES`. Esto genera un fichero de volcado de menor tamaño y acelera las inserciones cuando se recarga el fichero.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`, `--lines-terminated-by=...`

  Estas opciones se usan con la opción `-T` y tienen el mismo significado que las cláusulas correspondientes de `LOAD DATA INFILE`. Consulte Sección 13.2.5, “Sintaxis de `LOAD DATA INFILE`”.

- `--first-slave`, `-x`

  Obsoleto. Ahora es `--lock-all-tables`.

- `--flush-logs`, `-F`

  Vuelca los ficheros de log MySQL antes de empezar el volcado. Esta opción necesita el permiso `RELOAD`. Tenga en cuenta que si utiliza esta opción en combinación con la opción `--all-databases` (o `-A`) los logs se vuelcan *para cada base de datos volcada*. La excepción es cuando se usa `--lock-all-tables` o `--master-data`. En estos casos, los logs se vuelcan sólo una vez, en el momento en que todas las tablas están bloqueadas. Si quiere que el volcado de la base de datos y el del log ocurran exactamente en el mismo momento, debe usar `--flush-logs` junto con `--lock-all-tables` o `--master-data`.

- `--force`, `-f`

  Continúa incluso si ocurre un error SQL durante un volcado de tabla.

- `--host=nombre_de_equipo`, `-h nombre_de_equipo`

Vuelca datos de un servidor MySQL en el equipo dado. Por defecto el equipo es localhost.

- --hex-blob

  Vuelca columnas de cadenas de caracteres binarios usando notación hexadecimal (por ejemplo, 'abc' es 0x616263). Las columnas sobre las que tiene efecto en MySQL 5.0 son BINARY, VARBINARY, BLOB.

- --lock-all-tables, -x

  Bloquea todas las tablas de todas las bases de datos. Esto se consigue estableciendo un bloqueo de lectura global que dura durante todo el volcado. Esta opción desactiva automáticamente --single-transaction y --lock-tables.

- --lock-tables, -l

  Bloquea todas las tablas antes de comenzar el volcado. Las tablas se bloquean con READ LOCAL para permitir inserciones concurrentes en caso de tablas MyISAM . Para tablas transaccionales como InnoDB y BDB, --single-transaction es una opción mucho mejor, ya que no necesita bloquear las tablas.

  Tenga en cuenta que al volcar múltiples bases de datos, --lock-tables bloquea tablas para cada base de datos separadamente. Así, esta opción no garantiza que las tablas en el fichero volcado sean lógicamente consistentes entre bases de datos. Tablas en bases de datos distintas pueden volcarse en estados completamente distintos.

- --master-data[=*valor*]

  Esta opción causa que se escriba en la salida (output) la posición y el nombre de fichero del log binario. Esta opción necesita el permiso RELOAD y el log binario debe estar activado. Si el valor de la opción es igual a 1, la posición y nombre de fichero se escriben en la salida del volcado en forma de comando CHANGE MASTER que hace que un servidor esclavo empiece desde la posición correcta en el log binario del maestro si usa este volcado SQL del maestro para preparar un esclavo. Si el valor de la opción es igual a 2, el comando CHANGE MASTER se escribe como un comentario SQL. Ésta es la acción por defecto si se omite *valor* .

  La opción --master-data activa --lock-all-tables, a no ser que --single-transaction también esté especificado (en tal caso, se establece un bloqueo de lectura global sólo durante un corto periodo de tiempo al principio del volcado. Consulte la descripción de --single-transaction. En cualquier caso, cualquier acción en logs se realiza en el momento exacto del volcado. Esta opción automáticamente desactiva --lock-tables.

- --no-create-db, -n

  Esta opción suprime el comando CREATE DATABASE /*!32312 IF NOT EXISTS*/ db_name que se incluye de otro modo en la salida si se especifica las opciones --databases o --all-databases.

- --no-create-info, -t

  No escribe los comandos CREATE TABLE que recrean cada tabla volcada.

- --no-data, -d

  No escribe ningún registro de la tabla. Esto es muy útil si sólo quiere obtener un volcado de la estructura de una tabla.

- --opt

Esta opción es una abreviatura; es lo mismo que especificar --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset. Causa una operación de volcado rápida y produce un fichero de volcado que puede recargarse en un servidor MySQL rápidamente. **En MySQL 5.0, --opt está activado por defecto, pero puede desactivarse con --skip-opt**. Para desactivar sólo algunas de las opciones activadas por --opt, use la forma --skip; por ejemplo --skip-add-drop-table o --skip-quick.

- --password[=*contraseña*], -p[*contraseña*]

  La contraseña a usar al conectar con el servidor. Si usa la opción en forma corta (-p), *no* puede haber un espacio entre la opción y la contraseña. Si omite el valor de *contraseña* a continuación de la opción --password o -p en la línea de comandos, aparece un prompt pidiéndola.

- --port=*número_de_puerto*, -P *número_de_puerto*

  El puerto TCP/IP a usar en la conexión.

- --protocol={TCP | SOCKET | PIPE | MEMORY}

  Protocolo de conexión a usar.

- --quick, -q

  Esta opción es útil para volcar tablas grandes. Fuerza mysqldump a recibir los registros de una tabla del servidor uno a uno en lugar de recibir el conjunto completo de registros y guardarlos en memoria antes de escribirlos.

- --quote-names, -Q

  Limita los nombres de base de datos, tablas, y columnas con caracteres '`' . Si el modo SQL del servidor incluye la opción ANSI_QUOTES, los nombres se ponen entre caracteres '"'. En MySQL 5.0, --quote-names está activado por defecto, pero puede desactivarse con --skip-quote-names.

- --result-file=*fichero*, -r *fichero*

  Redirige la salida a un fichero dado. Esta opción debe usarse en Windows, ya que previene que los caracteres de nueva línea '\n' se conviertan en secuencias de retorno/nueva línea '\r\n'.

- --set-charset

  Añade SET NAMES *conjunto_de_caracteres_por_defecto* a la salida (output). Esta opción está activada por defecto. Para suprimir el comando SET NAMES, use --skip-set-charset.

- --single-transaction

  Esta opción realiza un comando SQL BEGIN antes de volcar los datos del servidor. Es útil sólo con tablas transaccionales tales como las InnoDB y BDB, ya que vuelca el estado consistente de la base de datos cuando se ejecuta BEGIN sin bloquear ninguna aplicación.

  Cuando use esta opción, debe tener en cuenta que sólo las tablas InnoDB se vuelcan en un estado consistente. Por ejemplo, cualquier tabla MyISAM o HEAP volcadas mientras se usa esta opción todavía pueden cambiar de estado.

  La opción --single-transaction y la opción --lock-tables son mutuamente exclusivas, ya que LOCK TABLES provoca que cualquier transacción pendiente se confirme implícitamente.

  Para volcar tablas grandes, debe combinar esta opción con --quick.

- --socket=*ruta*, -S *ruta*

  El fichero socket a usar cuando se conecta con localhost (que es el equipo por defecto).

- --skip-comments

  Consulte la descripción de la opción --comments.

- --tab=*ruta*, -T *ruta*

  Produce ficheros con datos separados por tabuladores. Para cada tabla volcada mysqldump crea un fichero *nombre_de_tabla*.sql que contiene el comando CREATE TABLE que crea la tabla, y un fichero *nombre_de_tabla*.txt que contiene los datos. El valor de esta opción es el directorio en el que escribir los ficheros.

  Por defecto, los ficheros de datos .txt se formatean usando tabuladores entre los valores de las columnas y una nueva línea tras cada registro. El formato puede especificarse explícitamente usando las opciones --fields-*xxx* y --lines--*xxx*.

  Nota: Esta opción debe usarse sólo cuando mysqldump se ejecuta en la misma máquina que el servidor mysqld. Se debe tener el permiso FILE, y el servidor debe tener permisos para escribir ficheros en el directorio que se especifique.

- --tables

  Tiene mayor prioridad que --databases o -B. Todos los argumentos que vienen después de esta opción se tratan como nombres de tablas.

- --user=*nombre_de_usuario*, -u *nombre_de_usuario*

  Nombre de usuario MySQL a usar al conectar con el servidor.

- --verbose, -v

  Modo explícito. Muestra más información sobre lo que hace el programa.

- --version, -V

  Muestra información de versión y sale.

- --where='*condición_where*', -w '*condición_where*'

  Vuelca sólo registros seleccionados por la condición WHERE dada. Tenga en cuenta que las comillas alrededor de la condición son obligatorias si contienen espacios o caracteres especiales para el intérprete de comandos.

  Ejemplos:

```
"--where=user='jimf'"
"-wuserid>1"
"-wuserid<1"
```

- --xml, -X

  Escribe la salida del volcado como XML bien formado.

Puede cambiar las siguientes variables usando las opciones --*nombre_de_variable*=*valor*:

- max_allowed_packet

Tamaño máximo del búfer para la comunicación cliente/servidor. En MySQL 5.0, el valor de esta variable puede ser de hasta 1GB.

- net_buffer_length

  Tamaño inicial del búfer para la comunicación cliente/servidor. Cuando se crean comandos de inserción de múltiples registros (como con las opciones --extended-insert o --opt), mysqldump crea registros de longitud máxima net_buffer_length. Si incrementa esta variable, debe asegurarse también de que la variable net_buffer_length en el servidor MySQL tenga como mínimo esta longitud.

También es posible cambiar variables usando la sintaxis --set-variable=*nombre_de_variable=valor* o -O *nombre_de_variable=valor*. Sin embargo, esta sintaxis está obsoleta.

El uso más común de mysqldump es para hacer una copia de seguridad de toda la base de datos:

```
shell> mysqldump --opt nombre_de_base_de_datos > fichero_de_seguridad.sql
```

El siguiente ejemplo muestra cómo volcar el fichero de seguridad de nuevo en el servidor:

```
shell> mysql nombre_de_base_de_datos < fichero_de_seguridad.sql
```

El siguiente ejemplo obtiene el mismo resultado que el anterior:

```
shell> mysql -e "source /ruta/fichero_de_seguridad.sql" nombre_de_base_de_datos
```

mysqldump es muy útil para poblar bases de datos copiando los datos de un servidor MySQL a otro:

```
shell> mysqldump --opt nombre_de_base_de_datos | mysql --host=nombre_de_equipo_remoto -C nombre_de_base_de_dat
```

Es posible volcar varias bases de datos con un solo comando:

```
shell> mysqldump --databases nombre_de_base_de_datos_1 [nombre_de_base_de_datos_2 ...] > mis_bases_de_datos.sq
```

Si quiere volcar todas las bases de datos, use la opción --all-databases:

```
shell> mysqldump --all-databases > todas_las_bases_de_datos.sql
```

Si las tablas se guardan con el motor de almacenamiento InnoDB, mysqldump proporciona una forma de realizar una copia de seguridad de las mismas (consulte los comandos a continuación). Esta copia de seguridad sólo necesita un bloqueo local de todas las tablas (usando FLUSH TABLES WITH READ LOCK) al principio del volcado. En cuanto obtiene el bloqueo, se lee el log binario y se libera el bloqueo. Si y sólo si un comando de actualización largo está en ejecución cuando se ejecuta FLUSH..., el servidor MySQL puede quedar bloqueado hasta que acabe este comando largo, y luego el volcado queda sin ningún bloqueo. Si el servidor MySQL recibe sólo comandos de actualización cortos (en el sentido de "poco tiempo de ejecución"), incluso si son muchos, el periodo inicial de bloqueo no debe ser un problema.

```
shell> mysqldump --all-databases --single-transaction > todas_las_bases_de_datos.sql
```

Para una recuperación en un momento dado (también comocido como "roll-forward", cuando necesita restaurar una copia de seguridad antigua y recrear los cambios que han ocurrido desde tal copia de

seguridad), es útil rotar el log binario (consulte Sección 5.10.3, “El registro binario (Binary Log)”) o al menos conozca las coordinadas del log binario que se corresponden con el volcado:

```
shell> mysqldump --all-databases --master-data=2 > todas_las_bases_de_datos.sql
or
shell> mysqldump --all-databases --flush-logs --master-data=2 > todas_las_bases_de_datos.sql
```

El uso simultáneo de `--master-data` y `--single-transaction` proporciona una forma de hacer copias de seguridad en línea apropiadas para recuperaciones en un momento dado, si las tablas se guardan con el motor de almacenamiento `InnoDB`.

Para más información sobre copias de seguridad, consulte Sección 5.8.1, “Copias de seguridad de bases de datos”.

8.8. El programa de copias de seguridad de base de datos `mysqlhotcopy`

`mysqlhotcopy` es un script Perl que fue escrito originalmente por Tim Bunce. Usa `LOCK TABLES`, `FLUSH TABLES`, y `cp` o `scp` para realizar una copia de seguridad rápida de la base de datos. Es la forma más rápida de hacer una copia de seguridad de la base de datos o de tablas, pero sólo puede ejecutarse en la misma máquina donde está el directorio de base de datos. `mysqlhotcopy` sólo realiza copias de seguridad de tablas `MyISAM`. Funciona en Unix y NetWare.

```
shell> mysqlhotcopy nombre_de_base_de_datos [/ruta/al/nuevo_directorio]
```

```
shell> mysqlhotcopy nombre_de_base_de_datos_1 ... nombre_de_base_de_datos_n /ruta/al/nuevo_directorio
```

En la base de datos señalada realiza una copia de seguridad de las tablas que verifican una expresión regular dada:

```
shell> mysqlhotcopy nombre_de_base_de_datos./expresión_regular/
```

La expresión regular para el nombre de tabla puede negarse precediéndola con una tilde ('~'):

```
shell> mysqlhotcopy nombre_de_base_de_datos./~expresión_regular/
```

`mysqlhotcopy` soporta las siguientes opciones:

- `--help`, `-?`

  Muestra un mensaje de ayuda y sale.

- `--allowold`

  No aborta si el objetivo ya existe (lo renombra añadiendo un sufijo `_old`).

- `--checkpoint=nombre_de_base_de_datos.nombre_de_tabla`

  Inserta puntos de referencia en la base de datos *`nombre_de_base_de_datos`* y en la tabla *`nombre_de_tabla`*.

- `--debug`

Activa la opción de depuración.

- `--dryrun`, `-n`

  Reporta acciones sin ejecutarlas realmente.

- `--flushlog`

  Vuelca logs tras bloquear todas las tablas.

- `--keepold`

  No borra objetivos previos (renombrados) cuando acaba.

- `--method=#`

  Método para copiar (`cp` o `scp`).

- `--noindices`

  No incluye los índices en la copia de seguridad. Esto hace que la copia de seguridad sea más inteligente y rápida. Los índices pueden reconstruirse posteriormente con `myisamchk -rq`.

- `--password=`*`contraseña`*`, -p`*`contraseña`*

  La contraseña a usar al conectar con el servidor. Tenga en cuenta que el valor de la contraseña no es opcional para esta opción, no como con otros programas MySQL.

- `--port=`*`número_de_puerto`*, `-P` *`número_de_puerto`*

  El puerto TCP/IP a usar cuando se conecta el servidor local.

- `--quiet`, `-q`

  Es silencioso excepto para errores.

- `--regexp=`*`expresión_regular`*

  Copia todas las bases de datos con nombres que cumplan la expresión regular dada.

- `--socket=`*`ruta`*, `-S` *`ruta`*

  El fichero socket Unix a usar para la conexión.

- `--suffix=`*`cadena`*

  El sufijo para nombres de bases de datos copiadas.

- `--tmpdir=`*`ruta`*

  Directorio temporal (en lugar de `/tmp`).

- `--triggers`

  Vuelca disparadores para cada tabla volcada. Esta opción está activada por defecto; desactívela con `--skip-triggers`. Esta opción se añadió en MySQL 5.0.11. Antes de esta versión, los disparadores no se vuelcan.

- `--user=`*`nombre_de_usuario`*, `-u` *`nombre_de_usuario`*

El nombre de usuario MySQL a usar cuando se conecta al servidor.

mysqlhotcopy lee los grupos de opciones [client] y [mysqlhotcopy] de los ficheros de opciones.

Para ejecutar mysqlhotcopy, debe tener: acceso a los ficheros de las tablas de las que está haciendo copia de seguridad; el permiso SELECT para estas tablas; y el permiso RELOAD (para poder ejecutar FLUSH TABLES).

Use perldoc para información adicional de mysqlhotcopy:

```
shell> perldoc mysqlhotcopy
```

8.9. El programa para importar datos mysqlimport

El cliente mysqlimport proporciona una interfaz de línea de comandos para el comando LOAD DATA INFILE. La mayoría de opciones de mysqlimport se corresponden directamente con cláusulas de LOAD DATA INFILE. Consulte Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.

Invoque mysqlimport así:

```
shell> mysqlimport [opciones] nombre_de_base_de_datos fichero_de_texto1 [fichero_de_texto2 ...]
```

Del nombre de cada fichero de texto especificado en la línea de comandos, mysqlimport elimina cualquier extensión, y utliza el resultado para determinar el nombre de la tabla a la que importar el contenido del fichero. Por ejemplo, los ficheros con nombres patient.txt, patient.text y patient se importarían todos a la tabla llamada patient.

mysqlimport soporta las siguientes opciones:

- --help, -?

  Muestra un mensaje de ayuda y sale.

- --columns=*lista_de_columnas*, -c *lista_de_columnas*

  Esta opción admite una lista de nombres de columnas separados por comas. El orden de los nombres de columna indica cómo emparejar las columnas de los ficheros de datos con las columnas de la tabla.

- --compress, -C

  Comprime toda la información enviada entre el cliente y el servidor, si ambos soportan compresión.

- --debug[=*opciones_de_depuración*], -# [*opciones_de_depuración*]

  Escribe un log de depuración. La cadena de caracteres *opciones_de_depuración* a menudo es 'd:t:o,*nombre_de_fichero*'.

- --delete, -D

  Vacía la tabla antes de importar el fichero de texto.

- --fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=..., --lines-terminated-by=...

  Estas opciones tienen el mismo significado que las cláusulas correspondientes de LOAD DATA INFILE. Consulte Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.

- `--force`, `-f`

  Ignora errores. Por ejemplo, si una tabla para un fichero de texto no existe, sigue procesando el resto de ficheros. Sin `--force`, `mysqlimport` finaliza si la tabla no existe.

- `--host=`*`nombre_de_equipo`*, `-h` *`nombre_de_equipo`*

  Importa datos al servidor MySQL en el equipo dado. El equipo por defecto es `localhost`.

- `--ignore`, `-i`

  Consulte la descripción para la opción `--replace`.

- `--ignore-lines=`*`n`*

  Ignora las primeras *`n`* líneas del fichero de datos.

- `--local`, `-L`

  Lee los ficheros de entrada localmente del equipo cliente.

- `--lock-tables`, `-l`

  Bloquea *todas* las tablas para escritura antes de procesar cualquier fichero de texto. Esto asegura que todas las tablas estén sincronizadas en el servidor.

- `--password[=`*`contraseña`*`]`, `-p[`*`contraseña`*`]`

  La contraseña a usar cuando se conecta al servidor. Si usa la opción en su forma corta (`-p`), *no* puede haber un espacio entre la opción y la contraseña. Si omite el valor de *`contraseña`* a continuación de `--password` o `-p` en la línea de comandos, aparece un prompt para que lo introduzca.

- `--port=`*`número_de_puerto`*, `-P` *`número_de_puerto`*

  El puerto TCP/IP para usar en la conexión.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  El protocolo de conexión a usar. Nuevo en MySQL 4.1.

- `--replace`, `-r`

  Las opciones `--replace` y `--ignore` controlan el tratamiento de registros de entrada que duplican registros existentes con valores clave únicos. Si especifica `--replace`, los nuevos registros replazan los existentes que tengan el mismo valor clave. Si especifica `--ignore`, los registros nuevos que duplican un registro existente con el mismo valor clave se ignoran. Si no especifica ninguna opción, se produce un error cuando se encuentra un valor duplicado, y el resto del fichero de texto se ignora.

- `--silent`, `-s`

  Modo silencioso. Sólo muestra mensajes de error.

- `--socket=`*`ruta`*, `-S` *`ruta`*

  Fichero socket a usar al conectar con `localhost` (que es el equipo por defecto).

- `--user=`*`nombre_de_usuario`*, `-u` *`nombre_de_usuario`*

  El nombre de usuario MySQL a usar cuando se conecta con el servidor.

- `--verbose`, `-v`

  Modo explícito. Muestra más información sobre lo que hace el programa.

- `--version`, `-V`

  Muestra información de versión y sale.

Ejemplo de una sesión que demuestra el uso de `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 |
+------+---------------+
```

8.10. Mostrar bases de datos, tablas y columnas con `mysqlshow`

El cliente `mysqlshow` puede usarse para ver rápidamente qué bases de datos existen, sus tablas, y las columnas de las tablas e índices.

`mysqlshow` proporciona una interfaz de línea de comandos para varios comandos SQL `SHOW`. La misma información puede obtenerse usando estos comandos directamente. Por ejemplo, puede ejecutarlos desde el programa cliente `mysql`. Consulte Sección 13.5.4, “Sintaxis de SHOW”.

Invoque `mysqlshow` así:

```
shell> mysqlshow [opciones] [nombre_de_base_de_datos [nombre_de_tabla [nombre_de_columna]]]
```

- Si no se da una base de datos, se muestran todas las bases de datos.
- Si no se da una tabla, se muestran todas las tablas de la base de datos.
- Si no se da una columna, se muestran todas las columnas y tipos de columnas de la tabla.

Si el último argumento contiene caracteres de shell o comodines SQL ('*', '?', '%', o '_'), sólo se muestran aquéllos nombres que coinciden con el comodín. Si un nombre de base de datos contiene algún carácter de subrayado, debe ponerse una barra invertida (algunos shells Unix necesitan dos) para obtener una lista de las tablas o columnas adecuadas. Los caracteres '*' y '?' se convierten en los caracteres comodines SQL '%' y '_'. Esto puede causar confusión cuando se trate de mostrar las columnas para una tabla con '_' en el nombre, ya que en este caso `mysqlshow` muestra sólo los nombres de tablas que cumplen con el

patrón. Esto se puede arreglar fácilmente añadiendo un carácter '%' extra en la línea de comandos como argumento separado.

mysqlshow soporta las siguientes opciones:

- --help, -?

  Muestra un mensaje de ayuda y sale.

- --character-sets-dir=*ruta*

  El directorio donde están instalados los conjuntos de caracteres. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- --compress, -C

  Comprime toda la información enviada entre el cliente y el servidor si ambos soportan compresión.

- --debug[=*opciones_de_depuración*], -# [*opciones_de_depuración*]

  Escribe un log de depuración. La cadena de caracteres *opciones_de_depuración* a menudo es 'd:t:o,*nombre_de_fichero*'.

- --default-character-set=*conjunto_de_caracteres*

  Usa *conjunto_de_caracteres* como el conjunto de caracteres por defecto. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”.

- --host=*nombre_de_equipo*, -h *nombre_de_equipo*

  Conecta con el servidor MySQL del equipo dado.

- --keys, -k

  Muestra los índices de la tabla.

- --password[=*contraseña*], -p[*contraseña*]

  La contraseña a usar cuando se conecta con el servidor. Si usa el formato corto de la opción (-p), *no* puede haber un espacio entre la opción y la contraseña. Si omite el valor de *contraseña* a continuación de la opción --password o -p en la línea de comandos, aparece un prompt para que lo introduzca.

- --port=*número de puerto*, -P *número de puerto*

  El puerto TCP/IP a usar para la conexión.

- --protocol={TCP | SOCKET | PIPE | MEMORY}

  El protocolo de conexión a usar.

- --show-table-type

  Muestra una columna indicando el tipo de tabla, como con SHOW FULL TABLES. Nuevo en MySQL 5.0.4.

- --socket=*ruta*, -S *ruta*

  El fichero socket a usar cuando se conecta con localhost (que es el equipo por defecto).

- `--status`, `-i`

  Muestra información extra de cada tabla.

- `--user=nombre_de_usuario`, `-u nombre_de_usuario`

  El nombre de usuario MySQL a usar cuando se conecta al servidor.

- `--verbose`, `-v`

  Modo explícito. Muestra más información sobre lo que hace el programa. Esta opción puede usarse varias veces para incrementar la cantidad de información.

- `--version`, `-V`

  Muestra información de versión y sale.

8.11. `perror`, explicación de códigos de error

Para la mayoría de errores de sistema, MySQL muestra, además del mensaje de texto interno, el código de error de sistema en uno de los siguientes estilos:

```
message ... (errno: #)
message ... (Errcode: #)
```

Puede aclararse qué significa cada código de error examinando la documentación del sistema o utilizando la utilidad `perror` .

`perror` muestra la descripción de códigos de error del sistema y del motor de almacenamiento (lo que maneja las tablas).

Invoque `perror` así:

```
shell> perror [opciones] código_de_error ...
```

Ejemplo:

```
shell> perror 13 64
Error code  13:  Permission denied
Error code  64:  Machine is not on the network
```

Nota: Para obtener el mensaje de error para un código de error de MySQL, invoque `perror` con la opción `--ndb` :

```
shell> perror --ndb errorcode
```

Tenga en cuenta que el significado de los mensajes de error de sistema pueden ser dependientes del sistema operativo. Un código de error puede significar cosas distintas en diferentes sistemas operativos.

8.12. La utilidad `replace` de cambio de cadenas de caracteres

El programa `replace` cambia cadenas de caracteres en ficheros o en la entrada estándar. Utiliza una máquina de estado finito para encontrar las cadenas de caracteres más largas en primer lugar. Puede utilizarse para cambiar cadenas de caracteres. Por ejemplo, el siguiente comando intercambia `a` y `b` en los ficheros dados, `fichero1` y `fichero2`:

```
shell> replace a b b a -- fichero1 fichero2 ...
```

Use la opción `--` para indicar dónde acaba la lista de reemplazo de cadena de caracteres y dónde empieza el nombre del fichero.

Todo fichero nombrado en la línea de comandos se modifica en su ubicación, así que puede ser conveniente hacer una copia de seguridad del fichero original antes de modificarlo.

Si no se especifica ningún fichero en la línea de comandos `replace` lee la entrada estándar y escribe en la salida estándar. En este caso, no se necesita ninguna opción `--` .

`msql2mysql` utiliza el programa `replace`. Consulte Sección 24.1.1, “msql2mysql — ”.

`replace` soporta las siguientes opciones:

- `-?`, `-I`

  Muestra un mensaje de ayuda y sale.

- `-# opciones_de_depuración`

  Escribe un log de depuración. La cadena de caracteres *`opciones_de_depuración`* a menudo es `'d:t:o,`*`nombre_de_fichero`*`'`.

- `-s`

  Modo silencioso. Muestra menos información sobre lo que hace el programa.

- `-v`

  Modo explícito. Muestra más información sobre lo que hace el programa.

- `-V`

  Muestra información de versión y sale.

Capítulo 9. Estructura de lenguaje

Tabla de contenidos

Este capítulo trata sobre las reglas a seguir cuando se escriban los siguientes elementos de sentencias SQL durante el uso de MySQL:

- Valores literales, como cadenas y números.
- Identificadores, como nombres de tablas y columans.
- Variables de usuario y de sistema
- Comentarios
- Palabras reservadas

9.1. Valores literales

En esta sección se trata la escritura de valores literales en MySQL. Estos incluyen a las cadenas, números, valores hexadeciales, valores booleanos y `NULL`. También se ocupa de las distintas situaciones (algunas muy propensas a error) en las que se puede incurrir al manejar estos tipos básicos de MySQL.

9.1.1. Cadenas de caracteres

Una cadena (string) es una secuencia de caracteres, encerrada por comillas simples (`''`) o dobles (`""`). Ejemplos:

```
'una cadena'
"otra cadena"
```

Si el modo de servidor tiene habilitado `ANSI_QUOTES`, las cadenas solamente pueden delimitarse con comillas simples. Una cadena delimitada por comillas dobles será interpretada como un identificador.

A partir de MySQL 4.1.1, las cadenas pueden tener una parte indicativa del conjunto de caracteres y una cláusula `COLLATE`:

```
[_conjunto_caracteres]'cadena' [COLLATE tipo_ordenación]
```

Ejemplos:

```
SELECT _latin1'cadena';
SELECT _latin1'cadena' COLLATE latin1_danish_ci;
```

Para más información sobre esta sintaxis consulte Sección 10.3.7, “Conjunto de caracteres y colación de columnas “carácter””.

Dentro de una cadena, ciertas secuencias de caracteres tienen un significado especial. Cada una de estas secuencias comienza con una barra diagonal invertida ('\'), conocida como *caracter de escape*. MySQL reconoce las siguientes secuencias de escape:

\0	Un caracter ASCII 0 (NUL).
\'	Un caracter de comilla simple ('').
\"	Un carácter de comilla doble ('"').
\b	Un carácter de retroceso.
\n	Un carácter de salto de línea.
\r	Un carácter de retorno de carro.
\t	Un carácter de tabulación.
\Z	ASCII 26 (Control-Z). Este carácter puede codificarse como '\Z' para solventar el problema de que el ASCII 26 se interpreta en Windows como fin de fichero. (El ASCII 26 causará problemas si se intenta emplear `mysql nombre_bd < nombre_fichero`.)
\\	Un carácter de barra invertida ('\').
\%	Un carácter '%'. Consulte la nota a continuación.
_	Un carácter '_'. Consulte la nota a continuación.

Estas secuencias son sensibles a mayúsculas. Por ejemplo, '\b' se interpreta como carácter de retroceso, pero '\B' se interpreta como 'B'.

Las secuencias '\%' y '_' se emplean para buscar apariciones literales de '%' y '_' en un contexto de búsqueda por patrones, donde en otro caso se debieran interpretar como caracteres comodines. Consulte Sección 12.3.1, “Funciones de comparación de cadenas de caracteres”. Hay que advertir que si se emplean '\%' o '_' en otra situación, devolverán las cadenas '\%' y '_' y no '%' y '_'.

En toda otra secuencia de escape, la barra invertida se ignora. Esto es, el carácter al que se aplica se interpreta como si no tuviera delante un carácter de escape.

Hay varias formas de incluir comillas dentro de una cadena:

- Un ''' dentro de una cadena que está delimitada por ''' debe escribirse como ''''.
- Un '"' dentro de una cadena que está delimitada por '"' debe escribirse como '""'.
- Se puede preceder el carácter de comillas con un carácter de escape. ('\').
- Un ''' dentro de una cadena delimitada con '"' no necesita ningún tratamiento especial, ni colocarla en forma doble ni precederla con un carácter de escape. Lo mismo se cumple para una '"' colocada en una cadena delimitada con '''.

Las siguientes sentencias `SELECT` demuestran cómo actúan las comillas y los caracteres de escape:

```
mysql> SELECT 'hola', '"hola"', '""hola""', 'hol''a', '\'hola';
+------+--------+----------+-------+-------+
| hola | "hola" | ""hola"" | hol'a | 'hola |
+------+--------+----------+-------+-------+

mysql> SELECT "hola", "'hola'", "''hola''", "hol""a", "\"hola";
+------+--------+----------+-------+-------+
| hola | 'hola' | ''hola'' | hol"a | "hola |
+------+--------+----------+-------+-------+

mysql> SELECT 'Estas\nSon\nCuatro\nLíneas';
+--------------------+
| Estas
Son
Cuatro
Líneas |
+--------------------+

mysql> SELECT 'barra\ desaparece';
+------------------+
| barra desaparece |
+------------------+
```

Si se pretende insertar datos binarios en una columna de tipo cadena (por ejemplo un `BLOB`), los siguientes caracteres deberán representarse con secuencias de escape:

`NUL`	Byte `NUL` (ASCII 0). Este carácter se representará con '`\0`' (una barra invertida seguida de un carácter ASCII '`0`').
`\`	Barra invertida (ASCII 92). Este carácter se representará con '`\\`'.
`'`	Comilla simple (ASCII 39). Este carácter se representará con '`\'`'.
`"`	Comilla doble (ASCII 34). Este carácter se representará con '`\"`'.

Al escribir programas de aplicación, cualquier cadena que pudiese contener cualquiera de estos caracteres especiales deberá ser preparada antes de utilizarse como valor en una sentencia SQL que se enviará al servidor MySQL. Esto puede hacerse de dos maneras:

- Procesando la cadena con una función que reemplace los caracteres especiales con una secuencia de escape. Por ejemplo, en un programa C, se puede emplear la función de la API de C `mysql_real_escape_string()`. Consulte Sección 24.2.3.48, “`mysql_real_escape_string()`”. La interfaz DBI de Perl proporciona un método `quote` para convertir caracteres especiales a las secuencias de escape equivalentes. Consulte Sección 24.4, “La API Perl de MySQL”.

- Como alternativa al reemplazo explícito de caracteres especiales, varias APIs de MySQL proporcionan la parametrización de consultas, lo que permite insertar marcadores especiales en una consulta y luego asociarles valores al momento de emitirla. En este caso, la API toma a su cargo el reemplazo de caracteres especiales en los valores.

9.1.2. Números

Los enteros se representan como secuencias de dígitos. Los flotantes utilizan '`.`' como separador decimal. Cada tipo de número puede estar precedido con '`-`' para indicar un valor negativo.

Ejemplos de enteros válidos:

```
1221
0
-32
```

Ejemplos de números de punto flotante válidos:

```
294.42
-32032.6809e+10
148.00
```

Un entero puede usarse en un contexto de punto flotante; se intepretará como el número de punto flotante equivalente.

9.1.3. Valores hexadecimales

MySQL soporta valores hexadecimales. En contextos numéricos, éstos actuarán como enteros (con precisión de 64 bits). En contextos de cadena, actuarán como cadenas binarias, donde cada par de dígitos hexadecimales es convertido a un carácter:

```
mysql> SELECT x'4D7953514C';
        -> 'MySQL'
mysql> SELECT 0xa+0;
        -> 10
mysql> SELECT 0x5061756c;
        -> 'Paul'
```

En MySQL 5.0, el tipo predeterminado para un valor hexadecimal es una cadena. Si se desea estar seguro de que el valor se tratará como un número, puede emplearse `CAST(... AS UNSIGNED)`:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
        -> 'A', 65
```

La sintaxis `0x` se basa en ODBC. Las cadenas hexadecimales son utilizadas a menudo por ODBC para proveer valores para columnas `BLOB`. La sintaxis `x'hexstring'` se basa en SQL standard.

Se puede convertir una cadena o un número en una cadena en formato hexadecimal con la función `HEX()`:

```
mysql> SELECT HEX('cat');
        -> '636174'
mysql> SELECT 0x636174;
        -> 'cat'
```

9.1.4. Valores booleanos

En MySQL 5.0, la constante `TRUE` se evalúa como `1` y la constante `FALSE`, como `0`. Los nombres de constantes pueden escribirse en cualquier combinación de mayúsculas y minúsculas.

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

9.1.5. Valores de bits

A partir de MySQL 5.0.3, los valores de bits pueden escribirse utilizando la notación `b'valor'`. *`value`* es un valor binario escrito empleando ceros y unos.

La notación de bits es conveniente para especificar valores que se asignarán a columnas BIT:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
+------+----------+----------+----------+
| b+0  | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+------+----------+----------+----------+
|  255 | 11111111 | 377      | FF       |
|   10 | 1010     | 12       | A        |
+------+----------+----------+----------+
```

9.1.6. Valores NULL

El valor NULL significa “no hay dato.” NULL puede escribirse en cualquier combinación de mayúsculas y minúsculas.

Debe tenerse en cuenta que el valor NULL no es lo mismo que 0 para tipos numéricos o la cadena vacía para tipos de cadena. Consulte Sección A.5.3, “Problemas con valores NULL”.

Para operaciones de exportación o importación de texto utilizando LOAD DATA INFILE o SELECT ... INTO OUTFILE, NULL se representa con la secuencia \N. Consulte Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.

9.2. Nombres de bases de datos, tablas, índices, columnas y alias

Los nombres de bases de datos, tablas, índices, columnas y alias son identificadores. Esta sección describe la sintaxis permitida para los identificadores de MySQL.

La siguiente tabla describe la longitud máxima y los caracteres permitidos para cada tipo de identificador.

Identificador	Longitud máxima (en bytes)	Caracteres permitidos
Base de datos	64	Cualquier carácter permitido en un nombre de directorio, excepto '/', '\', o '.'
Tabla	64	Cualquier carácter permitido en un nombre de fichero, excepto '/', '\', o '.'
Columna	64	Todos los caracteres
Índice	64	Todos los caracteres
Alias	255	Todos los caracteres

Adicionalmente a las restricciones detalladas en la tabla, ningún identificador puede contener un carácter ASCII 0 o un byte con un valor de 255. Los nombres de bases de datos, tablas y columnas no deberían terminar con caracteres de espacio. MySQL 5.0 permite el uso de comillas en identificadores, aunque es mejor evitarlos tanto como sea posible.

En MySQL 5.0, los identificadores se almacenan empleando Unicode (UTF8). Esto se aplica a identificadores en las definiciones de tabla que se almacenan en ficheros .frm y a identificadores almacenados en las tablas de permisos en la base de datos mysql. El tamaño de las columnas de tipo cadena en las tablas de permisos (y en cualquier otra tabla) de MySQL 5.0 equivale al número de caracteres, esto significa que (al contrario que en algunas versiones anteriores de MySQL) se pueden

utilizar caracteres multibyte sin reducir el número de caracteres permitidos para los valores almacenados en estas columnas.

Un identificador puede estar encerrado entre comillas o no. Si un identificador es una palabra reservada o contiene caracteres especiales, *se debe* encerrar entre comillas cada vez que se haga referencia a él. Para una lista de palabras reservadas, consulte Sección 9.6, “Tratamiento de palabras reservadas en MySQL”. Los caracteres especiales son aquellos que están fuera del grupo de caracteres alfanuméricos del conjunto de caracteres en uso, de '_', y de '$'.

El carácter de encomillado de identificador es el acento grave: ('`'):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

Si el modo de servidor SQL incluye la opción `ANSI_QUOTES`, también está permitido delimitar los identificadores con comillas dobles:

```
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)
```

Consulte Sección 5.3.2, “El modo SQL del servidor”.

En MySQL 5.0, los caracteres delimitadores de identificador pueden incluírse dentro del identificador *si se delimita el identificador*. Si el carácter que se incluirá dentro del identificador es el mismo que se utiliza para delimitarlo, habrá que colocarlo en forma doble. Las siguientes sentencias crean una tabla llamada `a`b` que contiene una columna llamada `c"d`:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

Se recomienda que no se utilicen nombres con el esquema *XeX*, tal como `1e` o `2e2`, porque una expresión como `1e+1` es ambigua. Podría interpretarse como la expresión `1e + 1` o como el número `1e+1`, dependiendo del contexto.

Hay que ser cuidadoso al utilizar `MD5` para producir nombres de tablas, porque puede producir nombres ilegales como los listados anteriormente.

9.2.1. Cualificadores de los identificadores

MySQL acepta nombres que pueden consistir en un solo identificador o múltiples identificadores. Los componentes de un nombre múltiple deben separarse con un punto ('.'). Las partes iniciales de un identificador múltiple actúan como calificadores que afectan el contexto en el cual se interpreta la parte final.

En MySQL es posible referirse a una columna empleando cualquiera de las siguientes formas:

Referencia de columna	Significado
col_name	La columna *col_name* de cualquier tabla empleada en la consulta que contenga una columna con ese nombre
tbl_name.col_name	La columna *col_name* de la tabla *tbl_name* en la base de datos predeterminada.

db_name.tbl_name.col_name	La columna *col_name* en la tabla *tbl_name* en la base de datos *db_name*.

Si cualquier componente de un nombre múltiple requiere delimitarlo, hay que hacerlo individualmente en lugar de delimitar el nombre como un todo. Por ejemplo, `` `mi-tabla`.`mi-columna` `` es legal, pero `` `mi-tabla.mi-columna` `` no lo es.

No es necesario especificar un prefijo de *nombre_tabla* o *nombre_bd.nombre_tabla* para referenciar una columna en una sentencia a menos que la referencia sea ambigua. Supóngase el caso de las tablas `t1` y `t2` cada una conteniendo una columna `c`, donde se recupera `c` en una sentencia `SELECT` que emplea ambas tablas `t1` y `t2`. En este caso, `c` es ambiguo porque no es único entre las tablas utilizadas en la sentencia. Se lo debe calificar con un nombre de tabla como `t1.c` o `t2.c` para indicar a cuál tabla se refiere la consulta. Del mismo modo, para recuperar desde una tabla `t` en la base de datos `db1` y desde la tabla `t` en la base de datos `db2` en la misma sentencia, hay que referirse a las columnas en aquellas tablas como `db1.t.col_name` y `db2.t.col_name`.

Una palabra a continuación de un punto en un nombre calificado debe ser un identificador, por lo que no es necesario delimitarlo, aun si es una palabra reservada.

La sintaxis *.tbl_name* hace referencia a la tabla *tbl_name* en la base de datos actual. Esta sintaxis se acepta por compatibilidad con ODBC, ya que algunos programas ODBC anteceden los nombres de tabla con un carácter '`.`'.

9.2.2. Sensibilidad a mayúsculas y minúsculas de identificadores

En MySQL, las bases de datos se corresponden con directorios dentro del directorio de datos. Cada tabla dentro de una base de datos corresponde a por lo menos un fichero dentro del directorio de la base de datos (y posiblemente más, dependiendo del motor de almacenamiento). Por lo tanto, es la sensibilidad a mayúsculas del sistema operativo subyacente la que determina la sensibilidad a mayúsculas en los nombres de tablas y bases de datos. Esto significa que los nombres de las tablas y las bases de datos son sensibles a mayúsculas en la mayoría de las variedades de Unix, pero no lo son en Windows. Una notable excepción es Mac OS X, el cual se basa en Unix pero utiliza en forma predeterminada un sistema de ficheros (HFS+) que no es sensible a mayúsculas. No obstante, Mac OS X también soporta volúmenes UFS, los cuales son sensibles a mayúsculas tal como cualquier Unix. Consulte Sección 1.7.4, “Extensiones MySQL al estándar SQL”.

Nota: Si bien los nombres de bases de datos y tablas no son sensibles a mayúsculas en algunas plataformas, no habría que referirse a una tabla o base de datos con diferentes combinaciones de mayúsculas y minúsculas dentro de la misma consulta. La siguiente consulta podría fallar porque se refiere a una tabla como `my_table` y `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Los nombres de columnas, índices, procedimientos almacenados y triggers no son sensibles a mayúsculas en ninguna plataforma, ni tampoco lo son los alias de columnas.

En forma predeterminada, los alias de tabla en MySQL 5.0 son sensibles a mayúsculas en Unix, pero no en Windows o Mac OS X. La siguiente consulta no funcionaría en Unix, porque se refiere al alias en ambas formas `a` y `A`:

```
mysql> SELECT nombre_col FROM nombre_tabla AS a
    -> WHERE a.nombre_col = 1 OR A.nombre_col = 2;
```

Sin embargo, la misma consulta está permitida en Windows. Para evitar estar pendiente de tales diferencias, lo mejor es adoptar una convención consistente, como crear y referirse a bases de datos y

tablas usando siempre minúsculas. Esto es lo recomendado para lograr máxima portabilidad y facilidad de uso.

La forma en que los nombres de tablas y bases de datos se almacenan en el disco y se usan en MySQL se define mediante la variable de sistema `lower_case_table_names`, a la cual se le puede establecer un valor al iniciar `mysqld`. `lower_case_table_names` puede tomar uno de los siguientes valores:

Valor	Significado
`0`	Los nombres de tablas y bases de datos se almacenan en disco usando el esquema de mayúsculas y minúsculas especificado en las sentencias `CREATE TABLE` o `CREATE DATABASE`. Las comparaciones de nombres son sensibles a mayúsculas. Esto es lo predeterminado en sistemas Unix. Nótese que si se fuerza un valor 0 con `--lower-case-table-names=0` en un sistema de ficheros insensible a mayúsculas y se accede a tablas `MyISAM` empleando distintos esquemas de mayúsculas y minúsculas para el nombre, esto puede conducir a la corrupción de los índices.
`1`	Los nombres de tablas se almacenan en minúsculas en el disco y las comparaciones de nombre no son sensibles a mayúsculas. MySQL convierte todos los nombres de tablas a minúsculas para almacenamiento y búsquedas. En MySQL 5.0, este comportamiento también se aplica a nombres de bases de datos y alias de tablas. Este valor es el predeterminado en Windows y Mac OS X.
`2`	Los nombres de tablas y bases de datos se almacenan en disco usando el esquema de mayúsculas y minúsculas especificado en las sentencias `CREATE TABLE` o `CREATE DATABASE`, pero MySQL las convierte a minúsculas en búsquedas. Las comparaciones de nombres no son sensibles a mayúsculas. **Nota**: Esto funciona *solamente* en sistemas de ficheros que no son sensibles a mayúsculas. Los nombres de las tablas `InnoDB` se almacenan en minúsculas, como cuando `lower_case_table_names` vale 1.

En MySQL 5.0 para Windows y Mac OS X, el valor predeterminado de `lower_case_table_names` es 1.

Si se utiliza MySQL en una sola plataforma, normalmente no habrá que cambiar la variable `lower_case_table_names`. Sin embargo, se pueden encontrar dificultades si se desea transferir tablas entre plataformas cuyos sistemas de ficheros tengan diferente sensibilidad a mayúsculas. Por ejemplo, en Unix, se pueden tener dos tablas diferentes llamadas `mi_tabla` y `MI_TABLA`, pero en Windows, estos dos nombres se consideran idénticos. Para evitar problemas de transferencia de datos originados en la combinación de mayúsculas y minúsculas de los nombres de bases de datos y tablas, se tienen dos opciones:

- Emplear `lower_case_table_names=1` en todos los sistemas. La principal desventaja de esto es que al emplear `SHOW TABLES` o `SHOW DATABASES` no se verán los nombres en su combinación original de minúsculas y mayúsculas.
- Emplear `lower_case_table_names=0` en Unix y `lower_case_table_names=2` en Windows. Esto preserva la combinación de mayúsculas y minúsculas en los nombres de bases de datos y tablas. La desventaja es que hay que tener la precaución de que las consultas siempre se refieran a las bases de datos y tablas en Windows respetando la combinación correcta de mayúsculas y minúsculas. Si se transfirieran las consultas a Unix, donde las mayúsculas y minúsculas son significativas, no funcionarán si no se utiliza la combinación correcta.

  Excepción: Si se utilizan tablas `InnoDB`, se debería establecer `lower_case_table_names` en 1 en todas las plataformas para forzar a que los nombres sean convertidos a minúsculas.

Notar que antes de establecer `lower_case_table_names` en 1 en Unix, se deberán convertir a minúsculas los nombres de bases de datos y tablas existentes antes de reiniciar `mysqld`.

9.3. Variables de usuario

MySQL 5.0 soporta variables de usuario, las cuales permiten almacenar un valor y hacer referencia a él más tarde; esto posibilita pasar valores de una sentencia a otra. *Las variables de usuario son específicas de la conexión*. Esto significa que una variable definida por un cliente no puede ser vista o utilizada por otros clientes. Todas las variables de un cliente son automáticamente liberadas cuando ese cliente abandona la conexión.

Las variables de usuario se escriben como `@nombre_var`, donde el nombre de variable `nombre_var` puede consistir de caracteres alfanuméricos tomados del conjunto de caracteres actual, '.', '_', y '$'. El conjunto de caracteres predeterminado es ISO-8859-1 (Latin1). Esto puede cambiarse con la opción de `mysqld --default-character-set`. Consulte Sección 5.9.1, “El conjunto de caracteres utilizado para datos y ordenación”. Los nombres de variables de usuario no son sensibles a mayúsculas en MySQL 5.0.

Una forma de establecer una variable de usuario es empleando una sentencia `SET`:

```
SET @nombre_var = expr [, @nombre_var = expr] ...
```

Con `SET`, tanto `=` como `:=` pueden usarse como operadores de asignación. La `expr` asignada a cada variable puede evaluarse a un valor entero, real, cadena, o `NULL`.

Una variable de usuario también puede recibir valores en otras sentencias que no sean `SET`. En este caso, el operador de asignación debe ser `:=` y no `=` porque `=` se considera operador de comparación en otras sentencias que no sean `SET`:

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+----------------------+------+------+------+
| @t1:=(@t2:=1)+@t3:=4 | @t1  | @t2  | @t3  |
+----------------------+------+------+------+
|                    5 |    5 |    1 |    4 |
+----------------------+------+------+------+
```

Las variables de usuario pueden emplearse en cualquier parte donde se permitan expresiones. Generalmente esto no incluye situaciones donde explícitamente se requiere un valor literal, como en la cláusula `LIMIT` de una sentencia `SELECT`, o la cláusula `IGNORE número LINES` de una sentencia `LOAD DATA`.

Si se hace referencia a una variable aún sin inicializar, su valor será `NULL`.

En MySQL 5.0, si a una variable se le asigna un valor de cadena, el conjunto de caracteres y la forma de comparación (collation) de la variable serán forzados para ser iguales a los de la cadena. Este comportamiento es implícito desde MySQL 5.0.3 y lo mismo sucede con las columnas de una tabla.

Nota: en una sentencia `SELECT`, cada expresión se evalúa solamente cuando se envía al cliente. Esto significa que en una cláusula `HAVING`, `GROUP BY`, u `ORDER BY`, no es posible hacer referencia a una expresión que comprenda variables que reciben su valor en la lista del `SELECT`. Por ejemplo, la siguiente sentencia *no* funcionará como se espera:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

La referencia a `b` en la cláusula `HAVING` hace referencia al alias de una expresión de la lista `SELECT` que hace uso de la variable `@aa`. Esto no funciona como se esperaría: `@aa` no contiene el valor de la fila actual, sino el valor del `id` de la fila anteriormente seleccionada.

La regla general es que nunca se asigne un valor a una variable de usuario en una parte de una sentencia *y* se use la misma variable en otra parte de la misma sentencia. Se podrían obtener los resultados esperados, pero esto no está garantizado.

Otro problema asociado a asignar el valor de una variable y emplearla en la misma sentencia es que el tipo de dato resultante estará basado en el tipo que tenía la variable al comienzo de la sentencia. El siguiente ejemplo ilustra esto:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

En esta sentencia `SELECT`, MySQL informa al cliente que la primer columna es una cadena, y convierte todos los accesos a `@a` en cadenas, aún cuando @a recibe un valor numérico en la segunda línea. Luego de que la sentencia `SELECT` se ejecuta, `@a` se considera un número para la siguiente sentencia.

Para evitar problemas con este comportamiento, no se debe inicializar y utilizar la misma variable en la misma sentencia, o, de lo contrario, hay que establecer su valor en `0`, `0.0`, o `''` para definir su tipo antes de utilizarla.

Una variable sin asignación tiene el valor `NULL` con un tipo cadena.

9.4. Variables de sistema

MySQL proporciona acceso a muchas variables de sistema y de conexión. Muchas variables pueden modificarse dinámicamente mientras el servidor se está ejecutando. Esto a menudo permite variar la operación del servidor sin tener que detenerlo y reiniciarlo.

El servidor `mysqld` mantiene dos clases de variables. Las variables globales afectan la operación general del servidor. Las variables de sesión actúan sobre la operación en conexiones de clientes individuales.

Cuando el servidor arranca, inicializa todas las variables globales a sus valores predeterminados. Estos valores pueden ser modificados por opciones especificadas en ficheros de opciones o en la línea de comandos. Luego de que el servidor se inicia, las variables globales pueden ser modificadas dinámicamente conectándose y emitiendo una sentencia `SET GLOBAL var_name`. Para cambiar una variable global debe tenerse el privilegio `SUPER`.

El servidor también mantiene un conjunto de variables de sesión para cada cliente que se conecta. Las variables de sesión de cliente se inicializan al momento de conectarse, empleando el valor actual de la correspondiente variable global. Las variables de sesión dinámicas pueden ser modificadas por el cliente mediante una sentencia `SET SESSION var_name`. No se requieren privilegios especiales para establecer el valor una variable de sesión, pero un cliente puede modificar solamente sus propias variables, no las de otros clientes.

Un cambio en una variable global es visible para cualquier cliente que acceda esa variable. Sin embargo, afectará solamente a las correspondientes variables de sesión de las conexiones que se realicen luego del cambio. No afectará las variables de sesión de los clientes actualmente conectados (ni siquiera las del cliente que emitió la sentencia `SET GLOBAL`).

Los valores de las variables globales y de sesión pueden establecerse y recuperarse usando varias sintaxis diferentes. Los siguientes ejemplos están basados en la variable `sort_buffer_size`.

Para establecer el valor de una variable `GLOBAL`, debe emplearse una de las siguientes sintaxis:

```
mysql> SET GLOBAL sort_buffer_size=valor;
mysql> SET @@global.sort_buffer_size=valor;
```

Para establecer el valor de una variable SESSION, debe emplearse una de las siguientes sintaxis:

```
mysql> SET SESSION sort_buffer_size=valor;
mysql> SET @@session.sort_buffer_size=valor;
mysql> SET sort_buffer_size=valor;
```

LOCAL es un sinónimo de SESSION.

Si al establecer el valor de una variable no se utiliza GLOBAL, SESSION, o LOCAL, por defecto se asume SESSION. Consulte Sección 13.5.3, “Sintaxis de SET”.

Para recuperar el valor de una variable GLOBAL debe utilizarse una de las siguientes sentencias:

```
mysql> SELECT @@global.sort_buffer_size;
mysql> SHOW GLOBAL VARIABLES like 'sort_buffer_size';
```

Para recuperar el valor de una variable SESSION debe utilizarse una de las siguientes sentencias:

```
mysql> SELECT @@sort_buffer_size;
mysql> SELECT @@session.sort_buffer_size;
mysql> SHOW SESSION VARIABLES like 'sort_buffer_size';
```

Aquí, también, LOCAL es un sinónimo de SESSION.

Cuando se recupera una variable con SELECT @@*nombre_var* (o sea, no se especifica global., session., o local.), MySQL devuelve el valor de SESSION si existe y el valor GLOBAL en otro caso.

En el caso de SHOW VARIABLES, si no se especifica GLOBAL, SESSION, o LOCAL, MySQL devuelve los valores de SESSION.

La razón por la que la palabra clave GLOBAL se requiere para establecer el valor de variables que solamente existen como GLOBAL pero no para recuperar dicho valor, es para prevenir futuros problemas. Si se elimina una variable SESSION con el mismo nombre que una variable GLOBAL, un cliente con el privilegio SUPER podría cambiar accidentalmente la variable GLOBAL en lugar de hacerlo solamente sobre la variable SESSION de su propia conexión. Si se agrega una variable SESSION con el mismo nombre que una GLOBAL, un cliente que intentase modificar la variable GLOBAL podría encontrarse con que sólo se ha modificado su propia variable SESSION.

Puede encontrarse mayor información acerca de las opciones de inicio del sistema y de las variables de sistema en Sección 5.3.1, “Opciones del comando mysqld” y Sección 5.3.3, “Variables de sistema del servidor”. Una lista de las variables que pueden establecerse en tiempo de ejecución se brinda en Sección 5.3.3.1, “Variables de sistema dinámicas”.

9.4.1. Variables estructuradas de sistema

MySQL 5.0 también soporta variables de sistema estructuradas. Una variable estructurada difiere de una variable de sistema convencional en dos aspectos:

- Su valor es una estructura con componentes que especifican parámetros de servidor que se consideran estrechamente relacionados.
- Pueden existir varias instancias de un determinado tipo de variable estructurada. Cada una tiene un nombre diferente y se refiere a un recurso mantenido por el servidor.

Actualmente, MySQL soporta un solo tipo de variable estructurada. Éste especifica parámetros que regulan el funcionamiento de los cachés de claves (key caches). Una variable estructurada de caché de claves tiene estos componentes:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

Esta sección describe la sintaxis para referirse a variables estructuradas. Para los ejemplos de sintaxis se emplean variables de caché de claves, pero los detalles específicos sobre cómo funcionan los cachés de claves se encuentran en Sección 7.4.6, “La caché de claves de `MyISAM`”.

Para referirse a un componente de una instancia de una variable estructurada, se emplea un nombre compuesto con el formato `nombre_instancia.nombre_componente`. Ejemplos:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

Siempre hay predefinida una instancia con el nombre `default` para cada variable de sistema estructurada. Si se hace referencia a un componente de una variable estructurada sin mencionar el nombre de instancia, se utiliza `default`. Por lo tanto, `default.key_buffer_size` y `key_buffer_size` se refieren a la misma variable de sistema.

Las reglas para la denominación de instancias y componentes pertenecientes a variables estructuradas son las siguientes:

- Para un determinado tipo de variable estructurada, cada instancia debe tener un nombre que sea único *dentro* de ese tipo de variable. Sin embargo, los nombres de instancia no necesitan ser únicos *a través* de distintos tipos de variable estructurada. Por ejemplo, cada variable estructurada tiene una instancia llamada `default`, así que `default` no es único a través de distintos tipos de variable.
- Los nombres de los componentes de cada tipo de variable estructurada deben ser únicos a través de todos los nombres de variables de sistema. Si esto no fuese así (o sea, si dos tipos diferentes de variable estructurada compartiesen nombres de miembros), no sería posible determinar la variable estructurada por defecto a emplear cuando un nombre de miembro no estuviese precedido por un nombre de instancia.
- Si un nombre de instancia de variable estructurada no fuese legal al usarlo como identificador sin delimitar, habrá que referirse a él delimitándolo con acentos graves (ASCII 96). Por ejemplo, `hot-cache` no es un nombre legal, pero `` `hot-cache` `` lo es.
- `global`, `session`, y `local` no son nombres legales de instancia. Esto evita conflictos con notaciones del tipo `@@global.`*`nombre_var`*, que se utilizan para hacer referencias a variables de sistema no estructuradas.

Actualmente, las primeras dos reglas no tienen posibilidad de ser infringidas, porque el único tipo de variable estructurada es el empleado para cachés de claves. Estas reglas cobrarán mayor significado si en el futuro se crean otros tipos de variable estructurada.

Con una excepción, se puede hacer referencia a los componentes de una variable estructurada utilizando nombres compuestos en cualquier contexto en que puedan aparecer nombres simples de variable. Por ejemplo, se puede asignar un valor a una variable estructurada empleando una opción de línea de comandos:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

En un fichero de opciones, se utilizaría:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

Si se inicia el servidor con esta opción, crea un caché de claves llamado `hot_cache` con un tamaño de 64KB adicionalmente al caché por defecto, que tiene un tamaño predeterminado de 8MB.

Suponiendo que se inicie el servidor de esta manera:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

En este caso, el servidor establece el tamaño del caché de claves pedeterminado a 256KB. (También se podría haber escrito `--default.key_buffer_size=256K`.) Adicionalmente, el servidor crea un segundo caché llamado `extra_cache` con un tamaño de 128KB, y fija un tamaño de 2048 bytes para los buffers de bloque destinados al caché de bloques de índice de tablas.

El siguiente ejemplo inicia el servidor con tres diferentes cachés de claves, manteniendo sus tamaños en una proporción de 3:1:1:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Los valores de las variables estructuradas también pueden establecerse y leerse en tiempo de ejecución. Por ejemplo, para establecer a 10MB el tamaño de un caché de claves llamado `hot_cache`, pueden emplearse cualquiera de estas sentencias:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

Para obtener el tamaño del caché, se realiza lo siguiente:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

Sin embargo, las siguientes sentencias no funcionarán. La variable no es interpretada como un nombre compuesto, sino como una cadena proporcionada a `LIKE` para buscar coincidencias con un patrón.

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

Esta es la excepción mencionada sobre la posibilidad de utilizar en cualquier sitio un nombre de variable estructurada del mismo modo que se hace con una variable simple.

9.5. Sintaxis de comentarios

El servidor MySQL soporta tres estilos de comentario:

- Desde un carácter '#' hasta el fin de la línea.
- Desde una secuencia '`--` ' hasta el final de la línea. Nótese que el estilo '`--` ' (doble guión) requiere que luego del último guión haya por lo menos un espacio en blanco (espacio, tabulación nueva línea, etc.). Esta sintaxis difiere ligeramente de la sintaxis de comentarios SQL estándar, como se trata en Sección 1.7.5.7, “Empezar un comentario con '--'”.

- Desde una secuencia '`/*`' hasta la próxima secuencia '`*/`'. La secuencia de cierre no necesita estar en la misma línea, lo que permite tener comentarios que abarquen múltiples lineas.

El siguiente ejemplo muestra los tres estilos de comentario:

```
mysql> SELECT 1+1;     # Este comentario llega hasta el final de la línea
mysql> SELECT 1+1;     -- Este comentario llega hasta el final de la línea
mysql> SELECT 1 /* este es un comentario en línea (in-line) */ + 1;
mysql> SELECT 1+
/*
Este es un
comentario en múltiples líneas
*/
1;
```

La sintaxis de comentarios descrita se aplica a la forma en que el servidor `mysqld` procesa las sentencias SQL. El programa cliente `mysql` también realiza algún tipo de proceso de sentencias antes de enviarlas al servidor. (Por ejemplo, para determinar donde acaba cada sentencia en una línea que contiene varias de ellas).

En MySQL 5.0, la única limitación en la forma en que `mysql` procesa los comentarios `/* ... */` es que un signo admirativo o exclamativo utilizado con este estilo de comentario marca porciones de sentencias SQL de ejecución condicional. Esto se aplica cuando se ejecuta `mysql` interactivamente y cuando se colocan comandos en un fichero y se emplea `mysql` para procesar el fichero con `mysql < nom_fich`. Para más información y ejemplos, consulte Sección 1.7.4, “Extensiones MySQL al estándar SQL”.

9.6. Tratamiento de palabras reservadas en MySQL

Un problema común se origina cuando se intenta utilizar en un identificador tal como un nombre de tabla o columna el nombre de un tipo de dato o una función incorporados en MySQL, como `TIMESTAMP` o `GROUP`. Es posible hacer esto (por ejemplo, `ABS` es un nombre de columna válido). Sin embargo, por defecto, al invocar una función no se permiten espacios entre su nombre y el carácter '`(`' que le sigue. Este requisito permite distinguir entre una llamada a una función y una referencia a una columna con el mismo nombre.

Un efecto secundario de este comportamiento es que omitir un espacio en ciertos contextos provoca que un identificador sea interpretado como un nombre de función. Por ejemplo, esta sentencia es legal:

```
mysql> CREATE TABLE abs (val INT);
```

Pero al omitirse el espacio luego de `abs`, se produce un error de sintaxis porque la sentencia pasa a invocar la función `ABS()`:

```
mysql> CREATE TABLE abs(val INT);
```

Si el modo de servidor SQL incluye el valor `IGNORE_SPACE`, el servidor permite que las invocaciones a funciones tengan espacios en blanco entre el nombre de la función y el carácter '`(`' que le sigue. Esto convierte a los nombres de funciones en palabras reservadas. Como resultado, los identificadores que tienen similar nombre a una función deben ser delimitados por comillas como se describe en Sección 9.2, “Nombres de bases de datos, tablas, índices, columnas y alias”. Para controlar el modo de servidor SQL consulte Sección 5.3.2, “El modo SQL del servidor”.

Una palabra a continuación de un punto en un nombre calificado (por ejemplo nom_bd.nom_tabla) debe ser un identificador, por lo que no es necesario delimitarlo, incluso si es una palabra reservada.

Las palabras en la siguiente tabla están explícitamente reservadas en MySQL. La mayoría de ellas están prohibidas por el estándar SQL para ser empleadas como nombres de columnas y/o tablas (por ejemplo,

GROUP). Unas pocas son reservadas porque MySQL las necesita y (actualmente) utiliza un intérprete (parser) yacc. Una palabra reservada puede emplearse como identificador si se la delimita con comillas.

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	BEFORE
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
CONDITION	CONSTRAINT	CONTINUE
CONVERT	CREATE	CROSS
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_USER	CURSOR	DATABASE
DATABASES	DAY_HOUR	DAY_MICROSECOND
DAY_MINUTE	DAY_SECOND	DEC
DECIMAL	DECLARE	DEFAULT
DELAYED	DELETE	DESC
DESCRIBE	DETERMINISTIC	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	DUAL	EACH
ELSE	ELSEIF	ENCLOSED
ESCAPED	EXISTS	EXIT
EXPLAIN	FALSE	FETCH
FLOAT	FLOAT4	FLOAT8
FOR	FORCE	FOREIGN
FROM	FULLTEXT	GRANT
GROUP	HAVING	HIGH_PRIORITY
HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND
IF	IGNORE	IN
INDEX	INFILE	INNER
INOUT	INSENSITIVE	INSERT
INT	INT1	INT2
INT3	INT4	INT8
INTEGER	INTERVAL	INTO
IS	ITERATE	JOIN
KEY	KEYS	KILL
LEADING	LEAVE	LEFT
LIKE	LIMIT	LINES

LOAD	LOCALTIME	LOCALTIMESTAMP
LOCK	LONG	LONGBLOB
LONGTEXT	LOOP	LOW_PRIORITY
MATCH	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND
MINUTE_SECOND	MOD	MODIFIES
NATURAL	NOT	NO_WRITE_TO_BINLOG
NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUT
OUTER	OUTFILE	PRECISION
PRIMARY	PROCEDURE	PURGE
READ	READS	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SMALLINT
SONAME	SPATIAL	SPECIFIC
SQL	SQLEXCEPTION	SQLSTATE
SQLWARNING	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SSL	STARTING
STRAIGHT_JOIN	TABLE	TERMINATED
THEN	TINYBLOB	TINYINT
TINYTEXT	TO	TRAILING
TRIGGER	TRUE	UNDO
UNION	UNIQUE	UNLOCK
UNSIGNED	UPDATE	USAGE
USE	USING	UTC_DATE
UTC_TIME	UTC_TIMESTAMP	VALUES
VARBINARY	VARCHAR	VARCHARACTER
VARYING	WHEN	WHERE
WHILE	WITH	WRITE
XOR	YEAR_MONTH	ZEROFILL

Siguen las nuevas palabras reservadas de MySQL 5.0:

ASENSITIVE	CALL	CONDITION

CONTINUE	CURSOR	DECLARE
DETERMINISTIC	EACH	ELSEIF
EXIT	FETCH	INOUT
INSENSITIVE	ITERATE	LEAVE
LOOP	MODIFIES	OUT
READS	RELEASE	REPEAT
RETURN	SCHEMA	SCHEMAS
SENSITIVE	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
TRIGGER	UNDO	WHILE

MySQL permite que algunas palabras clave sean utilizadas como identificadores sin delimitar porque mucha gente las ha usado previamente. Por ejemplo:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

Capítulo 10. Soporte de conjuntos de caracteres

Tabla de contenidos

Este capítulo trata los siguientes temas:

- Qué son los conjuntos de caracteres y colaciones.
- El sistema por defecto de múltiples niveles
- Sintaxis para especificar el conjunto de caracteres y colaciones.
- Funciones y operaciones relacionadas.
- Soporte Unicode
- El significado de cada conjunto de caracteres individual y de cada colación

Los motores de almacenamiento `MyISAM`, `MEMORY`, e `InnoDB` soportan el conjunto de caracteres.

10.1. Conjuntos de caracteres y colaciones en general

Un **conjunto de caracteres** es un conjunto de símbolos y codificaciones. Una **colación** es un conjunto de reglas para comparar caracteres en un conjunto de caracteres. Vamos a dejar clara la distinción con un ejemplo de un conjunto de caracteres imaginario.

Supongamos que tenemos un alfabeto con cuatro letras: 'A', 'B', 'a', 'b'. Damos a cada letra un número: 'A' = 0, 'B' = 1, 'a' = 2, 'b' = 3. La letra 'A' es un símbolo, el número 0 es la **codificación** para 'A', y la combinación de las cuatro letras y sus codificaciones es un **conjunto de caracteres**.

Suponga que queremos comparar dos cadenas de caracteres, 'A' y 'B'. La forma más fácil de hacerlo es mirar las codificaciones: 0 para 'A' y 1 para 'B'. Ya que 0 es menor a 1, decimos que 'A' es menor que 'B'. Lo que acabamos de hacer es aplicar una colación a un conjunto de caracteres. La colación es un conjunto de reglas (sólo una en este caso): “compara las codificaciones”. LLamamos a la más sencilla de todas las colaciones una colación **binaria**.

Pero, ¿qué pasa si queremos decir que las letras en mayúsculas y minúsculas son equivalentes? Entonces tendríamos como mínimo dos reglas: (1) tratar las letras minúsuclas 'a' y 'b' como equivalentes a 'A' y 'B'; (2) luego comparar las codificaciones. Llamamos a esto una colación **no sensible a mayúsuculas y minúsculas (case-insensitive)**. Es un poco más compleja que una colación binaria.

En el mundo real, la mayoría de conjuntos de caracteres tienen varios caracteres: no sólo 'A' y 'B' sino alfabetos completos, a veces varios alfabetos o sistemas de escritura orientales con miles de caracteres, junto con muchos símbolos especiales y signos de puntuación. También en el mundo real, la mayoría de colaciones tienen muchas reglas: no sólo distinción entre mayúsculas y minúsculas, sino también sensibilidad a tildes (una “tilde” es una marca añadida a un carácter como en alemán 'Ö') y mapeos de múltiples caracteres (tales com la regla que 'Ö' = 'OE' en una de las dos colaciones alemanas).

MySQL 5.0 puede hacer lo siguiente:

- Guardar cadenas de caracteres usando una variedad de conjuntos de caracteres
- Comparar cadenas de caracteres usando una variedad de colaciones
- Mezclar cadenas de caracteres con distintos conjuntos de caracteres o colaciones en el mismo servidor, la misma base de datos, o incluso la misma tabla
- Permitir la especificación de un conjunto de caracteres y una colación en cualquier nivel

En este aspecto, no sólo MySQL 5.0 es mucho más flexible que versiones anteriores de MySQL, sino mejor que otros SGBDs. Sin embargo, para usar estas características de forma correcta, necesita saber qué conjuntos de caracteres y colaciones están disponibles, cómo cambiar los valores por defecto, y cómo afectan al comportamiento de operadores de cadenas de caracteres y funciones.

10.2. Conjuntos de caracteres y colaciones en MySQL

El servidor MySQL soporta varios conjuntos de caracteres. Para listar los disponibles, use el comando `SHOW CHARACTER SET`:

```
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   | ISO 8859-1 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 |
...
```

(Para un listado completo, consulte Sección 10.10, “Conjuntos de caracteres y colaciones que soporta MySQL”.)

Cualquier conjunto de caracteres tiene siempre como mínimo una colación, aunque puede tener varias.

Para listar las colaciones para un conjunto de caracteres, use el comando `SHOW COLLATION`. Por ejemplo, para ver las colaciones para el conjunto de caracteres `latin1` (“ISO-8859-1 Europa Occidental”), use el siguiente comando, que muestra el nombre de las colaciones que empiezan con `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 |
+--------------------+---------+----+---------+----------+---------+
```

Las colaciones para `latin1` tienen los siguientes significados:

Colación	Significado
`latin1_german1_ci`	Alemán DIN-1
`latin1_swedish_ci`	Sueco/Finlandés
`latin1_danish_ci`	Danés/Noruego
`latin1_german2_ci`	Alemán DIN-2
`latin1_bin`	Binario según codificación `latin1`
`latin1_general_ci`	Multilingüe (Europa Occidental)
`latin1_general_cs`	Multilingüe (ISO Europa Occidental), sensible a mayúsculas
`latin1_spanish_ci`	Español moderno

Las colaciones tienen las siguientes características generales:

- Dos conjuntos de caracteres distintos no pueden tener la misma colación.
- Cada conjunto de caracteres tiene una colación que es la *colación por defecto*. Por ejemplo, la colación por defecto para `latin1` es `latin1_swedish_ci`.
- Hay una convención para nombres de colaciones: empiezan con el nombre del conjunto de caracteres al que están asociados, normalmente incluyen el nombre del idioma, y acaban con `_ci` (no distingue entre mayúsculas y minúsculas), `_cs` (distingue entre mayúsculas y minúsculas), o `_bin` (binario).

10.3. Determinar el conjunto de caracteres y la colación por defecto

Hay configuraciones por defecto para conjuntos de caracteres y colaciones en cuatro niveles: servidor, base de datos, tabla, y conexión. La siguiente descripción puede parecer compleja, pero en la práctica este funcionamiento de varios niveles conduce a resultados naturales y obvios.

10.3.1. Conjunto de caracteres y colación del servidor

El servidor MySQL tiene un conjunto de caracteres para el servidor y una colación, y ambos deben ser distintos al valor nulo.

MySQL determina el conjunto de caracteres y la colación del servidor con el siguiente procedimiento:

- Según las opciones en efecto cuando arranca el servidor
- Según los valores cambiados en tiempo de ejecución

A nivel de servidor, la decisión es simple. El conjunto de caracteres y la colación del servidor dependen inicialmente de la opción usada al arrancar `mysqld`. Se puede usar `--default-character-set` para el conjunto de caracteres y a éste se puede añadir `--default-collation` para la colación. No especificar un conjunto de caracteres, es como especificar `--default-character-set=latin1`. Especificar sólo un conjunto de caracteres (por ejemplo, `latin1`) pero no una colación, es como especificar `--default-charset=latin1 --default-collation=latin1_swedish_ci`, ya que `latin1_swedish_ci` es la colación por defecto para `latin1`. Por lo tanto, los siguientes tres comandos tienen el mismo efecto:

```
shell> mysqld
shell> mysqld --default-character-set=latin1
shell> mysqld --default-character-set=latin1 \
           --default-collation=latin1_swedish_ci
```

Una forma de cambiar la especificación es recompilando. Para cambiar el conjunto de caracteres por defecto y la colación al compilar las fuentes, debe utilizarse: `--with-charset` y `--with-collation` como argumentos para `configure`. Por ejemplo:

```
shell> ./configure --with-charset=latin1
```

O:

```
shell> ./configure --with-charset=latin1 \
           --with-collation=latin1_german1_ci
```

Tanto `mysqld` como `configure` verifican que la combinación del conjunto de caracteres y la colación es válida. Si no lo es, cada programa muestra un mensaje de error y acaba.

El conjunto de caracteres y la colación actuales están disponibles como los valores de las variables `character_set_server` y `collation_server` . Estas variables pueden cambiarse en tiempo de ejecución.

10.3.2. Conjuntos de caracteres y colaciones de la base de datos

Cada base de datos tiene un conjunto de caracteres y una colación que no pueden ser nulos. Los comandos `CREATE DATABASE` y `ALTER DATABASE` tienen cláusulas opcionales para especificar el conjunto de caracteres y colación de la base de datos:

```
CREATE DATABASE nombre_de_base_de_datos
    [[DEFAULT] CHARACTER SET nombre_de_conjunto_de_caracteres]
    [[DEFAULT] COLLATE nombre_de_colación]

ALTER DATABASE nombre_de_base_de_datos
    [[DEFAULT] CHARACTER SET nombre_de_conjunto_de_caracteres]
    [[DEFAULT] COLLATE nombre_de_colación]
```

Ejemplo:

```
CREATE DATABASE nombre_de_base_de_datos
    DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL elige el conjunto de caracteres y colación de la base de datos así:

- Si tanto `CHARACTER SET X` como `COLLATE Y` se especifican, entonces el conjunto de caracteres es *X* y la colación *Y*.
- Si `CHARACTER SET X` se especifica sin `COLLATE`, entonces el conjunto de caracteres es *X* y la colación es la de defecto.
- En el resto de casos, es el conjunto de caracteres y la colación del servidor.

La sintaxis de MySQL `CREATE DATABASE ... DEFAULT CHARACTER SET ...` es análoga a la sintaxis estándar SQL `CREATE SCHEMA ... CHARACTER SET ...`. Por ello, es posible crear bases de datos con distintos conjuntos de caracteres y colaciones en el mismo servidor MySQL.

El conjunto de caracteres de la base de datos y la colación se usan como valores por defecto para una tabla si no se especifica el conjunto de caracteres y colación en el comando `CREATE TABLE`. No tienen otro propósito.

El conjunto de caracteres y colación para la base de datos por defecto están disponibles como los valores de las variables `character_set_database` y `collation_database` . El servidor obtiene estas variables siempre que la base de datos por defecto cambia. Si no hay base de datos por defecto, las variables tienen el mismo valor que las variables correspondiente del lado del servidor, `character_set_server` y `collation_server`.

10.3.3. Conjunto de caracteres y colación de tabla

Cada tabla tiene un conjunto de caracteres y colación que no pueden ser nulas. Los comandos `CREATE TABLE` y `ALTER TABLE` tienen cláusulas opcionales para especificar el conjunto de caracteres y la colación:

```
CREATE TABLE nombre_de_tabla (lista_de_columnas)
    [DEFAULT CHARACTER SET nombre_de_conjunto_de_caracteres [COLLATE nombre_de_colación]]

ALTER TABLE nombre_de_tabla
    [DEFAULT CHARACTER SET nombre_de_conjunto_de_caracteres] [COLLATE nombre_de_colación]
```

Ejemplo:

```
CREATE TABLE t1 ( ... )
    DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL elije el conjunto de caracteres y colación de la siguiente forma:

- Si se especifican `CHARACTER SET X` y `COLLATE Y` , entonces el conjunto de caracteres es *X* y la colación *Y*.
- Si se especifica `CHARACTER SET X` sin `COLLATE`, el conjunto de caracteres es *X* y la colación es la de defecto.
- En cualquier otro caso, el conjunto de caracteres y colación son las del servidor.

El conjunto de caracteres y colación de la tabla se usan como valores por defecto si el conjunto de caracteres y la colación no se especifican en la definición de las columnas. El conjunto de caracteres de la tabla y la colación son extensiones MySQL; no hay mucho al respecto en el estándar SQL.

10.3.4. Conjunto de caracteres y colación de columnas

Cada columna “carácter” (esto es, una columna de tipo `CHAR`, `VARCHAR`, o `TEXT`) tiene un conjunto de caracteres y colación de columna, que no pueden ser nulos. La sintaxis de definición de columnas tiene cláusulas opcionales para especificar el conjunto de caracteres y la colación:

```
nombre_de_columna {CHAR | VARCHAR | TEXT} (ancho_de_columna)
    [CHARACTER SET nombre_de_conjunto_de_caracteres [COLLATE nombre_de_colación]]
```

Ejemplo:

```
CREATE TABLE Table1
(
    column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci
);
```

MySQL elige el conjunto de caracteres y la colación de la columna de la siguiente forma:

- Si se especifican `CHARACTER SET X` y `COLLATE Y` , entonces el conjunto de caracteres es *X* y la colación *Y*.
- Si se especifica `CHARACTER SET X` sin `COLLATE`, el conjunto de caracteres es *X* y la colación es la de defecto.
- En cualquier otro caso, se usan el conjunto de caracteres y la colación de la tabla.

Las cláusulas `CHARACTER SET` y `COLLATE` son de SQL estándar.

10.3.5. Ejemplos de asignación de conjunto de caracteres y colación

Los siguientes ejemplos muestran cómo MySQL determina los valores del conjunto de caracteres y colación por defecto.

Ejemplo 1: Definición de tabla y columna

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Aquí tenemos una columna con un conjunto de caracteres `latin1` y una colación `latin1_german1_ci` . La definición es explícita, así que es sencillo. Debe tenerse en cuenta que no hay problema al almacenar una columna `latin1` en una tabla `latin2`.

Ejemplo 2: Definición de tabla y columna

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

Ahora tenemos una columna con un conjunto de caracteres `latin1` y la colación por defecto. Aunque puede parecer normal, la colación por defecto no es la del nivel de tabla. En lugar de ello, ya que la colación para `latin1` siempre es `latin1_swedish_ci`, la columna `c1` tiene una colación de `latin1_swedish_ci` (no `latin1_danish_ci`).

Ejemplo 3: Definición de tabla y columna

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

Tenemos una columna con un conjunto de caracteres y colación por defecto. Bajo esta circunstancia, MySQL busca a nivel de tabla para determinar el conjunto de caracteres y colación para la columna. Así, el conjunto de caracteres para la columna `c1` es `latin1` y su colación es `latin1_danish_ci`.

Ejemplo 4: Definición de base de datos, tabla y columna

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

Creamos una columna sin especificar su conjunto de caracteres y colación. Tampoco especificamos un conjunto de caracteres y colación a nivel de tabla. En este caso, MySQL busca a nivel de base de datos para inspirarse. (La configuración de la base de datos pasa a ser la configuración de la tabla, y posteriormente la configuración de columna). Así, el conjunto de caracteres para la columna `c1` es `latin2` y su colación es `latin2_czech_ci`.

10.3.6. Conjunto de caracteres y colación de la conexión

Varias variables de sistema de conjunto de caracteres y colaciones están relacionadas con la interacción cliente-servidor. Algunas de ellas se han mencionado en secciones anteriores:

- El conjunto de caracteres y la colación del servidor están disponibles como los valores de las variables `character_set_server` y `collation_server` .
- El conjunto de caracteres y la colación de la base de datos por defecto están disponibles como valores de las variables `character_set_database` y `collation_database` .

Variables de conjuntos de caracteres y colaciones adicionales están involucradas en tratar el tráfico para la conexión entre un cliente y el servidor. Cada cliente tiene variables para el conjunto de caracteres y colación relacionados con la conexión.

Una “conexión” es lo que se hace al conectarse con el servidor. El cliente envía comandos SQL, tales como consultas, mediante la conexión al servidor. El servidor envía respuestas, tales como resultados, mediante la conexión de vuelta al cliente. Esto genera preguntas sobre tratamiento de conjuntos de caracteres y colaciones para conexiones de cliente, cada una de ellas puede responderse mediante variables de sistema:

- ¿ Qué conjunto de caracteres usa una consulta al salir del cliente?

  El servidor toma la variable `character_set_client` para usarla en las consultas enviadas por el cliente.

- ¿ Qué conjunto de caracteres debería usar el servidor para traducir una consulta tras recibirla?

  Para esto, el servidor usa `character_set_connection` y `collation_connection`. Esto convierte las consultas enviadas por el cliente de `character_set_client` a `character_set_connection` (excepto para cadenas de caracteres literales que tienen un introductor como `_latin1` o `_utf8`). `collation_connection` es importante para comparaciones de cadenas de caracteres literales. Para comparaciones de cadenas de caracteres con valores de columnas no importa, ya que las columnas tienen una precedencia mayor en las colaciones.

- ¿ Qué conjunto de caracteres debería usar el servidor para traducir los resultados o errores antes de enviar el mensaje de vuelta al cliente?

  La variable `character_set_results` indica el conjunto de caracteres usado por el servidor para devolver los resultados de las consultas al cliente. Esto incluye datos resultantes como los valores de las columnas, y metadatos resultantes como nombres de columnas.

Puede ajustar estas variables, o puede depender de los valores por defecto (en tal caso, puede obviar esta sección).

Hay dos comandos que afectan al conjunto de caracteres de conexión:

```
SET NAMES 'nombre_de_conjunto_de_caracteres'
SET CHARACTER SET nombre_de_conjunto_de_caracteres
```

`SET NAMES` indica qué hay en el comando SQL que envía el cliente. Por lo tanto, `SET NAMES 'cp1251'` le dice al servidor “los próximos mensajes entrantes de este cliente están en el conjunto de caracteres `cp1251`.” También especifica el conjunto de caracteres para los resultados que el servidor devuelve al cliente. (Por ejemplo, indica los conjuntos de caracteres de la columna si usa el comando `SELECT` .)

Un comando `SET NAMES 'x'` es equivalente a estos tres comandos:

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET character_set_connection = x;
```

Cambiar `character_set_connection` a `x` también cambia `collation_connection` de la colación por defecto a `x`.

`SET CHARACTER SET` es similar pero cambia el conjunto de caracteres y la colación para la conexión para ser las de la base de datos por defecto. Un comando `SET CHARACTER SET x` es equivalente a estos tres comandos:

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET collation_connection = @@collation_database;
```

Cuando un cliente se conecta, envía al servidor el nombre del conjunto de caracteres que quiere usar. El servidor cambia las variables `character_set_client`, `character_set_results`, y `character_set_connection` para ese conjunto de caracteres. (De hecho, el servidor efectúa una operación `SET NAMES` usando el conjunto de caracteres).

No es necesario ejecutar `SET NAMES` cada vez que se arranca el cliente `mysql`, aunque se quiera utilizar un conjunto de caracteres diferente del que hay por defecto. Puede añadirse la opción `--default-character-set` en la línea de comandos de `mysql`, o en el fichero de opciones. Por ejemplo, el siguiente fichero de opciones cambia las variables del conjunto de caracteres a `koi8r` cada vez que se ejecuta `mysql`:

```
[mysql]
default-character-set=koi8r
```

Ejemplo: Supongamos que `column1` se define como `CHAR(5) CHARACTER SET latin2`. Si no se especifica `SET NAMES` o `SET CHARACTER SET`, entonces para `SELECT column1 FROM t`, el servidor devuelve todos los valores para `column1` usando el conjunto de caracteres que el cliente especificó al conectar. Por otro lado, si se especifica `SET NAMES 'latin1'` o `SET CHARACTER SET latin1`, entonces justo antes de devolver los resultados, el servidor convierte el valor `latin2` a `latin1`. La conversión puede perder información si hay caracteres que no están en ambos conjuntos de caracteres.

Si no se quiere que el servidor haga ninguna conversión, debe cambiarse `character_set_results` a `NULL`:

```
mysql> SET character_set_results = NULL;
```

10.3.7. Conjunto de caracteres y colación de columnas “carácter”

Cada literal de cadenas de caracteres tiene un conjunto de caracteres y una colación, que no pueden ser nulos.

Un literal de cadena de caracteres puede tener un introductor de conjunto de caracteres opcional y una cláusula `COLLATE`:

```
[_nombre_de_conjunto_de_caracteres]'cadena_de_texto' [COLLATE nombre_de_colación]
```

Ejemplos:

```
SELECT 'cadena_de_texto';
SELECT _latin1'cadena_de_texto';
SELECT _latin1'cadena_de_texto' COLLATE latin1_danish_ci;
```

Para el comando simple `SELECT 'cadena_de_texto'`, la cadena de caracteres tiene el conjunto de caracteres y colación definida por las variables de sistema `character_set_connection` y `collation_connection`.

La expresión *`_nombre_de_conjunto_de_caracteres`* se llama formalmente *introductor*. Le dice al parser: “la siguiente cadena de caracteres utiliza el conjunto de caracteres *`X`*”. Esto ha sido fuente de confusión en el pasado. Enfaticemos pues que un introductor no causa ninguna conversión, sino que es estrictamente una señal que no cambia el valor de la cadena de caracteres. Un introductor también es legal antes de la notación para literales hexadecimales y literales numéricos hexadecimales (`x'literal'` y `0xnnnn`), y antes de `?` (parámetros de sustitución al usar comandos preparados dentro de la interfície de un lenguaje de programación).

Ejemplos:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 ?;
```

MySQL determina el conjunto de caracteres y colación para un literal así:

- Si se especifican *_X* y COLLATE *Y* , entonces el conjunto de caracteres es *X* y se usa la colación *Y*.
- Si se especifica *_X* pero no se especifica COLLATE , entonces el conjunto de caracteres es *X* y el conjunto de caracteres es el de defecto.
- En cualquier otro caso, se utilizan el conjunto de caracteres y la colación que hay en las variables de sistema character_set_connection y collation_connection .

Ejemplos:

- Una cadena de caracteres con conjunto de caracteres latin1 y colación latin1_german1_ci :

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- Una cadena de caracteres con conjunto de caracteres latin1 y su colación por defecto (esto es, latin1_swedish_ci):

```
SELECT _latin1'Müller';
```

- Una cadena de caracteres con conjunto de caracteres y colación por defecto de la conexión:

```
SELECT 'Müller';
```

Los introductores de conjunto de caracteres y la cláusula COLLATE se implementan según las especificaciones del estándar SQL.

10.3.8. Usar COLLATE en sentencias SQL

Con la cláusula COLLATE, puede obviarse la colación por defecto para comparación. COLLATE puede usarse en varias partes de los comandos SQL. Aquí se muestran algunos ejemplos:

- Con ORDER BY:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- Con AS:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- Con GROUP BY:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- Con funciones agregadas:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- Con DISTINCT:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- Con WHERE:

```
    SELECT *
    FROM t1
    WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
    SELECT *
    FROM t1
    WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- Con HAVING:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.3.9. Precedencia de la cláusula COLLATE

La cláusula COLLATE tiene alta precedencia (mayor que ||), así que las siguientes dos expresiones son equivalentes:

```
x || y COLLATE z
x || (y COLLATE z)
```

10.3.10. Operador BINARY

El operador BINARY es una abreviación de la cláusula COLLATE. BINARY *'x'* es equivalente a *'x'* COLLATE *y*, donde *y* es el nombre de la colación binaria para el conjunto de caracteres de '*x*'. Cada conjunto de caracteres tiene una colación binaria. Por ejemplo, la colación binaria para el conjunto de caracteres latin1 es latin1_bin, así si la columna a es del conjunto de caracteres latin1, los siguientes dos comandos tienen el mismo efecto:

```
SELECT * FROM t1 ORDER BY BINARY a;
SELECT * FROM t1 ORDER BY a COLLATE latin1_bin;
```

10.3.11. Casos especiales en los que determinar la colación es complicado

En la gran mayoría de consultas, resulta obvio qué colación usa MySQL para resolver una operación de comparación. Por ejemplo, en los siguientes casos, debe quedar claro que la colación es “la colación de la columna x”:

```
SELECT x FROM T ORDER BY x;
```

```
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

Sin embargo, cuando están implicados varios operandos, puede haber ambigüedad. Por ejemplo:

```
SELECT x FROM T WHERE x = 'Y';
```

¿Esta consulta debe usar la colación de la columna `x`, o de la columna de caracteres literal `'Y'`?

SQL estándar resuelve tales cuestiones usando lo que se solía llamar reglas “coercitivas”. Es decir: Como `x` e `'Y'` tienen colaciones, ¿cuál tiene precedencia? Puede ser difícil de resolver, pero las siguientes reglas resuelven la mayoría de situaciones:

- Una cláusula `COLLATE` explícita tiene una coercibilidad de 0. (No es coercible en absoluto.)
- La concatenación de dos cadenas de caracteres con diferentes colaciones tiene una coercibilidad de 1.
- La colación de una columna tiene una coercibilidad de 2.
- Una “constante de sistema” (la cadena de caracteres retornada por funciones como `USER()` o `VERSION()`) tiene una coercibilidad de 3.
- Una colación de un literal tiene una coercibilidad de 4.
- `NULL` o una expresión derivada de `NULL` tiene una coercibilidad de 5.

Los valores de coercibilidad precedentes son los de MySQL 5.0.3. Consúltese la nota posterior en esta sección para más información relacionada con versiones.

Estas reglas resuelven ambigüedades como:

- Uso de la colación con el valor más bajo de coercibilidad.
- Si ambos operadodres tienen la misma coercibilidad, entonces hay un error si las colaciones son distintas.

Ejemplos:

`columna1 = 'A'`	Usa colación de `columna1`
`columna1 = 'A' COLLATE x`	Usa colación de `'A'`
`columna1 COLLATE x = 'A' COLLATE y`	Error

La función `COERCIBILITY()` puede usarse para determinar la coercibilidad de una expresión de cadena de caracteres:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
mysql> SELECT COERCIBILITY('A');
        -> 4
```

Consulte Sección 12.9.3, “Funciones de información”.

En MySQL 5.0 antes de la versión 5.0.3, no hay constantes de sistema o coercibilidad ignorables. Funciones como `USER()` tienen una coercibilidad de 2 en lugar de 3, y los literales tienen una coercibilidad de 3 en lugar de 4.

10.3.12. A cada colación un conjunto de caracteres correcto

Recuerde que cada conjunto de caracteres tiene una o más colaciones, y cada colación está asociadad con uno y sólo un conjunto de caracteres. Por lo tanto, los siguientes comandos causan un mensaje de error ya que la colación `latin2_bin` no es legal con el conjunto de caracteres `latin1`:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1251: COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.3.13. Un ejemplo del efecto de una colación

Supongamos que la columna `X` en la tabla `T` tiene estos valores de columna `latin1`:

```
Muffler
Müller
MX Systems
MySQL
```

Y supongamos que los valores de la columna se retornan usando los siguientes comandos:

```
SELECT X FROM T ORDER BY X COLLATE nombre_de_colación;
```

El orden resultante de los valores para diferentes colaciones se muestra en esta tabla:

`latin1_swedish_ci`	`latin1_german1_ci`	`latin1_german2_ci`
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

La tabla es un ejemplo que muestra el efecto de la utilización de diferentes colaciones en una cláusula `ORDER BY`. El carácter que causa el orden de ordenación diferente en este ejemplo es la U con dos puntos encima (ü), que los alemanes llaman "U-umlaut".

- La primera columna muestra el resultado de `SELECT` usando la regla de colación Sueca/Finlandesa, que dice que U-umlaut se ordena con Y.
- La segunda columna muestra el resultado de `SELECT` usando la regla alemana DIN-1, que dice que U-umlaut se ordena con U.
- La tercera columna muestra el resultado de `SELECT` usando la regla alemana DIN-2, que dice que U-umlaut se ordena con UE.

10.4. Efectos del soporte de conjuntos de caracteres

Esta sección describe operaciones que tienen en cuenta información sobre el conjunto de caracteres en MySQL 5.0.

10.4.1. Cadenas de caracteres de resultado

MySQL tiene varios operadores y funciones que retornan una cadena de caracteres. Esta sección responde a la pregunta: ¿Cuál es el conjunto de caracteres y colación de esta cadena de caracteres?

Para funciones simples que toman una entrada de cadenas de caracteres y devuelven una cadena de caracteres como salida, el conjunto de caracteres y colación de la salida son las mismas que las de la entrada. Por ejemplo `UPPER(X)` devuelve una cadena de caracteres con un conjunto de caracteres y colación iguales a las de *X*. Lo mismo se aplica para `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, y `UPPER()`. (Debe tenerse en cuenta que la función `REPLACE()`, a diferencia del resto de funciones, ignora siempre la colación de la cadena de caracteres de entrada y realiza una comparación que no distingue entre mayúsculas y minúsculas).

Para operaciones que combinan varias cadenas de caracteres de entrada y devuelven una única cadena de salida, se aplica la “regla de agregación”estándar de SQL:

- Si hay un `COLLATE X` explícito, usa *X*.
- Si hay un `COLLATE X` y `COLLATE Y` explícitos, muestra un error.
- De lo contrario, si todas las colaciones son *X*, usa *X*.
- En cualquier otro caso, el resultado no tiene colación.

Por ejemplo, con `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, la colación resultante es *X*. Lo mismo se aplica para `CASE`, `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, y `LEAST()`.

Para operaciones que conviertan datos de tipo carácter, el conjunto de caracteres y colación de las cadenas de caracteres resultantes de las operaciones se definen por las variables de sistema `character_set_connection` y `collation_connection` . Esto se aplica en `CAST()`, `CHAR()`, `CONV()`, `FORMAT()`, `HEX()`, y `SPACE()`.

10.4.2. CONVERT()

`CONVERT()` proporciona un modo de convertir datos entre distintos conjuntos de caracteres. La sintaxis es:

```
CONVERT(expr USING transcoding_name)
```

En MySQL, "transcoding names" es lo mismo que los conjuntos de caracteres correspondientes.

Ejemplos:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` se implementa según la especificación de SQL estándar.

En modo SQL `TRADITIONAL`, si convierte una cadena de caracteres de fecha “zero” en una fecha, `CONVERT()` retorna `NULL`. MySQL 5.0.4 y versiones posteriores también muestran un mensaje de advertencia.

10.4.3. CAST()

Puede usar `CAST()` para convertir una cadena de caracteres a un conjunto de caracteres distinto. La sintaxis es:

```
```

```
CAST(cadena_de_caracteres AS tipo_de_datos CHARACTER SET nombre_de_conjunto_de_caracteres)
```

Ejemplo:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

Si usa `CAST()` sin especificar `CHARACTER SET`, el conjunto de caracteres y colación resultantes se definen por las variables de sistema `character_set_connection` y `collation_connection` . Si usa `CAST()` con `CHARACTER SET X`, el conjunto de caracteres y colación resultantes son `X` y la colación por defecto de `X`.

Puede no usar una cláusula `COLLATE` dentro de `CAST()`, pero puede hacerlo fuera. Esto es, `CAST(... COLLATE ...)` es ilegal, pero `CAST(...) COLLATE ...` es legal.

Ejemplo:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

En modo SQL `TRADITIONAL`, si convierte una cadena de caracteres de fecha “zero” a una fecha, `CAST()` retorna `NULL`. MySQL 5.0.4 y versiones posteriores muestran un mensaje de advertencia.

10.4.4. Sentencias SHOW

Varios comandos `SHOW` proporcionan información adicional de conjuntos de caracteres. Esto incluye `SHOW CHARACTER SET`, `SHOW COLLATION`, `SHOW CREATE DATABASE`, `SHOW CREATE TABLE` and `SHOW COLUMNS`.

El comando `SHOW CHARACTER SET` muestra todos los conjuntos de caracteres disponibles. Tiene una cláusula opcional `LIKE` que indica con qué nombres de conjunto de caracteres comparar. Por ejemplo:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | ISO 8859-1 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 |
+---------+-----------------------------+-------------------+--------+
```

Consulte Sección 13.5.4.1, “Sintaxis de SHOW CHARACTER SET”.

La salida de `SHOW COLLATION` incluye todos los conjuntos de caracteres. Tiene una cláusula opcional `LIKE` que indica con qué nombres de conjunto de caracteres comparar. Por ejemplo:

```
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 |
```

```
+-------------------+---------+----+---------+----------+---------+
```

Consulte Sección 13.5.4.2, “Sintaxis de SHOW COLLATION”.

SHOW CREATE DATABASE muestra el comando CREATE DATABASE que crea una base de datos dada. El resultado incluye todas las opciones de la base de datos DEFAULT CHARACTER SET y COLLATE están soportadas. Todas las opciones de las bases de datos se guardan en un fichero de datos llamado db.opt que se encuentra en el directorio de la base de datos.

```
mysql> SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

Consulte Sección 13.5.4.4, “Sintaxis de SHOW CREATE DATABASE”.

SHOW CREATE TABLE es similar, pero muestra el comando CREATE TABLE para crear una tabla dada. Las definiciones de columnas indican cualquier especificación del conjunto de caracteres, y las opciones de tabla incluyen información del conjunto de caracteres.

Consulte Sección 13.5.4.5, “Sintaxis de SHOW CREATE TABLE”.

El comando SHOW COLUMNS muestra la colación de las columnas de una tabla cuando se invoca como SHOW FULL COLUMNS. Columnas con tipos de datos CHAR, VARCHAR, o TEXT tienen colaciones no-NULL . Tipos numéricos y otros tipos no-carácter tiene colaciones NULL. Por ejemplo:

```
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:
*************************** 2. row ***************************
     Field: name
      Type: char(60)
 Collation: latin1_swedish_ci
      Null: NO
       Key:
   Default:
     Extra:
Privileges: select,insert,update,references
   Comment:
```

El conjunto de caracteres no es parte de lo que se muestra. (El nombre del conjunto de caracteres está implícito en el nombre de la colación.)

Consulte Sección 13.5.4.3, “Sintaxis de SHOW COLUMNS”.

10.5. Soporte Unicode

En MySQL 5.0, hay dos conjuntos de caracteres para guardar datos Unicode:

- `ucs2`, el conjunto de caracteres UCS-2 Unicode.
- `utf8`, la codificación UTF8 del conjunto de caracteres Unicode.

En UCS-2 (representación binaria de Unicode), cada carácter se representa con un código de dos bytes Unicode con el byte más significativo primero. Por ejemplo: "LETRA LATINA MAYÚSCULA A" tiene código 0x0041 y se almacena como una secuencia de dos bytes: 0x00 0x41. "LETRA MINÚSCULA CIRÍLICA YERU" (Unicode 0x044B) se almacenca como la secuencia de dos bytes: 0x04 0x4B. Para caracteres Unicode y sus códigos, consulte Unicode Home Page.

Actualmente, UCS-2 no puede usarse como un conjunto de caracteres cliente, lo que significa que `SET NAMES 'ucs2'` no funciona.

El conjunto de caracteres UTF8 (transformación de la representación Unicode) es una forma alternativa de guardar los datos Unicode. Está implementado según el RFC 3629. La idea del conjunto de caracteres UTF8 es que varios caracteres Unicode se codifican usando secuencias de bytes de longitudes diferentes:

- Letras básicas latinas, dígitos y signos de puntuación usan un byte.
- La mayoría de letras europeas y del Medio Oriente se guardan en una secuencia de dos bytes: letras latinas extendidas (con tilde, diéresis, etc), cirílico, griego, armenio, hebreo, arábigo, sirio, y otros.
- Ideogramas koreanos, chinos, y japoneses usan secuencias de tres bytes.

RFC 3629 describe secuencias de codificación que usan de uno a cuatro bytes. Actualmente, el soporte MySQL UTF8 no incluye secuencias de cuatro bytes. (Un estándar antiguo de codificación UTF8 se da en RFC 2279, que describe secuencias UTF8 que usan de uno a seis bytes. RFC 3629 sustituye al obsoleto RFC 2279; por esta razón, las secuencias con cinco y seis bytes ya no se usan).

Consejo: para ahorrar espacio con UTF8, use `VARCHAR` en lugar de `CHAR`. De otro modo, MySQL tiene que reservar 30 bytes para una columna `CHAR(10) CHARACTER SET utf8`, ya que es la longitud máxima posible.

10.6. UTF8 para metadatos

Los metadatos son los datos acerca de los datos. Cualquier cosa que describa la base de datos, opuestamente a los contenidos de la misma, son metadatos. Así, los nombres de columna, nombres de base de datos, nombres de usuario, nombres de versión y la mayoría de resultados de `SHOW` son metadatos.

La representación de los metadatos debe satisfacer estos requerimientos:

- Todos los metadatos deben ser del mismo conjunto de caracteres. De otro modo `SHOW` no funcionaría correctamente ya que distintos registros en la misma columna tendrían distintos conjuntos de caracteres.
- Los metadatos deben incluir todos los caracteres en todos los idiomas. De otro modo, los usuarios no serían capaces de nombrar tablas y columnas en su propio idioma.

Para satisfacer ambos requerimientos, MySQL guarda los metadatos en un conjunto de caracteres Unicode, llamado UTF8. Esto no causa ningún problema si no se utilizan nunca caracteres acentuados o no-latinos. Si se utilizan, debe tenerse en cuenta que los metadatos están en UTF8.

Esto significa que las funciones `USER()`, `CURRENT_USER()`, `DATABASE()`, y `VERSION()` tienen el conjunto de caracteres UTF8 por defecto, al igual que sinónimos como `SESSION_USER()` y `SYSTEM_USER()`.

El servidor activa la variable de sistema `character_set_system` con el nombre del conjunto de caracteres de los metadatos:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

El almacenamiento de metadatos usando Unicode *no* significa que las cabeceras de columnas y los resultados de la función `DESCRIBE` estén en el conjunto de caracteres `character_set_system` por defecto. Cuando introduce `SELECT column1 FROM t`, el nombre `column1` mismo vuelve del servidor al cliente con el conjunto de caracteres determinado por el comando `SET NAMES` . Más específicamente, el conjunto de caracteres usado se determina por el valor de la variable de sistema `character_set_results` . Si esta variable se pone a `NULL`, no se realiza ninguna conversión y el servidor devuelve metadatos usando su conjunto de caracteres original (el conjunto indicado por `character_set_system`).

Para que el servidor pase los resultados de metadatos con un conjunto de caracteres no-UTF8, utilícese `SET NAMES` para forzar al servidor a hacer una conversión del conjunto de caracteres (consulte Sección 10.3.6, “Conjunto de caracteres y colación de la conexión”), o hacer que el cliente haga la conversión. Es más eficiente que el cliente haga la conversión, pero esta opción no está disponible para todos los clientes.

Si utiliza (por ejemplo) la función `USER()` para comparación o asignación dentro de un comando, no se preocupe. MySQL realiza algunas conversiones automáticas.

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

Esta sentencia funciona porque los contenidos de `latin1_column` se convierten automáticamente a UTF8 antes de la comparación.

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

Esta sentencia funciona porque los contenidos de `USER()` se convierten automáticamente a `latin1` antes de la asignación. La conversión automática no está completamente implementada aún, pero debería funcionar correctamente en versiones posteriores.

Aunque la conversión automática no está en el estándar SQL, el documento estándar SQL dice que cada conjunto de caracteres es (en términos de caracteres soportados) un “subconjunto” de Unicode. Puesto que es un principio bien conocido que “lo que se aplica a un superconjunto se puede aplicar a un subconjunto”, creemos que una colación para Unicode se puede aplicar para comparaciones con cadenas de caracteres no-Unicode.

10.7. Compatibilidad con otros SGBDs (Sistemas gestores de bases de datos)

Para compatibilidad con MaxDB estos dos comandos son iguales:

```
CREATE TABLE t1 (f1 CHAR(n) UNICODE);
CREATE TABLE t1 (f1 CHAR(n) CHARACTER SET ucs2);
```

10.8. Formato del nuevo fichero de conjunto de caracateres

En MySQL 5.0, la configuración del conjunto de caracteres se guarda en ficheros XML, un fichero por conjunto de caracteres. (Antes de MySQL 4.1, esta información se guardaba en ficheros `.conf`.

10.9. Conjunto de caracteres nacional

ANSI SQL define `NCHAR` o `NATIONAL CHAR` como formas de indicar que una columna `CHAR` debe usar algún conjunto de caracteres predefinido. MySQL 5.0 usa `utf8` como su conjunto de caracteres predefinido. Por ejemplo, estas declaraciones de tipos de columnas son equivalentes:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

Igual que éstas:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

Puede usar `N'literal'` para crear una cadena de caracteres en el conjunto de caracteres nacional. Estos dos comandos son equivalentes:

```
SELECT N'some text';
SELECT _utf8'some text';
```

10.10. Conjuntos de caracteres y colaciones que soporta MySQL

MySQL soporta más de 70 colaciones para más de 30 conjuntos de caracteres. Los conjuntos de caracteres y sus colaciones por defecto se muestran con el comando `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   | ISO 8859-1 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 |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
```

```
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
| macce    | Mac Central European        | macce_general_ci    |      1 |
| macroman | Mac West European           | macroman_general_ci |      1 |
| cp852    | DOS Central European        | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |      1 |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |      1 |
| cp1256   | Windows Arabic              | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic              | cp1257_general_ci   |      1 |
| binary   | Binary pseudo charset       | binary              |      1 |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |      1 |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |      2 |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |      3 |
+----------+-----------------------------+---------------------+--------+
```

10.10.1. Conjuntos de caracteres Unicode

MySQL tiene dos conjuntos de caracteres Unicode. Puede almacenar texto en unos 650 idiomas usando estos conjuntos de caracteres. Hemos añadido varias colaciones para estos dos nuevos conjuntos, y pronto añadiremos más.

- `ucs2` (UCS-2 Unicode) colaciones:

```
mysql> SHOW COLLATION LIKE 'ucs2%';
+--------------------+---------+-----+---------+----------+---------+
| Collation          | Charset | Id  | Default | Compiled | Sortlen |
+--------------------+---------+-----+---------+----------+---------+
| ucs2_general_ci    | ucs2    |  35 | Yes     | Yes      |       1 |
| ucs2_bin           | ucs2    |  90 |         | Yes      |       1 |
| ucs2_unicode_ci    | ucs2    | 128 |         | Yes      |       8 |
| ucs2_icelandic_ci  | ucs2    | 129 |         | Yes      |       8 |
| ucs2_latvian_ci    | ucs2    | 130 |         | Yes      |       8 |
| ucs2_romanian_ci   | ucs2    | 131 |         | Yes      |       8 |
| ucs2_slovenian_ci  | ucs2    | 132 |         | Yes      |       8 |
| ucs2_polish_ci     | ucs2    | 133 |         | Yes      |       8 |
| ucs2_estonian_ci   | ucs2    | 134 |         | Yes      |       8 |
| ucs2_spanish_ci    | ucs2    | 135 |         | Yes      |       8 |
| ucs2_swedish_ci    | ucs2    | 136 |         | Yes      |       8 |
| ucs2_turkish_ci    | ucs2    | 137 |         | Yes      |       8 |
| ucs2_czech_ci      | ucs2    | 138 |         | Yes      |       8 |
| ucs2_danish_ci     | ucs2    | 139 |         | Yes      |       8 |
| ucs2_lithuanian_ci | ucs2    | 140 |         | Yes      |       8 |
| ucs2_slovak_ci     | ucs2    | 141 |         | Yes      |       8 |
| ucs2_spanish2_ci   | ucs2    | 142 |         | Yes      |       8 |
| ucs2_roman_ci      | ucs2    | 143 |         | Yes      |       8 |
| ucs2_persian_ci    | ucs2    | 144 |         | Yes      |       8 |
+--------------------+---------+-----+---------+----------+---------+
```

- `utf8` (UTF-8 Unicode) colaciones:

```
mysql> SHOW COLLATION LIKE 'utf8%';
+--------------------+---------+-----+---------+----------+---------+
| Collation          | Charset | Id  | Default | Compiled | Sortlen |
+--------------------+---------+-----+---------+----------+---------+
| utf8_general_ci    | utf8    |  33 | Yes     | Yes      |       1 |
| utf8_bin           | utf8    |  83 |         | Yes      |       1 |
| utf8_unicode_ci    | utf8    | 192 |         | Yes      |       8 |
| utf8_icelandic_ci  | utf8    | 193 |         | Yes      |       8 |
| utf8_latvian_ci    | utf8    | 194 |         | Yes      |       8 |
| utf8_romanian_ci   | utf8    | 195 |         | Yes      |       8 |
| utf8_slovenian_ci  | utf8    | 196 |         | Yes      |       8 |
```

```
| utf8_polish_ci     | utf8    | 197 |         | Yes      |       8 |
| utf8_estonian_ci   | utf8    | 198 |         | Yes      |       8 |
| utf8_spanish_ci    | utf8    | 199 |         | Yes      |       8 |
| utf8_swedish_ci    | utf8    | 200 |         | Yes      |       8 |
| utf8_turkish_ci    | utf8    | 201 |         | Yes      |       8 |
| utf8_czech_ci      | utf8    | 202 |         | Yes      |       8 |
| utf8_danish_ci     | utf8    | 203 |         | Yes      |       8 |
| utf8_lithuanian_ci | utf8    | 204 |         | Yes      |       8 |
| utf8_slovak_ci     | utf8    | 205 |         | Yes      |       8 |
| utf8_spanish2_ci   | utf8    | 206 |         | Yes      |       8 |
| utf8_roman_ci      | utf8    | 207 |         | Yes      |       8 |
| utf8_persian_ci    | utf8    | 208 |         | Yes      |       8 |
+--------------------+---------+-----+---------+----------+---------+
```

La colación `utf8_unicode_ci` está implementada según el Algoritmo de Colación Unicode (UCA, Unicode Collation Algorithm) descrito en http://www.unicode.org/reports/tr10/. La colación usa la versión-4.0.0 UCA para el peso de las claves: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. (La siguiente exposición se basa en `utf8_unicode_ci`, pero también es cierta para `ucs2_unicode_ci`.)

Actualmente, la colación `utf8_unicode_ci` tiene soporta sólo parcialmente el Algoritmo de colación Unicode. Algunos caracteres no están todavía soportados. Además, la combinacion de marcas no está soportada completamente. Esto afecta al vietnamita y a algunos idiomas minoritarios en Rusia como el Udmurt, Tatar, Bashkir, y Mari.

La caractarística más significativa en `utf8_unicode_ci` es que soporta expansiones, esto es, cuando un carácter resulta igual a una combinación de otros caracteres. Por ejemplo, en alemán y otros idiomas 'ß' es igual a '`ss`'.

`utf8_general_ci` es una colación heredada que no soporta expansiones. Sólo puede hacer comparaciones entre caracteres uno a uno. Esto significa que las comparaciones para colaciones `utf8_general_ci` son más rápidas, pero ligeramente menos correctas, que las comparaciones en `utf8_unicode_ci`).

Por ejemplo, las siguientes ecuaciones funcionan tanto en `utf8_general_ci` como en `utf8_unicode_ci`:

```
Ä = A
Ö = O
Ü = U
```

En cambio, esto es cierto para `utf8_general_ci`:

```
ß = s
```

Minentras que para `utf8_unicode_ci` es cierta la siguiente:

```
ß = ss
```

Colaciones específicas de idiomas para el conjunto de caracteres `utf8` están implementadas sólo si la ordenación con `utf8_unicode_ci` no funciona bien para un idioma. Por ejemplo, `utf8_unicode_ci` funciona bien para alemán y francés, así que no es necesario crear colaciones `utf8` especiales para estos dos idiomas.

`utf8_general_ci` también es satisfactorio para alemán y francés, excepto que 'ß' es igual a '`s`', y no a '`ss`'. Si esto es aceptable para su aplicación, debe usar `utf8_general_ci` porque es más rápido. En caso contraro , use `utf8_unicode_ci` porque es más preciso.

`utf8_swedish_ci`, como otras colaciones `utf8` específicas de un idioma, deriva de `utf8_unicode_ci` con reglas adicionales de idioma. Por ejemplo, en sueco, la siguiente comparación es cierta, lo que no esperaría un alemán o un francés:

```
Ü = Y < Ö
```

Las colaciones `utf8_spanish_ci` y `utf8_spanish2_ci` se corresponden con español moderno y español tradicional respectivamente. En ambas colaciones , '`ñ`' es una letra independiente, entre '`n`' y '`o`'. Además, para español tradicional '`ch`' es una letra, ordenada entre '`c`' y `d`, y '`ll`' es una letra que se coloca entre '`l`' y '`m`'

10.10.2. Conjuntos de caracteres de Europa occidental

Los conjuntos de caracteres para Europa Occidental cubren la mayoría de lenguas de Europa Occidental como francés, español, catalán, vasco, protugués, italiano, albano, alemán, holandés, sueco, noruego, finlandés, feroés, islandés, irlandés, escocés e inglés.

- `ascii` (US ASCII) colaciones:
  - `ascii_bin`
  - `ascii_general_ci` (por defecto)
- `cp850` (DOS Europa Occidental) colaciones:
  - `cp850_bin`
  - `cp850_general_ci` (por defecto)
- `dec8` (DEC Europa Occidental) colaciones:
  - `dec8_bin`
  - `dec8_swedish_ci` (por defecto)
- `hp8` (HP Europa Occidental) colaciones:
  - `hp8_bin`
  - `hp8_english_ci` (por defecto)
- `latin1` (ISO 8859-1 Europa Occidental) colaciones:
  - `latin1_bin`
  - `latin1_danish_ci`
  - `latin1_general_ci`
  - `latin1_general_cs`
  - `latin1_german1_ci`
  - `latin1_german2_ci`
  - `latin1_spanish_ci`
  - `latin1_swedish_ci` (por defecto)

`latin1` es el conjunto de caracteres por defecto. La colación `latin1_swedish_ci` es la colación por defecto que probablemente usan la mayoría de clientes MySQL. Se dice frecuentemente que está basado en las reglas de colación sueca/finlandesa, pero hay suecos y finlandeses que no están de acuerdo con esta afirmación.

Las colaciones `latin1_german1_ci` y `latin1_german2_ci` están basadas en los estándars DIN-1 y DIN-2, mientras DIN significa *Deutsches Institut für Normung* (el equivalente alemán de ANSI). DIN-1 es el diccionario de la colación y DIN-2 son las "páginas amarillas" de la colación.

- `latin1_german1_ci` (diccionario) reglas:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- `latin1_german2_ci` (páginas amarillas) reglas:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

En la colación `latin1_spanish_ci`, '`ñ`' es una letra independiente, entre '`n`' y '`o`'.

- `macroman` (Mac Europa Occidental) colaciones:
  - `macroman_bin`
  - `macroman_general_ci` (por defecto)
- `swe7` (7bit Sueca) colación:
  - `swe7_bin`
  - `swe7_swedish_ci` (por defecto)

10.10.3. Conjuntos de caracteres de Europa central

También proveemos algún soporte para conjuntos de caracteres usados en la República Checa, Eslovaquia, Hungría, Romania, Eslovenia, Croacia y Polonia.

- `cp1250` (Windows Europa Central) colaciones:
  - `cp1250_bin`
  - `cp1250_croatian_ci`
  - `cp1250_czech_cs`
  - `cp1250_general_ci` (por defecto)
- `cp852` (DOS Europa Central) colaciones:
  - `cp852_bin`
  - `cp852_general_ci` (por defecto)

- `keybcs2` (DOS Kamenicky Checo-Eslovaco) colaciones:
  - `keybcs2_bin`
  - `keybcs2_general_ci` (por defecto)
- `latin2` (ISO 8859-2 Europa Central) colaciones:
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_cs`
  - `latin2_general_ci` (por defecto)
  - `latin2_hungarian_ci`
- `macce` (Mac Europa Central) colaciones:
  - `macce_bin`
  - `macce_general_ci` (por defecto)

10.10.4. Conjuntos de caracteres del sur de Europa y de Oriente Medio

- `armscii8` (ARMSCII-8 Armenia) colaciones:
  - `armscii8_bin`
  - `armscii8_general_ci` (por defecto)
- `cp1256` (Windows Arabiga) colaciones:
  - `cp1256_bin`
  - `cp1256_general_ci` (por defecto)
- `geostd8` (GEOSTD8 Georgia) colaciones:
  - `geostd8_bin`
  - `geostd8_general_ci` (por defecto)
- `greek` (ISO 8859-7 Grecia) colaciones:
  - `greek_bin`
  - `greek_general_ci` (default)
- `hebrew` (ISO 8859-8 Hebrew) collations:
  - `hebrew_bin`
  - `hebrew_general_ci` (por defecto)
- `latin5` (ISO 8859-9 Turquía) colaciones:
  - `latin5_bin`

- latin5_turkish_ci (por defecto)

10.10.5. Conjuntos de caracteres bálticos

Los conjuntos de caracteres bálticos cubren estonio, letón, y lituano. Hay dos conjuntos de caracteres bálticos soportados en la actualidad:

- cp1257 (Windows Báltico) colaciones:
  - cp1257_bin
  - cp1257_general_ci (por defecto)
  - cp1257_lithuanian_ci
- latin7 (ISO 8859-13 Báltico) colaciones:
  - latin7_bin
  - latin7_estonian_cs
  - latin7_general_ci (por defecto)
  - latin7_general_cs

10.10.6. Conjuntos de caracteres cirílicos

Estos son los conjuntos de caracteres cirílicos y colaciones para usar con bielorruso, búlgaro, ruso y ucraniano.

- cp1251 (Windows cirílico) colaciones:
  - cp1251_bin
  - cp1251_bulgarian_ci
  - cp1251_general_ci (por defecto)
  - cp1251_general_cs
  - cp1251_ukrainian_ci
- cp866 (DOS Ruso) colaciones:
  - cp866_bin
  - cp866_general_ci (por defecto)
- koi8r (KOI8-R Relcom Rusia) colaciones:
  - koi8r_bin
  - koi8r_general_ci (por defecto)
- koi8u (KOI8-U Ukrania) colaciones:
  - koi8u_bin

- `koi8u_general_ci` (por defecto)

10.10.7. Conjuntos de caracteres asiáticos

Los conjuntos de caracteres asiáticos que soportamos incluyen chino, japonés, koreano y thai. Esto puede ser complicado. Por ejemplo, los conjuntos chinos deben permitir miles de caracteres distintos.

- `big5` (Big5 chino tradicional) colaciones:
  - `big5_bin`
  - `big5_chinese_ci` (por defecto)
- `cp932` (SJIS para Windows japonés) collations:
  - `cp932_bin`
  - `cp932_japanese_ci` (por defecto)
- `eucjpms` (UJIS para Windows japonés) colaciones:
  - `eucjpms_bin`
  - `eucjpms_japanese_ci` (por defecto)
- `euckr` (EUC-KR Koreano) colaciones:
  - `euckr_bin`
  - `euckr_korean_ci` (por defecto)
- `gb2312` (GB2312 chino simplificado) colaciones:
  - `gb2312_bin`
  - `gb2312_chinese_ci` (por defecto)
- `gbk` (GBK chino simplificado) colaciones:
  - `gbk_bin`
  - `gbk_chinese_ci` (por defecto)
- `sjis` (Shift-JIS japonés) colaciones:
  - `sjis_bin`
  - `sjis_japanese_ci` (por defecto)
- `tis620` (TIS620 Thai) colaciones:
  - `tis620_bin`
  - `tis620_thai_ci` (por defecto)
- `ujis` (EUC-JP japonés) colaciones:
  - `ujis_bin`

- `ujis_japanese_ci` (por defecto)

Capítulo 11. Tipos de columna

Tabla de contenidos

MySQL soporta un número de tipos de columnas divididos en varias categorías: tipos númericos, tipos de fecha y hora, y tipos de cadenas de caracteres. Este capítulo primero hace un repaso de estos tipos de columnas, y luego proporciona una descripción detallada de las propiedades de los tipos de cada categoría, y un resumen de sus requerimientos de almacenamiento. El repaso es intencionalmente breve. Las descripciones más detalladas deben consultarse para obtener más información acerca de los tipos de datos particulares, como los formatos permitidos para especificar los tipos.

MySQL 5.0 soporta extensiones para tratar datos espaciales. Información al respecto se proporciona en Capítulo 18, *Extensiones espaciales de MySQL*.

Varias descripciones de los tipos de columnas usan estas convenciones:

- *`M`*

  Indica la máxima anchura al mostrar los datos. El máximo ancho de muestra es 255.

- *`D`*

  Se aplica a tipos de coma flotante y de coma fija e indica el número de dígitos a continuación del punto decimal. El valor máximo posible es 30, pero no debe ser mayor que *`M`*-2.

-

  Los corchetes ('`[`' y '`]`') indican partes de especificadores de tipos que son opcionales.

11.1. Panorámica de tipos de columna

11.1.1. Panorámica de tipos numéricos

A continuación hay un resumen de los tipos de columnas numéricos. Para información adicional, consulte Sección 11.2, “Tipos numéricos”. Los requerimientos de almacenamiento de columna se dan en Sección 11.5, “Requisitos de almacenamiento según el tipo de columna”.

M indica la anchura máxima para mostrar. La anchura máxima es 255. La anchura de muestra no tiene nada que ver con el tamaño de almacenamiento o el rango de valores que el valor puede contener, como se describe en Sección 11.2, “Tipos numéricos”.

Si especifica `ZEROFILL` para columnas numéricas,, MySQL añade automáticamente el atributo `UNSIGNED` en la columna.

`SERIAL` es un alias para `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT`.

`SERIAL DEFAULT VALUE` en la definición de una columna de tipo entero es un alias para `NOT NULL AUTO_INCREMENT UNIQUE`.

Atención: Debe tener en cuenta que cuando usa la resta entre valores enteros cuando uno de los operandos es de tipo `UNSIGNED`, el resultado no tiene signo. Consulte Sección 12.8, “Funciones y operadores de cast”.

- `BIT[(M)]`

  En un tipo de datos bit. *M* indica el número de bits por valor, de 1 a 64. El valor por defecto es 1 si se omite *M* .

  Este tipo de datos se añadió en MySQL 5.0.3 para `MyISAM`, una extensión en 5.0.5 para `MEMORY`, `InnoDB`, y `BDB`. Antes de 5.0.3, `BIT` es un sinónimo de `TINYINT(1)`.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  Un entero muy pequeño. El rango con signo es de `-128` a `127`. El rango sin signo es de `0` a `255`.

- `BOOL`, `BOOLEAN`

  Son sinónimos para `TINYINT(1)`. Un valor de cero se considera falso. Valores distintos a cero se consideran ciertos.

  En el futuro, se introducirá tratamiento completo de tipos booleanos según el estándar SQL.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

  Un entero pequeño. El rango con signo es de `-32768` a `32767`. El rango sin signo es de `0` a `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

  Entero de tamaño medio. El rango con signo es de `-8388608` a `8388607`. El rango sin singo es de `0` a `16777215`.

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

  Un entero de tamaño normal. El rango con signo es de `-2147483648` a `2147483647`. El rango sin signo es de `0` a `4294967295`.

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

  Es un sinónimo de `INT`.

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

  Un entero grande. El rango con signo es de `-9223372036854775808` a `9223372036854775807`. El rango sin signo es de `0` a `18446744073709551615`.

  Algunos aspectos a considerar con respecto a las columnas `BIGINT` :

  - Toda la aritmética se hace usando valores `BIGINT` o `DOUBLE`, así que no debe usar enteros sin signos mayores que `9223372036854775807` (63 bits) except con funciones bit! Si lo hace, algunos de los últimos dígitos en el resultado pueden ser erróneos por culpa de errores de redondeo al convertir valores `BIGINT` a `DOUBLE`.

    MySQL 5.0 puede tratar `BIGINT` en los siguientes casos:

    - Cuando usa enteros para almacenar valores grandes sin signo en una columna `BIGINT` .
    - En `MIN(col_name)` o `MAX(col_name)`, donde `col_name` se refiere a una columna `BIGINT` .
    - Al usar operadores (`+`, `-`, `*`, y así) donde ambos operadores son enteros.
  - Siempre puede guardar un valor entero exacto en un columna `BIGINT` almacenándolo usando una cadena de caracteres. En este caso, MySQL realiza una conversión cadena de caracteres-número que no implica representación de doble precisión intermedia.
  - Los operadores `-`, `+`, y `*` usan `BIGINT` en operaciones aritméticas cuando ambos operandos son valores enteros. Esto significa que si multiplica dos enteros grandes (o resultados de funciones que devuelven enteros), puede obtener resultados inesperados cuando el resultado es mayor que `9223372036854775807`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

Número con coma flotante. *p* representa la precisión. Puede ir de 0 a 24 para números de coma flotante de precisión sencilla y de 25 a 53 para números de coma flotante con doble precisión. Estos tipos son como los tipos `FLOAT` y `DOUBLE` descritos a continuación. `FLOAT(p)` tiene le mismo rango que los tipos correspondientes `FLOAT` y `DOUBLE`, pero la anchura de muestra y el número de decimales no están definidos.

En MySQL 5.0, este es un valor de coma flotante auténtico.

Esta sintaxis se proprociona para compatibilidad con ODBC.

Usar `FLOAT` puede darle algunos problemas inesperados ya que todos los cálculos se en MySQL se hacen con doble precisión. Consulte Sección A.5.7, “Resolver problemas con registros que no salen”.

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

  Un número de coma flotante pequeño (de precisión simple). Los valores permitidos son de `-3.402823466E+38` a `-1.175494351E-38`, `0`, y de `1.175494351E-38` a `3.402823466E+38`. Si se especifica `UNSIGNED`, los valores negativos no se permiten. *M* es la anchura de muestra y *D* es el número de dígitos significativos. `FLOAT` sin argumentos o `FLOAT(p)` (donde *p* está en el rango de 0 a 24) es un número de coma flotante con precisión simple.

- `DOUBLE[(M,B)] [UNSIGNED] [ZEROFILL]`

  Número de coma flotante de tamaño normal (precisión doble). Los valores permitidos son de `-1.7976931348623157E+308` a `-2.2250738585072014E-308`, `0`, y de `2.2250738585072014E-308` a `1.7976931348623157E+308`. Si se especifica `UNSIGNED`, no se permiten valores negativos. *M* es la anchura de muestra y *B* es el número de bits de precisión. `DOUBLE` sin parámetros o `FLOAT(p)` (donde *p* está en el rango de 25 a 53) es un número de coma flotante con doble precisión. Un número de coma flotante con precision sencilla tiene una precisión de 7 decimales aproximadamente; un número con coma flotante de doble precisión tiene una precisión aproximada de 15 decimales.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

  Son sinónimos de `DOUBLE`. Excepción: Si el modo del servidor SQL incluye la opción `REAL_AS_FLOAT`, `REAL` es un sinónimo para `FLOAT` en lugar de `DOUBLE`.

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

  A partir de MySQL 5.0.3:

  Número de punto fijo exacto y empaquetado. *M* es el número total de dígitos y *D* es el número de decimales. El punto decimal y (para números negativos) el signo '`-`' no se tiene en cuenta en *M*. Si *D* es 0, los valores no tienen punto decimal o parte fraccional. El máximo número de dígitos (*M*) para

DECIMAL es 64. El máximo número de decimales soportados (*D*) es 30. Si UNSIGNED se especifica, no se permiten valores negativos.

Si se omite *D*, el valor por defecto es 0. Si se omite *M*, el valor por defecto es 10.

Todos los cálculos básicos (+, -, *, /) con columnas DECIMAL se hacen con precisión de 64 dígitos decimales.

Antes de MySQL 5.0.3:

Número de punto decimal fijo sin empaquetar. Se comporta como una columna CHAR ; "sin empaquetar" significa que el número se alacena como una cadena de caracteres, usando un carácter para cada dígito del valor. *M* es el número total de dígitos y *D* es el número de decimales. El punto decimal y (para números negativos) el signo '-' no se cuenta en *M*, aunque se reserva espacio para él. Si *D* es 0, los valores no tienen punto decimal ni parte fraccional. El máximo rango de los valores DECIMAL es el mismo que para DOUBLE, pero el rango real para una columna DECIMAL dada puede estar restringido por la elección de *M* y *D*. Si se especifica UNSIGNED no se permiten números negativos.

Si se omite *D*, el valor por defecto es 0. Si se omite *M*, el valor por defecto es 10.

- DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  Son sinónimos para DECIMAL. El sinónimo FIXED está disponible por compatibilidad con otros servidores.

11.1.2. Panorámica de tipos de fechas y hora

Un resumen de los tipos de columnas temporales se muestra a continuación. Para información adicional, consulte Sección 11.3, “Tipos de fecha y hora”. Los requerimientos de almacenamiento se dan en Sección 11.5, “Requisitos de almacenamiento según el tipo de columna”.

- DATE

  Una fecha. El rango soportado es de '1000-01-01' a '9999-12-31'. MySQL muestra valores DATE en formato 'YYYY-MM-DD', pero permite asignar valores a columnas DATE usando cadenas de caracteres o números.

- DATETIME

  Combinación de fecha y hora. El rango soportado es de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'. MySQL muestra valores DATETIME en formato 'YYYY-MM-DD HH:MM:SS', pero permite asignar valores a las columnas DATETIME usando cadenas de caracteres o números.

- TIMESTAMP[(*M*)]

  Una marca temporal. El rango es de '1970-01-01 00:00:00' hasta el año 2037.

Una columna `TIMESTAMP` es útil para registrar la fecha y hora de una operación `INSERT` o `UPDATE` . La primera columna `TIMESTAMP` en una tabla se rellena automáticamente con la fecha y hora de la operación más reciente si no le asigna un valor. Puede asignar a cualquier columna `TIMESTAMP` la fecha y hora actual asignándole un valor `NULL` .

En MySQL 5.0, `TIMESTAMP` se retorna como una cadena de caracteres en el formato `'YYYY-MM-DD HH:MM:SS'` cuya anchura de muestra son 19 caracteres. Si quiere obtener el valor como un número, debe añadir `+0` a la columa timestamp .

- `TIME`

  Una hora. El rango es de `'-838:59:59'` a `'838:59:59'`. MySQL muestra los valores `TIME` en formato `'HH:MM:SS'`, pero permite asingar valores a columnas `TIME` usando números o cadenas de caracteres.

- `YEAR[(2|4)]`

  Un año en formato de dos o cuatro dígitos. El valor por defecto está en formato de cuatro dígitos. En formato de cuatro dígitos, los valores permitidos son de `1901` a `2155`, y `0000`. En formato de dos dígitos, los valores permitidos son de `70` a `69`, representando los años de 1970 a 2069. MySQL muestra los valores `YEAR` en formato `YYYY` pero permite asignar valores a columnas `YEAR` usando cadenas de caracteres o números.

11.1.3. Panorámica de tipos de cadenas de caracteres

Un resumen de los tipos de columnas de cadenas de caracteres se muestra a continuación. Para información adicional, consulte Sección 11.4, “Tipos de cadenas de caracteres”. Los requerimientos de almacenamiento de estas columnas se dan en Sección 11.5, “Requisitos de almacenamiento según el tipo de columna”.

En algunos casos, MySQL puede cambiar una columna de cadena de caracteres a un tipo diferente para un comando `CREATE TABLE` o `ALTER TABLE` . Consulte Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”.

Los tipos de cadenas de caracteres MySQL 5.0 incluyen algunas características que puede que no haya encontrado trabajando con versiones anteriores de MySQL anteriores a la 4.1:

- Las definiciones de columnas para varios tipos de datos de cadenas de caracteres incluyen un atributo `CHARACTER SET` para especificar el conjunto de caracteres y, ocasionalmente, una colación. (`CHARSET` es sinónimo de `CHARACTER SET`.) Estos atributos se aplican a los tipos `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, y `SET`. Por ejemplo:

```
CREATE TABLE t
(
    c1 CHAR(20) CHARACTER SET utf8,
    c2 CHAR(20) CHARACTER SET latin1 COLLATE latin1_bin
);
```

  Esta definición de tabla crea una columna llamada `c1` que tiene un conjunto de caracteres `utf8` con la colación por defecto para ese conjunto de caracteres, y una columna llamada `c2` que tiene el conjunto

de caracteres `latin1` y la colación binaria para el conjunto de caracteres. La colación binaria no es sensible a mayúsculas.

- MySQL 5.0 interpreta las especificaciones de longitud en las definiciones de las columnas en unidades de caracteres . (En algunas versiones anteriores de MySQL la longitud se interpreta en bytes.)
- Para los tipos `CHAR`, `VARCHAR`, y the `TEXT`, el atributo `BINARY` hace que se asigne a la columna la colación binaria del conjunto de caracteres.
- Las ordenaciones y comparaciones de las columnas de tipo carácter se basan en el conjunto de caracteres asignado a la columna. Para versiones anteriores, la comparación y ordenación se basan en la colación del conjunto de caracteres del servidor. Para columnas `CHAR` y `VARCHAR`, puede declarar que la columna con el atributo `BINARY` realice la ordenación y la comparación usando los códigos de los valores subyacentes en lugar del orden léxico.

Para más información acerca del soporte de conjuntos de caracteres en MySQL 5.0, consulte Capítulo 10, *Soporte de conjuntos de caracteres*.

- `[NATIONAL] CHAR(M) [BINARY | ASCII | UNICODE]`

  Una cadena de caracteres de longitud fija que siempre tiene el número necesario de espacios a la derecha para ajustarla a la longitud especificada al almacenarla. *`M`* representa la longitud de la columna. El rango de *`M`* en MySQL 5.0 es de 0 a 255 caracteres.

  Nota: Los espacios a la derecha se borran cuando se obtiene los valores `CHAR` .

  Antes de MySQL 5.0.3, una columna `CHAR` con una longitud especificada mayor que 255 se convierte al tipo `TEXT` más pequeño que pueda tener los valores de la longitud dada. Por ejemplo, `CHAR(500)` se convierte a `TEXT`, y `CHAR(200000)` se convierte en `MEDIUMTEXT`. Esta es una característica de compatibilidad. Sin embargo, esta conversión causa que la columna tenga longitud variable, y también afecta a la eliminación de espacios.

  `CHAR` es una abreviatura para `CHARACTER`. `NATIONAL CHAR` (o su forma equivalente de, `NCHAR`) es la forma estándar de SQL de definir que una columna `CHAR` debe usar el conjunto de caracteres por defecto. Este es el comportamiento por defecto en MySQL.

  El atributo `BINARY` es una abreviatura para especificar la colación binaria del conjunto de caracteres de la columna. La ordenación y comparación se basa en los valores numéricos de los caracteres.

  El tipo de columna `CHAR BYTE` es un alias para `CHAR BINARY`. Esta es una característica de compatibilidad.

  El atributo `ASCII` puede especificarse para `CHAR`. Asigna el conjunto de caracteres `latin1`.

  El atributo `UNICODE` puede especificarse en MySQL 5.0 para `CHAR`. Asigna el conjunto de caracteres `ucs2` .

  MySQL le permite crear un tipo de columna `CHAR(0)`. Esto es útil cuando tiene que cumplir con las especificaciones de alguna aplicación vieja que dependa de la existencia de una columna pero que no usa realmente el valor. Esto es también útil cuando necesita una columna que sólo pueda tener dos valores: Una columna `CHAR(0)` que no esté definido como `NOT NULL` ocupa sólo un bit y sólo puede tener dos valores `NULL` y `''` (la cadena de caracteres vacía).

- `CHAR`

  Es un sinónimo de `CHAR(1)`.

- `[NATIONAL] VARCHAR(M) [BINARY]`

  Cadena de caracteres de longitud variable. *M* representa la longitud de columna máxima. En MySQL 5.0, el rango de *M* es de 0 a 255 antes de MySQL 5.0.3, y de 0 a 65,535 en MySQL 5.0.3 y posterior. (La longitud máxima real de un `VARCHAR` en MySQL 5.0 se determina por el tamaño de registro máximo y el conjunto de caracteres que use. La longitud máxima *efectiva* desde MySQL 5.0.3 es de 65,532 bytes.)

  Nota: Antes de 5.0.3, los espacios finales se eliminaban cuando se almacenaban los valores `VARCHAR`, lo que difiere de le especificación estándar de SQL.

  Previo a MySQL 5.0.3, una columna `VARCHAR` con una longitud especificada mayor a 255 se convertía al valor de tipo `TEXT` más pequeño que podía soportar el valor de la longitu dada. Por ejemplo, `VARCHAR(500)` se convertía a `TEXT`, y `VARCHAR(200000)` se convertía a `MEDIUMTEXT`. Esto era una cuestión de compatibilidad. Sin embargo, esta conversión afectaba la eliminación de espacios finales.

  `VARCHAR` es la abreviación de `CHARACTER VARYING`.

  En MySQL 5.0, el atributo `BINARY` es abreviatura para especificar la colación binaria del conjunto de caracteres de la columna. La ordenación y la comparación se basa en los valores numéricos de los caracteres.

  Desde MySQL 5.0.3, `VARCHAR` se guarda con un prefijo de longitud de uno o dos bytes + datos. La longitud del prefijo es de dos bytes si la columna `VARCHAR` se declara con una longitud mayor a 255.

- `BINARY(M)`

  El tipo `BINARY` es similar al tipo `CHAR`, pero almacena cadenas de datos binarios en lugar de cadenas de caracteres no binarias.

- `VARBINARY(M)`

  El tipo `VARBINARY` es similar al tipo `VARCHAR`, pero almacena cadenas de caracteres binarias en lugar de cadenas de caracteres no binarias.

- `TINYBLOB`

  Una columna `BLOB` con una longitud máxima de 255 (2^8 - 1) bytes.

- `TINYTEXT`

  Una columna `TEXT` con longitud máxima de 255 (2^8 - 1) caracteres.

- `BLOB[(M)]`

  Una columna `BLOB` con longitud máxima de 65,535 (2^16 - 1) bytes.

  Una longitud opcional *`M`* puede darse para este tipo en MySQL 5.0. Si se hace, MySQL creará las columnas como el tipo `BLOB` de tamaño mínimo para tratar los valores de *`M`* bytes.

- `TEXT[(M)]`

  Una columna `TEXT` con longitud máxima de 65,535 (2^16 - 1) caracteres.

  En MySQL 5.0, se puede dar una longitud opcional *`M`* . En ese caso MySQL creará las columnas con el tipo `TEXT` de longitud mínima para almacenar los valors de longitud *`M`* .

- `MEDIUMBLOB`

  Una columna `BLOB` con longitud de 16,777,215 (2^24 - 1) bytes.

- `MEDIUMTEXT`

  Una columna `TEXT` con longitud máxima de 16,777,215 (2^24 - 1) caracteres.

- `LONGBLOB`

  Una columna `BLOB` con longitud máxima de 4,294,967,295 o 4GB (2^32 - 1) bytes. La longitud máxima *efectiva* (permitida) de las columnas `LONGBLOB` depende del tamaño máximo configurado para los paquetes en el protocolo cliente/servidor y la memoria disponible.

- `LONGTEXT`

  Una columna `TEXT` con longitud máxima de 4,294,967,295 or 4GB (2^32 - 1) caracteres. La longitud máxima *efectiva* (permitida) de columnas `LONGTEXT` depende del tamaño máximo de paquete configurado en el protocolo cliente/servidor y la memoria disponible.

- `ENUM('value1','value2',...)`

  Una enumeración. Un objeto de cadena de caracteres que sólo puede tener un valor, elegido de una lista de valores `'value1'`, `'value2'`, `...`, `NULL` o el valor de error especial `''` . Una columna `ENUM`

puede tener un máximo de 65,535 valores distintos. Los valores `ENUM` se representan internamente como enteros.

- `SET('value1','value2',...)`

  Un conjunto. Un objeto de cadena de caracteres que puede tener cero o más valores que deben pertencer a la lista de valores `'value1'`, `'value2'`, `...` Una columna `SET` puede tener un máximo de 64 miembros. Los valores `SET` se representan internamente como enteros.

11.2. Tipos numéricos

MySQL soporta todos los tipos de datos SQL numéricos estándar. Estos tipos incluyen los tipos numéricos exactos (`INTEGER`, `SMALLINT`, `DECIMAL`, y `NUMERIC`), así como los tipos de datos aproximados (`FLOAT`, `REAL`, y `DOUBLE PRECISION`). La palabra clave `INT` es sinónimo de `INTEGER`, y la palabra clave `DEC` es sinónimo de `DECIMAL`.

En MySQL 5.0.3, un tipo de datos `BIT` está disponible para almacenar valores de un bit. (Antes de 5.0.3, MySQL interpreta `BIT` como `TINYINT(1)`.) En MySQL 5.0.3, `BIT` lo soporta sólo tablas `MyISAM`. MySQL 5.0.5 extiende soporte de `BIT` para `MEMORY`, `InnoDB`, y `BDB`.

Como extensión de los estándares SQL, MySQL soporta los tipos enteros `TINYINT`, `MEDIUMINT`, y `BIGINT`. La siguiente tablas muestra el almacenamiento requerido y el rango para cada uno de los tipos enteros.

Tipo	Bytes	Valor Mínimo	Valor Máximo
		(Con signo/Sin signo)	**(Con signo/Sin signo)**
`TINYINT`	1	`-128`	`127`
		`0`	`255`
`SMALLINT`	2	`-32768`	`32767`
		`0`	`65535`
`MEDIUMINT`	3	`-8388608`	`8388607`
		`0`	`16777215`
`INT`	4	`-2147483648`	`2147483647`
		`0`	`4294967295`
`BIGINT`	8	`-9223372036854775808`	`9223372036854775807`
		`0`	`18446744073709551615`

MySQL soporta otra extensión para especificar de forma óptima el ancho a mostrar de un tipo entero en paréntesis después de la palabra clave para el tipo (por ejemplo, `INT(4)`). Esta especificación opcional del ancho de muestra se usa para alinear a la izquierda la muestra de los valores con ancho menor que el ancho especificado para la columna.

El ancho de muestra no restringe el rango de valores que pueden almacenarse en la columna, no el número de dígitos que se muestran para valores con ancho que exceda el especificado para la columna.

Cuando se usa en conjunción con el atributo de extensión opcional `ZEROFILL`, el relleno por defecto de espacios se replaza por ceros. Por ejmplo, para una columna declarada como `INT(5) ZEROFILL`, un valor de `4` se muestra como `00004`. Tenga en cuenta que si almacena valores mayores que el ancho de

muestra en una columna entera, puede tener problemas cuando MySQL genera tablas temporales para algunos joins complicados, ya que en estos casos MySQL cree que los datos encajan en el ancho original de la columna.

Todos los tipos enteros pueden tener un atributo opcional (no estándar) `UNSIGNED`. Los valores sin signo pueden usarse cuando quiere permitir sólo números no negativos en una columna y necesita un rango numérico mayor para la columna.

Tipos de coma flotante y de coma fija pueden ser `UNSIGNED`. Como con los tipos enteros, este atributo evita que los valores negativos se almacenen en la columna. Sin embargo, a diferencia de los tipos enteros, el rango superior de los valores de la columna sigue siendo el mismo.

Si especifica `ZEROFILL` para una columna numérica, MySQL añade automáticamente el atributo `UNSIGNED` a la columna.

Para columnas de tipo coma flotante, MySQL usa cuatro bytes para valores de precisión simple y ocho bytes para valores de doble precisión.

El tipo `FLOAT` se usa para representar tipos numéricos aproximados. El estándar SQL permite una especificación opcional de la precisión (pero no del rango del exponente) en bits a continación de la palabra clave `FLOAT` entre paréntesis. La implementación de MySQL soporta esta especificación opcional de precisión, pero el valor de precisión se usa sólo para determinar el tamaño de almacenamiento. Una precisión de 0 a 23 resulta en una columna de precisión simple de cuatro bytes de tamaño `FLOAT` . Una precisión de 24 a 53 resulta en una columna de doble precisión de ocho bytes de tamaño `DOUBLE` .

Cuando se especifica la palaba clave `FLOAT` para tipos de columnas sin especificar la precisión, MySQSL usa cuatro bytes para almacenar los valors.MySQL también soporta una sintaxis alternativa con dos números entre paréntesis a continación de la palabra clave `FLOAT` . El primer número representa el ancho a mostrar y el segundo número especifica el número de dígitos a almacenar a continuación del punto decimal (como con `DECIMAL` y `NUMERIC`). Cuando se pide a MySQL que almacene un número para tales columnas con más dígitos decimales a continuación del punto decimal del especificado para la columna, el valor se redondea para elminar los dígitos extras cuando se almacena el valor.

En SQL estándar, los tipos `REAL` y `DOUBLE PRECISION` no aceptan especificaciones de precisión. MySQL soporta una sintaxis alternativa con dos números dados entre paréntesis a continuación del nombre del tipo. El primer número representa el ancho a mostrar y el segundo número especifica el número de dígitos a almacenar y mostrar a continuación del punto decimal. Como una extensión al estándar SQL, MySQL reconoce `DOUBLE` como sinónimo del tipo `DOUBLE PRECISION` . En contraste con el requerimiento estándar que la precisión para `REAL` sea menor que la usada para `DOUBLE PRECISION`, MySQL implementa ambas como valores de punto flotante de doble precisión con tamaño de ocho bytes (a no ser que el modo SQL del servidor incluya la opción `REAL_AS_FLOAT`).

Para portabilidad máxima, el código que requiera almacenamiento de datos numéricos aproximados debe usar `FLOAT` o `DOUBLE PRECISION` sin especificar la precisión ni el número de dígitos decimales.

Los tipos `DECIMAL` y `NUMERIC` se implementan como el mismo tipo en MySQL. Se usan para guardar valores para los que es importante preservar una precisión exacta, por ejemplo con datos monetarios. Cuando se declara una columna de alguno de estos tipos, la precisión y la escala puede especificarse (y usualmente se hace), por ejemplo:

```
salary DECIMAL(5,2)
```

En este ejemplo, `5` es la precisión y `2` es la escala. La precisión representa el número de dígitos decimales significativos que se almacenan para los valores, y la escala representa el número de dígitos que pueden almacenarse a continuación del punto decimal.

Desde MySQL 5.0.3, los valores `DECIMAL` y `NUMERIC` se almacenan en formato binario. Antes de 5.0.3, MySQL almacena los valores `DECIMAL` y `NUMERIC` como cadenas de caracteres, en lugar de binario. .Un carácter se usa para cada dígito del valor, el punto decimal (si la escala es mayor que 0), y el signo '`-`' (para números negativos). Si la escala es 0, los valores `DECIMAL` y `NUMERIC` no contienen punto decimal o parte fraccional.

SQL estándar requiere que la columna `salary` sea capaz de almacenar cualquier valor con cinco dígitos y dos decimales. En este caso, por lo tanto, el rango de valores que puede almacenarse en la columna `salary` es desde `-999.99` a `999.99`. MySQL fuerza este límite desde MySQL 5.0.3. Antes de 5.0.3, MySQL 5.0 variaba este límite de forma que, en el límite positivo del rango, la columna podía almacenar números hasta `9999.99`. (Para números positivos, MySQL 5.0.2 y anteriores usaba el byte reservado para el signo para extender el límite superior del rango.)

En SQL estándar, la sintaxis `DECIMAL(M)` es equivalente a `DECIMAL(M,0)`. Similarmente, la sintaxis `DECIMAL` es equivalente a `DECIMAL(M,0)`, donde la implementación se permite para decidir el valor de *`M`*. Ambas formas de los tipos `DECIMAL` y `NUMERIC` se soportan en MySQL 5.0. El valor por defecto de *`M`* es 10.

El máximo rango de los valores `DECIMAL` y `NUMERIC` es el mismo para `DOUBLE`, pero el rango real para un valor dado en una columna `DECIMAL` o `NUMERIC` puede restringirse con la precisión o escala para una columna dada. Cuando en tal columna se asigna un valor con más dígitos siguiendo el punto decimal de los permitidos por la escala específica, el valor se convierte a tal escala. (El comportamiento preciso depende del sistema operativo, pero generalmente el efecto es que se trunca al número de dígitos permitidos.)

Desde MySQL 5.0.3, el tipo de datos `BIT` puede usarse para guardar valores de un bit. Un tipo `BIT(M)` permite el almacenamiento de valores de *`M`*-bit . *`M`* tiene un rango de 1 a 64.

Para especificar valores bit, puede usar la notación `b'value'` . *`value`* es un valor binario escrito usando ceros y unos. Por ejemplo, `b'111'` y `b'100000000'` representan 7 y 128, respectivamente. Consulte Sección 9.1.5, “Valores de bits”.

Si asigna un valor a una columna `BIT(M)` con menos de *`M`* bits , el valor se alinea a la izquierda con ceros. Por ejemplo, asignar un valor `b'101'` a una columna `BIT(6)` es, en efecto, lo mismo que asignar `b'000101'`.

Cuando se intenta almacenar un valor en una columna numérica que está fuera del rango permitido por la columna, MySQL corta el valor en el final del rango permitido y guarda el valor resultante.

Por ejemplo, el ranto de una coluna `INT` es de `-2147483648` a `2147483647`. Si intenta insertar `-9999999999` en una columna `INT`, MySQL reemplaza el valor con el mínimo valor del rango y almacena `-2147483648` en su lugar. De forma similar, si trata de insertar `9999999999`, MySQL reemplaza el valor con el valor máximo del rango y almacena `2147483647` en su lugar.

Si la columna `INT` es `UNSIGNED`, el tamaño del rango de la columna es el mismo, pero los límites cambian a `0` y `4294967295`. Si intenta almacenar `-9999999999` y `9999999999`, los valores almacenados en la columna son `0` y `4294967296`.

Cuando se asigna un valor fuera de rango especificado (o por defecto) a una columna de coma flotante o fija, MySQL almacena el valor representado por el valor correspondiente al límite de rango correspondiente.

Las conversiones debidas a valores fuera de rango se reportan como advertencias para los comandos `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, y `INSERT` de múltiples registros.

11.3. Tipos de fecha y hora

Los tipos de fecha y hora para representar valores temporales son `DATETIME`, `DATE`, `TIMESTAMP`, `TIME`, y `YEAR`. Cada tipo temporal tiene un rango de valores legales, así como un valor “zero” que se usa cuando se especifica un valor ilegal que MySQL no puede representar. El tipo `TIMESTAMP` tiene un comportamiento automático especial, descrito posteriormente.

Desde MySQL 5.0.2, MySQL da advertencias/errores si trata de insertar una fecha ilegal. Puede hacer que MySQL acepte ciertas fechas, tales como `'1999-11-31'`, usando el modo SQL `ALLOW_INVALID_DATES` . (Antes de 5.0.2, este modo era el comportamiento por defecto de MySQL). Esto es útil cuando quiere almacenar el valor “posiblemente erróneo” que el usuario especifica (por ejemplo, en un formulario web) en la base de datos para un posterior procesamiento. En este modo, MySQL sólo verifica que el mes esté en el rango de 0 a 12 y que el día esté en el rango de 0 a 31. Estos rangos incluyen cero ya que MySQL permite almacenar fechas cuando el día o el mes son cero en columnas `DATE` o `DATETIME` . Esto es muy útil para aplicaciones que necesiten almacenar una fecha de nacimiento para la que no sepa la fecha exacta. En este caso, simplemente almacena la fecha como `'1999-00-00'` o `'1999-01-00'`. Si almacena valores similares a estos, no debe esperar obtener resultados correctos para funciones tales como `DATE_SUB()` or `DATE_ADD` que necesitan fechas completas. (Si no quiere permitir ceros en las fechas, puede usar el modo SQL `NO_ZERO_IN_DATE`).

MySQL permite almacenar `'0000-00-00'` como “fecha de pruebas” (si no está usando el modo SQL `NO_ZERO_DATE`). Esto es mejor que usar (y usa menos espacio de datos e índice) que usar valores `NULL` .

Modificando la variable de sistema `sql_mode` al modo apropiado, puede especificar exactamente qué tipos de datos quiere soportar con MySQL. Consulte Sección 5.3.2, “El modo SQL del servidor”.

Aquí hay algunas consideraciones generales a tener en cuenta cuando se trabaja con tipos de fecha y hora:

- MySQL muestra los valores para una fecha u hora en un formato de salida estándar, pero trata de intepretar una variedad de formatos para los valores de entrada que se proporcionan (por ejemplo, cuando especifica un valor para asignar o comparar con un tipo fecha u hora). Sólo los formatos descritos en las siguientes secciones son soportados. Se espera la entrada de valores legales. Si se usan otros formatos pueden ocurrir resultados imprevisibles.

- Las fechas con años de dos dígitos son ambituas, ya que no se sabe el siglo. MySQL interpreta los años de dos dígitos usando las siguientes reglas:

  - Los años del rango `70-99` se convierten en `1970-1999`.

  - Los años del rango `00-69` se convierten en `2000-2069`.

- Aunque MySQL trata de interpretar los valores con varios formatos, las fechas siempre deben darse en el orden año-mes-día (por ejemplo, `'98-09-04'`), en lugar del formato mes-día-año o día-mes-año usados comunmente (por ejemplo, `'09-04-98'`, `'04-09-98'`).

- MySQL convierte automáticamente una fecha u hora a un número si el valor se usa en un contexto numérico y viceversa.

- Cuando MySQL encuentra un valor para fecha u hora que está fuera de rango o es ilegal para el tipo (como se describe al inicio de la sección), lo convierte al valor “cero” para ese tipo. La excepción es que los valores fuera de rango del tipo `TIME` se reemplazan por el valor límite de rango apropiado para el tipo `TIME` .

  La siguiente tabla muestra el formato del valor “cero” para cada tipo. Tenga en cuenta que el uso de estos valores produce mensajes de advertencia si el modo SQL `NO_ZERO_DATE` está activado.

Tipo de Columna	“Cero” Valor
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	00000000000000
TIME	'00:00:00'
YEAR	0000

- Los valores “cero” son especiales, pero puede almacenarlos o referirse a ellos explícitamente usando los valores mostrados en la tabla. También puede hacerlo usando los valores '0' o 0, que son más sencillos de escribir.
- Los valores de fecha u hora “cero” usados a través de MyODBC se convierten automáticamente a NULL en MyODBC 2.50.12 y posterior, ya que ODBC no puede tratar estos valores.

11.3.1. Los tipos de datos DATETIME, DATE y TIMESTAMP

Los tipos DATETIME, DATE, and TIMESTAMP están relacionados. Esta sección describe sus características, en qué se parecen y en qué difieren.

El tipo DATETIME se usa cuando necesita valores que contienen información de fecha y hora. MySQL recibe y muestra los valores DATETIME en formato 'YYYY-MM-DD HH:MM:SS' . El rango soportado es de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'. (“Soportado” significa que aunque valores anteriores pueden funcionar, no hay garantías)

El tipo DATE se usa cuando necesita sólo un valor de fecha, sin una parte de hora. MySQL recibe y muestra los valores DATE en formato 'YYYY-MM-DD' . El rango soportado es de '1000-01-01' a '9999-12-31'.

El tipo TIMESTAMP tiene varias propiedades, en función de la versión de MySQSL y el modo SQL que esté ejecutando el servidor. Estas propiedades se describen posteriormente en esta sección.

Puede especificar valores DATETIME, DATE, y TIMESTAMP usando cualquier de los siguientes formatos:

- Como cadena de caracteres en formato 'YYYY-MM-DD HH:MM:SS' o 'YY-MM-DD HH:MM:SS' . Una sintaxis “relajada” se permite: Cualquier carácter de puntuación puede usarse como delimitador entre partes de fecha o de hora. Por ejemplo, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45', y '98@12@31 11^30^45' son equivalentes.
- Como cadena de caracteres en formato 'YYYY-MM-DD' or 'YY-MM-DD' . Se permite una sintaxis “relajada” . Por ejemplo, '98-12-31', '98.12.31', '98/12/31', y '98@12@31' son equivalentes.
- Como cadena de caracteres sin delimitadores en formato 'YYYYMMDDHHMMSS' o 'YYMMDDHHMMSS', mientras que la cadena de caracteres tenga sentido como fecha. Por ejemmplo, '19970523091528' y '970523091528' se interpretan como '1997-05-23 09:15:28', pero '971122129015' es ilegal (tiene una parte de minutos sin sentido) y se convierte en '0000-00-00 00:00:00'.
- Como cadena de caracteres sin delimitadores en formato 'YYYYMMDD' o 'YYMMDD' , mientras que el cadena de caracteres tenga sentido como fecha. Por ejemplo, '19970523' y '970523' se interpretan como '1997-05-23', pero '971332' es ilegal (tiene una parte de mes y día sin sentido) y se convierte en '0000-00-00'.
- Como número en formato YYYYMMDDHHMMSS o YYMMDDHHMMSS, mientras que el número tenga sentido como fecha. Por ejemplo, 19830905132800 y 830905132800 se interpretan como '1983-09-05 13:28:00'.

- Como número en formato YYYYMMDD o YYMMDD, mientras que el número tenga sentido como fecha. Por ejemplo, 19830905 y 830905 se interpretan como '1983-09-05'.
- Como resultado de una función que retorne un valor acceptable en un contexto DATETIME, DATE, o TIMESTAMP , como NOW() o CURRENT_DATE.

Los valores ilegales de DATETIME, DATE, o TIMESTAMP se convierten al valor “cero” del tipo apropiado ('0000-00-00 00:00:00', '0000-00-00', o 00000000000000).

Para valores especificados como cadenas de caracteres que incluyan partes de fecha delimitadas, no es necesario especificar dos dígitos para valores de mes o día menores que 10. '1979-6-9' es lo mismo que '1979-06-09'. Similarmente, para valores especificados como cadenas de caracteres que incluyan delimitadores para la parte de hora, no es necesario especificar dos dígitos para horas, minutos o segundos menores que 10. '1979-10-30 1:2:3' es lo mismo que '1979-10-30 01:02:03'.

Los valores especificados como números deben tener una longitud de 6, 8, 12, o 14 dígitos. Si un número tiene una longitud de 8 o 14 dígitos, se asume que está en formato YYYYMMDD o YYYYMMDDHHMMSS y que el año lo dan los primeros 4 dígitos. Si el número tiene 6 o 12 dígitos de longitud, se asume que está en formato YYMMDD o YYMMDDHHMMSS y que el año lo dan los primeros 2 dígitos. A los números que no tengan estas longitudes se les añaden ceros a la izquierda hasta la longitud más cercana permitida.

Los valores especificados como cadenas de caracteres no delimitadas se interpretan usando su longitud. Si la cadena de caracteres tiene longitud 8 o 14, el año se asume como dado por los primeros 4 caracteres. En el resto de caso, se supone que el año lo dan los primeros 2 caracteres. La cadena de caracteres se interpreta de izquierda a derecha para encontrar el año, mes, día, hora, minuto y segundo, para tantas partes como representa la cadena de caracteres. Esto significa que no debe usar cadenas de caracteres con menos de 6 caracteres. Por ejemplo, si especifica '9903', pensando que representa Marzo, 1999, MySQL inserta un valor “cero” en la tabla. Esto es porque el valor de año y mes son 99 y 03, pero la parte de día no se encuentra, así que el valor no es una fecha legal. Sin embargo, puede especificar explícitamente un valor de cero para representar partes de día y mes. Por ejemplo, puede usar '990300' para insertar el valor '1999-03-00'.

Puede asignar valores de un tipo a un objeto de un tipo diferente hasta un límite. Sin embargo, hay algunas alteraciones del valor o pérdidas de información:

- Si asigna un valor DATE a un objeto DATETIME o TIMESTAMP, la parte de hora del valor resultante se cambia a '00:00:00' ya que el valor DATE no contiene información temporal.
- Si asigna un valor DATETIME o TIMESTAMP a un objeto DATE, la parte temporal del valor resultante se borra porque el tipo DATE no tiene información temporal.
- Tenga en cuenta que aunque DATETIME, DATE, y TIMESTAMP pueden especificarse usando el mismo conjunto de formatos, los tipos no tienen el mismo rango de valores. Por ejemplo, TIMESTAMP no pueden ser anteriores a 1970 o posteriores a 2037. Esto significa que una fecha como '1968-01-01', que sería legal como DATETIME o DATE no es un valor válido TIMESTAMP y se convierte a 0 si se asigna a un objeto de este tipo.

Tenga en cuenta ciertas cosas al especificar valores temporales:

- El formato relajado para valores especificados como cadenas de caracteres puede ser problemático. Por ejemplo, un valor como '10:11:12' puede parecer una hora por el delimitador ':', pero si se usa en un contexto de fecha se interpreta como '2010-11-12'. El valor '10:45:15' se convierte a '0000-00-00' ya que '45' no es un mes legal.
- El servidor MySQL realiza sólo chequeo básico de la validez de las fechas: Los rangos para año, mes y día son de 1000 a 9999, 00 a 12, y 00 a 31, respectivamente. Cualquier fecha que contenga partes

fuera de estos rangos está sujeta a conversión a '0000-00-00'. Tenga en cuenta que esto permite almacenar fechas inválidas como '2002-04-31'. Para asegurar que una fecha es válida, haga una comprobación en su aplicación.

- Fechas con valores de año de dos dígitos son ambíguas porque no se conoce el siglo. MySQL interpreta los años de dos dígitos usando las siguientes reglas:
  - Los valores de años en el rango 00-69 se convierten a 2000-2069.
  - Los valores de años en el rango 70-99 se convierten a 1970-1999.

11.3.1.1. Propiedades de TIMESTAMP desde MySQL 4.1

Nota: En antiguas versiones de MySQL (antes de la 4.1), las propiedades de las columnas TIMESTAMP difieren significativamente en muchas cosas de lo que se describe en esta sección. Si necesita convertir datos TIMESTAMP antiguos para que funcionen con MySQL 5.0, asegúrese de consultar Manual de referencia de MySQL 4.1 para más detalles.

En MySQL 5.0, TIMESTAMP se muestran en el mismo formato que las columnas DATETIME. En otras palabras, el ancho de muestra se limita a 19 caracteres, y el formato es YYYY-MM-DD HH:MM:SS.

El servidor MySQL puede ejecutarse en modo MAXDB . Cuando el servidor corre en este modo, TIMESTAMP es idéntico a DATETIME. Esto es, si el servidor está ejecutándose en modo MAXDB cuando se crea una tabla, las columnas TIMESTAMP se crean como columnas DATETIME . Como resultado, tales columnas usan el formato de salida de DATETIME, tienen el mismo rango de valores, y no hay inicialización automática o actualización de la fecha y hora actual.

Para activar el modo MAXDB, cambie el modo SQL del servido aMAXDB cuando arranque usando la opción --sql-mode=MAXDB o cambiando en tiempo de ejecución la variable global sql_mode:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

Un cliente puede hacer que el servidor se ejecute en modo MAXDB para sus propias conexiones como se muestra:

```
mysql> SET SESSION sql_mode=MAXDB;
```

Desde MySQL 5.0.2, MySQL no acepta valores timestamp que incluyan cero en la columna de día u hora o valores que no sean fechas válidas. La única excepción es el valor especial '0000-00-00 00:00:00'.

En MYSQL 5.0, tiene considerable flexibilidad para determinar cuando se actualiza e inicializa automáticamente TIMESTAMP y qué columna debe tener ese comportamiento:

- Puede asignar la fecha y hora actual como el valor por defecto y el valor de actualización automático, como se hacía anteriormente. Pero es posible tener sólo uno u otro comportamiento automático, o ninguno de ellos. (No es posible tener un comportamiento para una columna y el otro para la otra columna.)
- Puede especificar qué columna TIMESTAMP inicializar o actualizar con la fecha y hora actuales. Ya no hace falta que sea la primera columna TIMESTAMP .

Tenga en cuenta que la información en la siguiente discusión se aplica a columnas TIMESTAMP sólo para tablas no creadas con el modo MAXDB activado. (Como se menciona anteriormente, el modo MAXDB hace que las columnas se creen como columnas DATETIME .) Las reglas que gobiernan la inicialización y actualización de columnas TIMESTAMP en MySQL 5.0 son las siguientes:

- Si un valor DEFAULT se especifica para la primera columna TIMESTAMP en una tabla, no se ignora. El valor por defecto puede ser CURRENT_TIMESTAMP o una fecha y hora constante.
- DEFAULT NULL es lo mismo que DEFAULT CURRENT_TIMESTAMP para la *primera* columna TIMESTAMP. Para cualquier otra columna TIMESTAMP, DEFAULT NULL se trata como DEFAULT 0.
- Cualquier columna TIMESTAMP individual en una tabla puede actualizarse e inicializarse con la fecha y hora actual automáticamente.
- En un comando CREATE TABLE, la primera columna TIMESTAMP puede declararse de cualquiera de las siguientes formas:
  - Con las cláusulas DEFAULT CURRENT_TIMESTAMP y ON UPDATE CURRENT_TIMESTAMP, la columna tiene la fecha y hora actual como su valor por defecto, y se actualiza automáticamente.
  - Sin las cláusulas DEFAULT ni ON UPDATE, es lo mismo que DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP.
  - Con la cláusula DEFAULT CURRENT_TIMESTAMP y sin ON UPDATE, la columna tiene la fecha y hora actual como valor por defecto pero no se actualiza automáticamente.
  - Sin cláusula DEFAULT y con cláusula ON UPDATE CURRENT_TIMESTAMP, la columna tiene por defecto 0 y se actualiza automáticamente.
  - Con un valor constante DEFAULT, la columna tiene el valor dado por defecto. Si la columna tiene una cláusula ON UPDATE CURRENT_TIMESTAMP se actualiza automáticamente, de otro modo no lo hace.

  En otras palabras, puede usar la fecha y hora actuales para el valor inicial y el valor de actualización automática, o uno de ellos o ninguno. (Por ejemplo, puede especificar ON UPDATE para activar actualización automática sin tener la columna inicializada .)
- Cualquiera de CURRENT_TIMESTAMP, CURRENT_TIMESTAMP(), o NOW() puede usarse en las cláusulas DEFAULT y ON UPDATE . Todas tienen el mismo efecto.

  El orden de los dos atributos no importa. Si se especifican DEFAULT y ON UPDATE para una columna TIMESTAMP, cualquiera puede preceder al otro.

  Ejemplo. Estos comandos son equivalentes:

```
CREATE TABLE t (ts TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
DEFAULT CURRENT_TIMESTAMP);
```

- Para especificar el valor por defecto o actualización automática para una columna TIMESTAMP distinta a la primera, debe suprimir la actualización e inicialización automática de la primera columna TIMESTAMP asignándole explícitamente una valor constante DEFAULT (por ejemplo, DEFAULT 0 o DEFAULT '2003-01-01 00:00:00'). Luego, para la otra columna TIMESTAMP, las reglas son las mismas que para la misma columna TIMESTAMP, excepto que no puede omitir ambas cláusulas DEFAULT y ON UPDATE . Si lo hace, no habrá inicialización ni actualización automática.

  Ejemplo, estos comandos son equivalentes:

```
CREATE TABLE t (
    ts1 TIMESTAMP DEFAULT 0,
    ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
                  ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (
    ts1 TIMESTAMP DEFAULT 0,
    ts2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
                  DEFAULT CURRENT_TIMESTAMP);
```

En MySQL 5.0, puede asignar la zona horaria actual para cada conexión, como se describe en Sección 5.9.8, “Soporte de zonas horarias en el servidor MySQL”. Los valores TIMESTAMP se almacenan en UTC, convirtiéndose desde la zona horaria actual para almacenamiento, y volviéndose a convertir a la zona horaria actual al mostrarse. Mientras la zona horaria permanezca constante, puede obtener el mismo valor que hay almacenado. Si almacena un valor TIMESTAMP, cambia la zona horaria y luego rescata el valor, es diferente que el valor almacenado. Esto ocurre porque no se usa la misma zona horaria para la conversión en ambas direcciones. La zona horaria actual está disponible en la variable de sistema time_zone.

Puede incluir el atributo NULL en la definición de una columna TIMESTAMP para permitir que la columna contenga valores NULL . Por ejemplo:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

Si el atributo NULL no se especifica, asignar el valor NULL a la columna resulta en que se almacena la hora y fecha actuales. Tenga en cuenta que una columna TIMESTAMP que permita valores NULL **no** almacenará la fecha y hora actual a no ser que su valor por defecto se defina como CURRENT_TIMESTAMP, **o** NOW() o CURRENT_TIMESTAMP se inserte en la columna. En otras palabras, una columna TIMESTAMP definida como NULL se actualizará automáticamente sólo si se crea usando una definición como las siguientes:

```
CREATE TABLE t (ts NULL DEFAULT CURRENT_TIMESTAMP);
```

De otro modo - esto es, si la columna TIMESTAMP se define usando NULL pero no usando DEFAULT TIMESTAMP, como se muestra aquí...

```
CREATE TABLE t1 (ts NULL DEFAULT NULL);
CREATE TABLE t2 (ts NULL DEFAULT '0000-00-00 00:00:00');
```

...entonces debe insertar el valor explícitamente correspondiente a la fecha y hora actuales. Por ejemplo:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

11.3.2. El tipo TIME

MySQL devuelve y muestra los valores TIME en formato 'HH:MM:SS' (o formato 'HHH:MM:SS' para valores de hora grandes). TIME tiene rango de '-838:59:59' a '838:59:59'. La razón por la que la parte de hora puede ser tan grande es que el tipo TIME puede usarse no sólo para representar una hora del día (que debe ser menor a 24 horas), pero también el tiempo transcurrido o un intervalo de tiempo entre dos eventos (que puede ser mucho mayor a 24 horas, o incluso negativo).

Puede especificar valores TIME en una variedad de formatos:

- Como cadena de caracteres en formato 'D HH:MM:SS.fracción'. También puede usar una de las siguientes sintaxis “relajadas” : 'HH:MM:SS.fracción', 'HH:MM:SS', 'HH:MM', 'D HH:MM:SS', 'D

HH:MM', 'D HH', o 'SS'. Aquí D representa días y puede tener un valor de 0 a 34. Tenga en cuenta que MySQL no almacena la fracción (todavía).

- Como cadena de caracteres sin delimitadores en formato 'HHMMSS', mientras que tenga sentido como hora. Por ejemplo, '101112' se entiende como '10:11:12', pero '109712' es ilegal (no tiene una parte de minutos correcta) y pasa a ser '00:00:00'.
- Como número en formato HHMMSS, mientras tenga sentido como hora. Por ejemplo, 101112 se entiende como '10:11:12'. Los siguientes formatos alternativos también se entienden: SS, MMSS, HHMMSS, HHMMSS.fracción. Tenga en cuenta que MySQL no almacena la fracción (todavía).
- Como resultado de una función que retorna un valor que es aceptable en un contexto TIME, tal como CURRENT_TIME.

Para valores TIME especificados como cadenas de caracteres que incluyan un delimitador de las partes de hora, no es necesario especificar dos dígitos para horas, minutos o segundos que tengan un valor inferior a 10. '8:3:2' es lo mismo que '08:03:02'.

Tenga cuidado con asignar valores abreviados a una columna TIME . Sin comas, MySQL interpreta los valores asumiendo que los dos dígitos más a la derecha representan segundos. (MySQL interpreta valores TIME como tiempo transcurrido en lugar de horas del día.) Por ejemplo puede pensar que '1112' y 1112 significan '11:12:00' (12 minutos tras las 11 en punto), pero MySQL los interpreta como '00:11:12' (11 minutos, 12 segundos). Similarmente, '12' y 12 se interpretan como '00:00:12'. Los valores TIME con comas, por contrario, se tratan siempre como hora del día. Esto es, '11:12' significa '11:12:00', no '00:11:12'.

Los valores fuera del rango de TIME pero que son legales se cambian por el valor límite de rango más cercano. Por ejemplo '-850:00:00' y '850:00:00' se convierten en '-838:59:59' y '838:59:59'.

Valores TIME ilegales se convierten a '00:00:00'. Tenga en cuenta que como '00:00:00' es un valor TIME legal, no hay forma de decir si un valor '00:00:00' almacenado en una tabla, se insertó como '00:00:00' o como valor ilegal.

11.3.3. El tipo de datos YEAR

El tipo YEAR es un tipo de un byte usado para representar años.

MySQL devuelve y muestra los valores YEAR en formato YYYY . El rango es de 1901 a 2155.

Puede especificar los valores YEAR en una variedad de formatos:

- Como cadena de caracteres de cuatro dígitos en el rango de '1901' a '2155'.
- Como número de cuatro dígitos en el rango de 1901 a 2155.
- Como cadena de caracteres de dos dígitos en el rango de '00' a '99'. Los valores en los rangos de '00' a '69' y de '70' a '99' se convierten en valores YEAR en el rango de 2000 a 2069 y de 1970 a 1999.
- Como número de dos dígitos en el rango de 1 a 99. Los valores en los rangos de 1 a 69 y de 70 a 99 se convierten en valores YEAR en los rangos de 2001 a 2069 y de 1970 a 1999. Tenga en cuenta que el rango para números de dos dígitos es ligeramente distinto del rango para cadenas de caracteres de dos dígitos, ya que no especifica el cero directamente como número y tiene que ser interpretado como 2000. Debe especificarlo como cadena de caracteres '0' o '00' o se interpreta como 0000.
- Como resultado de una función que retorne un valor que se acepte en un contexto YEAR, como NOW().

Valores `YEAR` ilegales se convierten en `0000`.

11.3.4. Efecto 2000 (Y2K) y tipos de datos

MySQL no tiene problemas con el año 2000 (Y2K) (consulte Sección 1.4.5, “Conformidad con el efecto 2000”), pero los valores de entrada presentados a MySQL pueden tenerlos. Cualquier entrada con valores de años de dos dígitos es ambigua, ya que no se conoce el siglo. Tales valores deben interpretarse en forma de cuatro dígitos, ya que MySQL los almacena internamente usando cuatro dígitos.

Para tipos `DATETIME`, `DATE`, `TIMESTAMP`, y `YEAR`, MySQL interpreta las fechas con valores de año ambíguos usando las siguientes reglas:

- Años en el rango `00-69` se convierten a `2000-2069`.
- Años en el rango `70-99` se convierten a `1970-1999`.

Recuerde que estas reglas proporcionan sólo suposiciones razonables sobre lo que significan los valores. Si el heurístico usado por MySQL no produce los valores correctos, debe proporcionar entrada no ambígua con años de cuatro dígitos.

`ORDER BY` ordena valores `TIMESTAMP` o `YEAR` correctamente que tengan años de dos digitos.

Algunas funciones como `MIN()` y `MAX()` convierten un `TIMESTAMP` o `YEAR` a número. Esto significa que un valor con un año de dos dígitos no funciona correctamente con estas funciones. En este caso la solución es convertir `TIMESTAMP` o `YEAR` a formato de cuatro dígitos o usar algo como `MIN(DATE_ADD(timestamp,INTERVAL 0 DAYS))`.

11.4. Tipos de cadenas de caracteres

Los tipos de cadenas de caracteres son `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, y `SET`. Esta sección describe cómo funcionan estos tipos y cómo usarlo en sus consultas.

11.4.1. Los tipos `CHAR` y `VARCHAR`

Los tipos `CHAR` y `VARCHAR` son similares, pero difieren en cómo se almacenan y recuperan. Desde MySQL 5.0.3, también difieren en la longitud máxima y en cómo se tratan los espacios finales.

Los tipos `CHAR` y `VARCHAR` se declaran con una longitud que indica el máximo número de caracteres que quiere almacenar. Por ejemplo, `CHAR(30)` puede almacenar hasta 30 caracteres.

La longitud de una columna `CHAR` se fija a la longitud que se declara al crear la tabla. La longitud puede ser cualquier valor de 0 a 255. Cuando los valores `CHAR` se almacenan, se añaden espacios a la derecha hasta las longitud específica. Cuando los valores `CHAR` se recuperan, estos espacios se borran.

Los valores en columnas `VARCHAR` son cadenas de caracteres de longitud variable. En MySQL 5.0, la longitud puede especficarse de 0 a 255 antes de MySQL 5.0.3, y de 0 a 65,535 en 5.0.3 y versiones posteriores. (La máxima longitud efectiva de un `VARCHAR` en MySQL 5.0 se determina por el tamaño de registro máximo y el conjunto de caracteres usados. La longitud máxima total es de 65,532 bytes.)

En contraste con `CHAR`, `VARCHAR` almacena los valores usando sólo los caracteres necesarios, más un byte adicional para la longitud (dos bytes para columnas que se declaran con una longitud superior a 255).

Los valores `VARCHAR` no se cortan al almacenarse. El tratamiento de espacios al final depende de la versión. Desde MySQL 5.0.3, los espacios finales se almacenan con el valor y se retornan, según el estándar SQL. Antes de MySQL 5.0.3, los espacios finales se eliminan de los valores cuando se

almacenan en una columna VARCHAR, esto significa que los espacios también están ausentes de los valores retornados.

Durante el almacenamiento y la recuperación de valores no hace ninguna conversión de mayúsculas y minúsculas.

Si asigna un valor a una columna CHAR o VARCHAR que exceda la longitud máxima de la columna, el valor se trunca. Si los caracteres truncados no son espacios, se genera una advertencia. Puede hacer que aparezca un error en lugar de una advertencia usando modo SQL estricto. Consulte Sección 5.3.2, “El modo SQL del servidor”.

Antes de MySQL 5.0.3, si necesita un tipo de datos para el que no se borren los espacios finales, considere usar un tipo BLOB o TEXT . También, si quiere almacenar valores binarios como resultados de cifrado o compresión que puedan contener valores byte arbitrarios, use una columna BLOB en lugar de CHAR o VARCHAR, para evitar problemas potenciales con eliminación de espacios finales que puedan cambiar los valores de los datos.

La siguiente tabla ilustra las diferencias entre los dos tipos de columnas mostrando el resultado de almacenar varios valores de cadenas de caracteres en columnas CHAR(4) y VARCHAR(4) :

Valor	CHAR(4)	Almacenamiento necesario	VARCHAR(4)	Almacenamiento necesario
''	' '	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

Los valores retornados de las columnas CHAR(4) y VARCHAR(4) son los mismos en cada caso, ya que los espacios finales se eliminan en la recuperación de valores CHAR.

En MySQL 5.0, los valores en columnas CHAR y VARCHAR se almacenan y comparan según la colación del conjunto de caracteres asignado a la columna.

CHAR BYTE es un alias para CHAR BINARY. Existe por cuestión de compatibilidad.

El atributo ASCII asigna el conjunto de caracteres latin1 a una columna CHAR . El atributo UNICODE asigna el conjunto de caracteres ucs2 .

MySQL puede cambiar silenciosamente el tipo de una columna CHAR o VARCHAR en tiempo de creación. Consulte Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”.

11.4.2. Los tipos BINARY y VARBINARY

Los tipos BINARY y VARBINARY son similares a CHAR y VARCHAR, excepto que contienen cadenas de caracteres binarias en lugar de cadenas de caracteres no binarias. Esto es, contienen cadenas de bytes en lugar de cadenas de caracteres. Esto significa que no tienen conjunto de caracteres asignado, y la comparación y ordenación se basa en los valores numéricos de los valores de los bytes.

La longitud máxima disponible es la máxima para BINARY t VARBINARY como para CHAR y VARCHAR, excepto que la longitud para BINARY y VARBINARY es una longitud en bytes en lugar de en caracteres.

El tratamiento de los espacios finales es el mismo para BINARY y VARBINARY como lo es para CHAR y VARCHAR. Cuando se almacenan los valores BINARY, se rellenan con espacios a la derecha hasta la

longitud especificada. Cuando los valores BINARY se recuperan, los espacios finales se eliminan. Para VARBINARY, los espacios finales se eliminan cuando los valores se almacenan. Desde MySQL 5.0.3, los espacios finales se mantienen. Debe considerar estas características si planea usar estos tipos de datos para almacenar datos binarios que deban acabar con espacios.

En MySQL 5.0, BINARY y VARBINARY son tipos de datos distintos. Para CHAR(*M*) BINARY y VARCHAR(*M*) BINARY, el atributo BINARY hace que se use la colación binaria para la columna, pero la columna no contiene cadenas de caracteres no binarios en lugar de cadenas binarias de bytes. Por ejemplo CHAR(5) BINARY se trata como CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin, asumiendo que el conjunto de caracteres por defecto es latin1.

11.4.3. Los tipos BLOB y TEXT

Un BLOB es un objeto binario que puede tratar una cantidad de datos variables. Los cuatro tipos BLOB son TINYBLOB, BLOB, MEDIUMBLOB, y LONGBLOB. Difieren sólo en la longitud máxima de los valores que pueden tratar.

Los cuatro tipos TEXT son TINYTEXT, TEXT, MEDIUMTEXT, y LONGTEXT. Se corresponden a los cuatro tipos BLOB y tienen las mismas longitudes y requerimientos de almacenamiento.

Consulte Sección 11.5, “Requisitos de almacenamiento según el tipo de columna”.

Las columnas BLOB se tratan como cadenas de caracteres binarias (de bytes). Las columnas TEXT se tratan como cadenas de caracteres no binarias (de carácateres). Las columnas BLOB no tienen conjunto de caracteres, y la ordenación y la comparación se basan en los valores numéricos de los bytes. Las columnas TEXT tienen un conjunto de caracteres y se ordenan y comparan en base de la colación del conjunto de caracteres asignada a la columna desde MySQL 4.1.

No hay conversión de mayúsculas/minúsculas para columnas TEXT o BLOB durante el almacenamiento o la recuperación.

Si asiguna un valor a una columna BLOB o TEXT que exceda la longitud máxima del tipo de la columna, el valor se trunca. Si los caracteres truncados no son espacios, aparece una advertencia. Puede hacer que aparezca un error en lugar de una advertencia usando el modo SQL estricto. Consulte Sección 5.3.2, “El modo SQL del servidor”.

En la mayoría de aspectos, puede tratar una columna BLOB como VARBINARY que puede ser tan grande como desee. Similarmente, puede tratar columnas TEXT como VARCHAR. BLOB y TEXT difieren de VARBINARY y VARCHAR en los siguientes aspectos::

- No se eliminan espacios al final para columnas BLOB y TEXT cuando los valores se almacenan o recuperan. Antes de MySQL 5.0.3, esto difiere de VARBINARY y VARCHAR, para los que se eliminaban los epacios al final cuando se almacenaban.

  Tenga en cuenta que TEXT realiza comparación espacial extendida para coincidir con el objeto comparado, exactamente como CHAR y VARCHAR.

- Para índices en columnas BLOB y TEXT, debe especificar una longitud de prefijo para el índice. Para CHAR y VARCHAR, la longitud de prefijo es opciona. Consulte Sección 7.4.3, “Índices de columna”.

- BLOB y TEXT no pueden tener valores DEFAULT .

En MySQL 5.0, LONG y LONG VARCHAR se mapean con el tipo de datos MEDIUMTEXT. Esto existe por compatibilidad. Si usa el atributo BINARY con el tipo de columna TEXT, se asigna la colación binaria del conjunto de caracteres a la columna.

MySQL Connector/ODBC define los valores BLOB como LONGVARBINARY y valores TEXT como LONGVARCHAR.

Como los valores BLOB y TEXT pueden ser extremadamente grandes, puede encontrar algunas restricciones al usarlos:

- Sólo los primeros max_sort_length bytes de la columna se usan al ordenar. El valor por defecto de max_sort_length es 1024; este valor puede cambiarse usando la opción --max_sort_length al arrancar el servidor mysqld . Consulte Sección 5.3.3, “Variables de sistema del servidor”.

  Puede hacer que haya más bytes significativos al ordenar o agrupar incrementando el valor de max_sort_length en tiempo de ejecución. Cualquier cliente puede cambiar el valor de su variable de sesión max_sort_length :

  ```
  mysql> SET max_sort_length = 2000;
  mysql> SELECT id, comment FROM tbl_name
      -> ORDER BY comment;
  ```

  Otra forma de usar GROUP BY o ORDER BY en una columna BLOB o TEXT conteniendo valores grandes cuando quiere que más de max_sort_length bytes sean significativos es convertir el valor de la columna en un objeto de longitud fija. La forma estándar de hacerlo es con la función SUBSTRING . Por ejemplo, el siguiente comando causa que 2000 bytes de la columna comment se tengan en cuenta para ordenación:

  ```
  mysql> SELECT id, SUBSTRING(comment,1,2000) FROM tbl_name
      -> ORDER BY SUBSTRING(comment,1,2000);
  ```

- El tamaño máximo de un objeto BLOB o TEXT se determina por su tipo, pero el valor máximo que puede transmitir entre el cliente y el servidor viene determinado por la cantidad de memoria disponible y el tamaño de los buffers de comunicación. Puede cambiar el tamaño de los buffers de comunicación cambiando el valor de la variable max_allowed_packet, pero debe hacerlo para el servidor y los clientes . Por ejemplo, mysql y mysqldump le permite cambiar el valor de la variable del cliente max_allowed_packet . Consulte Sección 7.5.2, “Afinar parámetros del servidor”, Sección 8.3, “La herramienta intérprete de comandos mysql”, y Sección 8.7, “El programa de copia de seguridad de base de datos mysqldump”.

Cada valor BLOB o TEXT se representa internamente como un objeto a parte. Esto se hace en contraste con todos los otros tipos de columnas, para los que el almacenamiento se hace una vez por columna cuando se abre la tabla.

11.4.4. El tipo de columna ENUM

Un ENUM es un objeto de cadenas de caracteres con un valor elegido de una lista de valores permitidos que se enumeran explícitamente en la especificación de columna en tiempo de creación de la tabla.

El valor puede ser la cadena vacía ('') o NULL bajo ciertas circunstancias:

- Si inserta un valor inválido en un ENUM (esto es, una cadena de caracteres no presente en la lista de valores permitidos), la cadena vacía se inserta en lugar de un valor especial de error. Esta cadena puede distinguirse de una cadena vacía “normal” por el hecho que esta cadena tiene un valor numérico 0. Más información posteriormente.

- Si se declara una columna ENUM para permitir NULL, el valor NULL es un valor legal para la columna, y el valor por defecto es NULL. Si una columna ENUM se declara NOT NULL, su valor por defecto es el primer elemento de la lista de valores permitidos.

Cada valor de la enumeración tiene un índice:

- Los valores de la lista de elementos permitidos en la especificación de la columna se numeran empezando por 1.
- El valor de índice de la cadena errónea es 0. Esto significa que puede usar el siguiente comando SELECT para encontrar registros con el valor inválido ENUM asignado:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- El índice del valor NULL es NULL.

Por ejemplo, una columna especificada como ENUM('one', 'two', 'three') puede tener cualquiera de los valores mostrados aquí. El índice de cada valor se muestra:

Valor	Índice
NULL	NULL
''	0
'one'	1
'two'	2
'three'	3

Una enumeración puede tener un máximo de 65,535 elementos.

Los espacios finales se borran automáticamente para valores ENUM miembros cuando se crea la tabla.

Cuando se reciben, los valores almacenados en una columna ENUM se muestran usando el formato de mayúsculas/minúsculas usado en la definición de la columna. En MySQL 4.1.1, las columnas ENUM pueden recibir un conjunto de caracteres y colación. Para colaciones binarias o sensibles a mayúsculas/minúsculas, el formato se tiene en cuenta al asignar valores a la columna.

Si recibe un valor ENUM en contexto numérico, se retorna el índice del valor. Por ejemplo, puede recibir valores numéricos de una columna ENUM así:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

Si almacena un número en una columna ENUM, el número se trata como índice, y el valor almacenado es el miembro de la enumeración con ese índice. (Sin embargo, esto no funciona con LOAD DATA, que trata toda la entrada como cadenas de caracteres.) No es recomendable definir una columna ENUM con valores de enumeración que parezcan números, ya que esto puede causar confusión. Por ejemplo, la siguiente columna tiene miembros de enumeración con valores de '0', '1', y '2', pero valores de índice 1, 2, y 3:

```
numbers ENUM('0','1','2')
```

Los valores ENUM se ordenan según el order en que se enumeran los mienbros en la especificación de la columna. (En otras palabras, los valores ENUM se ordenan según sus números de índice.) Por ejemplo, 'a' se ordena antes que 'b' para ENUM('a', 'b'), pero 'b' se ordena antes de 'a' para ENUM('b', 'a'). La cadena vacía se ordena antes de las cadenas no vacías, y los valores NULL se ordenan antes de todos los otros valores de la enumeración. Para evitar resultados inesperados, especifique la lista ENUM en orden alfabético. También puede usar GROUP BY CAST(col AS VARCHAR) o GROUP BY CONCAT(col) para asegurarse que la columna se ordena léxicamente en lugar de por número de índice.

Si quiere determinar todos los valores posibles para una columna ENUM, use SHOW COLUMNS FROM *tbl_name* LIKE *enum_col* y parsee la definición de ENUM en la segunda columna de la salida.

11.4.5. El tipo SET

Un SET es un objeto de cadenas de caracteres que tiene cero o más valores, cada uno de ellos debe elegirse de una lista de valores posibles especificada cuando se crea la tabla. Los valores de columnas SET que consisten de múltiples miembros del conjunto se especifican con los miembros separados por comas (','). Una consecuencia de esto es que los miembros de SET no pueden contener comas ellos mismos.

Por ejemplo, una columna especificada como SET('one', 'two') NOT NULL puede tener cualquiera de estos valores:

```
''
'one'
'two'
'one,two'
```

Un SET puede tener un máximo de 64 miembros distintos.

Los espacios finales se borran automáticamente de los miembros de un SET cuando se crea la tabla.

Cuando se recuperan, los valors almacenados en una columna SET se muestran usando la sensibilidad de mayúsculas/minúsculas usando en la definición de la columna. En MySQL 5.0, las columnas SET pueden tener un conjunto de caracteres y colación. Para colaciones binarias o sensibles a mayúsculas/minúsculas, esta sensibilidad se tiene en cuenta al asignar valores a la columna.

MySQL almacena valores SET numéricamente, con el bit de menos peso del valor almacenado correspondiente al primer miembro del conjunto. Si recibe un valor SET en un contexto numérico, el valor recibido tiene los bits asignados correspondientes a los miembros que coinciden con el valor de la columna. Por ejemplo, puede recuperar los valores numéricos de una columna SET así:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

Si se almacena un número en una columna SET, los bits que se asignan en la representación binaria del número determinan los miembros del conjunto en el valor de la columna. Para una columna especificada como SET('a','b','c','d'), los miembros tienen los siguientes valores decimales y binarios:

SET **Miembro**	**Valor decimal**	**Valor binario**
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

Si asigna un valor de 9 a esta columna, esto es 1001 en binario, de forma que el primer y cuarto miembro delSET 'a' y 'd' se seleccionan y el valor resultante es 'a,d'.

Para un valor que contenga más de un elemento SET, no importa el orden en que se listen los elementos cuando inserte el valor. Tampoco no importa cuántas veces se lista un elemento dado para el valor. Cuando el valor se recupera posteriormente, cada elemento en el valor aparece una vez, con los elementos listados según el orden en que se especificaron al crear la tabla. Si una columna se especifica como SET('a','b','c','d'), 'a,d', 'd,a', y 'd,a,a,d,d' aparecen como 'a,d' al recuperarse.

Si asigna un valor no soportado a una columna `SET`, el valor se ignora.

Los valores `SET` se ordenan numéricamente. Los valores `NULL` se ordenan antes de los no `NULL`.

Normalmente, busca valores `SET` usando la función `FIND_IN_SET()` o el operador `LIKE` :

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

El primer comando encuentra registros cuando *`set_col`* contiene el miembro *`value`* del conjunto. El segundo es similar, pero no igual: encuentra registros cuando *`set_col`* contengan el valor *`value`* en cualquier sitio, incluso cuando es una subcadena de otro miembro del conjunto.

Los siguientes comandos también son legales:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

El primero de estos comandos busca valores que contengan el primer miembro del conjunto. El segundo busca una coincidencia exacta. Tenga cuidado con las comparaciones del segundo tipo. Comparar valores del conjunto *`'val1,val2'`* retorna distintos resultados que comparar valores de *`'val2,val1'`*. Debe especificar los valores en el mismo orden en que se listan en la definición de la columna.

Si desea determinar todos los valores posibles para una columna `SET`, use `SHOW COLUMNS FROM` *`tbl_name`* `LIKE` *`set_col`* y parsee la definición de `SET` en la segunda columna de la salida.

11.5. Requisitos de almacenamiento según el tipo de columna

Los requerimientos de almacenamiento para cada uno de los tipos de columnas soportados por MySQL se listan por categoría.

El máximo tamaño de un registro en una tabla `MyISAM` es 65,534 bytes. Cada columna `BLOB` y `TEXT` cuenta sólo de cinco a nueve bytes más alla de su tamaño.

Si una tabla `MyISAM` incluye cualquier tipo de columna de tamaño variable, el formato de rebistro también tiene longitud variable. Cuando se crea una tabla. MySQL puede, bajo ciertas condiciones, cambiar una columna de tamaño variable a fijo o viceversa. Consulte Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas” para más información.

Requerimientos de almacenamiento para tipos numéricos

Tipo de columna	Almacenamiento requerido
`TINYINT`	1 byte
`SMALLINT`	2 bytes
`MEDIUMINT`	3 bytes
`INT`, `INTEGER`	4 bytes
`BIGINT`	8 bytes
`FLOAT(`*`p`*`)`	4 bytes si 0 <= *`p`* <= 24, 8 bytes si 25 <= *`p`* <= 53
`FLOAT`	4 bytes
`DOUBLE [PRECISION]`, objeto `REAL`	8 bytes
`DECIMAL(`*`M`*`,`*`D`*`)`, `NUMERIC(`*`M`*`,`*`D`*`)`	Varía; consulte la siguiente explicación

`BIT(M)`	aproximadamente (*M*+7)/8 bytes

Los requerimientos de almacenamiento para `DECIMAL` (y `NUMERIC`) son específicos para cada versión:

Desde MySQL 5.0.3, los valores para columnas `DECIMAL` más largos se representan usando un formato binario que empaqueta nueve dígitos decimales en cuatro bytes. El almacenamiento para la parte entera y fraccional se determinan separadamente. Cada múltiplo de nueve dígitos requiere cuatro bytes, y el dígito "de resto" requiere alguna fracción de cuatro bytes. El almacenamiento requerido para los dígitos "de resto" se da en la siguiente tabla:

Resto Dítigos	**Número de bytes**
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

Antes de MySQL 5.0.3, las columnas `DECIMAL` se representan como cadenas y el requerimiento de almacenamiento es: *M*+2 bytes si *D* > 0, *M*+1 bytes si *D* = 0 (*D*+2, si *M* < *D*)

Requerimientos de almacenamiento para tipos de fecha y hora

Tipo de columna	Almacenamiento requerido
`DATE`	3 bytes
`DATETIME`	8 bytes
`TIMESTAMP`	4 bytes
`TIME`	3 bytes
`YEAR`	1 byte

Requerimientos de almacenamiento para tipos de cadenas de caracteres

Tipo de columna	Almacenamiento requerido
`CHAR(M)`	*M* bytes, 0 <= *M* <= 255
`VARCHAR(M)`	*L*+1 bytes, donde *L* <= *M* y 0 <= *M* <= 255
`BINARY(M)`	*M* bytes, 0 <= *M* <= 255
`VARBINARY(M)`	*L*+1 bytes, donde *L* <= *M* y 0 <= *M* <= 255
`TINYBLOB`, `TINYTEXT`	*L*+1 byte, donde *L* < 2^8
`BLOB`, `TEXT`	*L*+2 bytes, donde *L* < 2^16
`MEDIUMBLOB`, `MEDIUMTEXT`	*L*+3 bytes, donde *L* < 2^24

LONGBLOB, LONGTEXT	*L*+4 bytes, donde *L* < 2^32
ENUM('*value1*','*value2*',...)	1 o 2 bytes, dependiendo del número de valores de la enumeración (65,535 valores como máximo)
SET('*value1*','*value2*',...)	1, 2, 3, 4, o 8 bytes, dependiendo del número de miembros del conjunto (64 miembros como máximo)

Los tipos VARCHAR y BLOB y TEXT son de longitud variable. Para cada uno, los requerimientos de almacenamiento depende de la longitud de los valores de la (representados por *L* en la tabla precedente), en lugar que el tamaño máximo del tipo. Por ejemplo, una columna VARCHAR(10) puede tratar una cadena con una lengitud máxima de 10. El almacenamiento requerido real es la longitud de la cadena (*L*), más 1 byte para registrar la longitud de la cadena. Para la cadena 'abcd', *L* es 4 y el requerimiento de almacenamiento son 5 bytes.

Para los tipos CHAR, VARCHAR, y TEXT, los valores *L* y *M* en la tabla precedente debe interpretarse como números de caracteres en MySQL 5.0, y las longitudes para estos tipos en las especificaciones de la colmna indican el número de caracteres. Por ejemplo, para almacenar un valor TINYTEXT necesita *L* caracteres + 1 byte.

Desde MySQL 5.0.3, el motor NDBCLUSTER soporta sólo columnas de longitud fija. Esto significa que una columna VARCHAR de una tabla en MySQL Cluster se comportará casi como si fuera de tipo CHAR (excepto que cada registro todavía tiene un byte extra). Por ejemplo, en una tabla Cluster, *cada* registro en una columna declarada como VARCHAR(100) necesitará 101 bytes para almacenamiento, sin tener en cuenta la longitud de la columna almacenada en cualquier registro.

Los tipos BLOB y TEXT requieren 1, 2, 3, o 4 bytes para almacenar la longitud de la columna, dependiendo de la longitud máxima posible del tipo. Consulte Sección 11.4.3, “Los tipos BLOB y TEXT”.

Las columnas TEXT y BLOB se implementan de forma distinta en el motor de almacenamiento NDB Cluster , donde cada registro en una columna TEXT se compone de dos partes separadas. Una de estas es de longitud fija (256 bytes), y se almacena realmente en la tabla original. La otra consiste de cualquier dato de más de 256 bytes, que se almacena en una tabla oculta. Los registros en esta segunda tabla siempre tienen una longitud de 2,000 bytes . Esto significa que el tamaño de una columna TEXT es 256 si *size* <= 256 (donde *size* representa el tamaño del registro); de otro modo, el tamaño es 256 + *size* + (2000 - (*size* - 256) % 2000).

El tamaño de un objeto ENUM lo determina el número de diferentes valores de la enumeración. Un byte se usa para enumeraciones de hasta 255 valores posibles. Dos bytes se usan para enumeraciones teniendo entre 256 y 65,535 valores posibles. Consulte Sección 11.4.4, “El tipo de columna ENUM”.

El tamaño de un objeto SET se determina por el número de diferentes mienbros del conjunto. Si el tamaño del conjunto es *N*, el objeto ocupa (*N*+7)/8 bytes, redondeado a 1, 2, 3, 4, o 8 bytes. Un SET puede tener como máximo 64 miembros. Consulte Sección 11.4.5, “El tipo SET”.

11.6. Escoger el tipo de columna correcto

Para almacenamiento óptimo, debe tratar de usar el tipo más preciso en todos los casos. Por ejemplo, si una columna entera se usa para valores en el rango de 1 a 99999, entonces MEDIUMINT UNSIGNED es el mejor tipo. De los tipos que representan todos los valores requeridos, este tipo usa la menor cantidad de espacio.

Las tablas creadas en MySQL 5.0.3 y versiones posteriores usan un nuevo formato de almacenamiento para columnas DECIMAL . Todos los cálculos básicos (+,-,*,/) con columnas DECIMAL se realizan con precisión de 64 dígitos decimales. Consulte Sección 11.1.1, “Panorámica de tipos numéricos”.

Los cálculos con valores `DECIMAL` se realizan usando operaciones de doble precisioón. Si la precisión no es muy importante, o si la velocidad es la máxima prioridad, el tipo `DOUBLE` puede ser lo bastante bueno. Para alta precisión, siempre puede convertirlo a un tipo de punto fijo como `BIGINT`. Esto le permite hacer todos los cálculos con enteros de 64-bit, luego convertir los resultados de nuevo a valores de coma flotante.

11.7. Usar tipos de columnas de otros motores de bases de datos

Para facilitar el uso de código escrito por implementadores de SQL de otros vendedores, MySQL mapea los tipos de columnas como se muestra en la siguiente tabla. Estos mapeos hacen más fácil importar las definiciones de tablas de otros motores de bases de datos a MySQL:

Tipos de otros vendedores	Tipos MySQL
`BOOL`,	`TINYINT`
`BOOLEAN`	`TINYINT`
`CHAR VARYING(M)`	`VARCHAR(M)`
`DEC`	`DECIMAL`
`FIXED`	`DECIMAL`
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`
`INT3`	`MEDIUMINT`
`INT4`	`INT`
`INT8`	`BIGINT`
`LONG VARBINARY`	`MEDIUMBLOB`
`LONG VARCHAR`	`MEDIUMTEXT`
`LONG`	`MEDIUMTEXT`
`MIDDLEINT`	`MEDIUMINT`
`NUMERIC`	`DECIMAL`

El mapeo de tipos de columnas se realiza en tiempo de creación de la tabla, tras el cual se descartan las especificaciones originales de tipos. Si crea una tabla con tipos usados por otros vendedores y luego realiza un comando `DESCRIBE tbl_name`, MySQL muestra la estructura de tabla usando los tipos MySQL equivalentes. Por ejemplo:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG, d NUMERIC);
Query OK, 0 rows affected (0.08 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 (0.00 sec)
```

Capítulo 12. Funciones y operadores

Tabla de contenidos

Las expresiones pueden usarse en varios puntos de los comandos SQL, tales como en las cláusulas `ORDER BY` o `HAVING` de los comandos `SELECT` , en la cláusula `WHERE` de los comandos `SELECT`, `DELETE`, o `UPDATE` o en comandos, `SET` . Las expresiones pueden escribirse usando valores literales, valores de columnas, `NULL`, funciones y operadores. Este capítulo describe las funciones y operadores permitidos para escribir expresiones en MySQL.

Una expresión que contiene `NULL` siempre produce un valor `NULL` a no ser que se indique de otro modo en la documentación para una función u operador particular.

Nota: Por defecto, no deben haber espacios en blanco entre un nombre de función y los paréntesis que lo siguen. Esto ayuda al parser de MySQL a distinguir entre llamadas a funciones y referencias a tablas o columnas que tengan el mismo nombre que una función. Sin embargo, se permiten espacios antre los argumentos de las funciones.

Puede decirle a MySQL server que acepte espacios tras los nombres de funciones arrancando con la opción `--sql-mode=IGNORE_SPACE`. Los programas cliente pueden pedir este comportamiento

usando la opción `CLIENT_IGNORE_SPACE` para `mysql_real_connect()`. En cualquier caso, todos los nombres de función son palabras reservadas. Consulte Sección 5.3.2, “El modo SQL del servidor”.

Para una mayor brevedad, la mayoría de ejemplos de este capítulo muestran la salida del programa `mysql` de forma abreviada. En lugar de mostrar ejemplos en este formato:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

Se usa este otro:

```
mysql> SELECT MOD(29,9);
        -> 2
```

12.1. Operadores

12.1.1. Precedencias de los operadores

La precedencia de operadores se muestra en la siguiente lista, de menor a mayor precedencia. Los operadores que se muestran juntos en una línea tienen la misma precedencia.

```
:=
||, OR, XOR
&&, AND
NOT
BETWEEN, CASE, WHEN, THEN, ELSE
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
<<, >>
-, +
*, /, DIV, %, MOD
^
- (resta unaria), ~ (inversión de bit unaria)
!
BINARY, COLLATE
```

La precedencia mostrada para `NOT` es desde MySQL 5.0.2. En versiones anteriores, o desde 5.0.2 si el modo `HIGH_NOT_PRECEDENCE` está activo, la precedencia de `NOT` es la misma que la del operador `!` . Consulte Sección 5.3.2, “El modo SQL del servidor”.

12.1.2. Paréntesis

- `( ... )`

  Use paréntesis para forzar el orden de evaluación en una expresión. Por ejemplo:

  ```
  mysql> SELECT 1+2*3;
          -> 7
  mysql> SELECT (1+2)*3;
          -> 9
  ```

12.1.3. Funciones y operadores de comparación

Las operaciones de comparación dan un valor de `1` (CIERTO), `0` (FALSO), o `NULL`. Estas operaciones funcionan tanto para números como para cadenas de caracteres. Las cadenas de caracteres se convierten automáticamente en números y los números en cadenas cuando es necesario.

Algunas de las funciones de esta sección (tales como `LEAST()` y `GREATEST()`) retornan valores distintos a `1` (CIERTO), `0` (FALSO), o `NULL`. Sin embargo, el valor que retornan se basa en operaciones de comparación realizadas como describen las siguientes reglas.

MySQL compara valores usando las siguientes reglas:

- Si uno o ambos argumentos son `NULL`, el resultado de la comparación es `NULL`, excepto para el operador de comparación `NULL`-safe `<=>` .
- Si ambos argumentos en una operación de comparación son cadenas, se comparan como cadenas.
- Si ambos argumentos son enteros, se comparan como enteros.
- Los valores hexadecimales se tratan como cadenas binarias si no se comparan con un número.
-
- En todos los otros casos, los argumentos se comparan como números con punto flotante (reales).

Por defecto, la comparación de cadenas no es sensible a mayúsculas y usa el conjunto de caracteres actual (ISO-8859-1 Latin1 por defecto, que siempre funciona bien para inglés).

Par convertir un valor a un tipo específico para una comparación, puede usar la función `CAST()` . Los valores de cadenas de caracteres pueden convertirse a un conjunto de caracteres distinto usando `CONVERT()`. Consulte Sección 12.8, “Funciones y operadores de cast”.

Los siguientes ejemplos ilustran conversión de cadenas a números para operaciones de comparación:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

Tenga en cuanta que cuando compara una columna de cadenas de caracteres con un número, MySQL no puede usar el índice de la columna para buscar rápidamente le valor. Si *`str_col`* es una columna de cadenas indexada, el índice no puede usarse al realizar la búsqueda en el siguiente comando:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

La razón es que hay diferentes cadenas que pueden convertirse al valor `1`: `'1'`, `' 1'`, `'1a'`, ...

- `=`

  Igual:

  ```
  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
```

- `<=>`

  `NULL`-safe equal. Este operador realiza una comparación de igualdad como el operador `=`, pero retorna 1 en lugar de `NULL` si ambos operandos son `NULL`, y 0 en lugar de `NULL` si un operando es `NULL`.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
        -> 1, NULL, NULL
```

- `<>`, `!=`

  Diferente:

```
mysql> SELECT '.01' <> '0.01';
        -> 1
mysql> SELECT .01 <> '0.01';
        -> 0
mysql> SELECT 'zapp' <> 'zappp';
        -> 1
```

- `<=`

  Menor que o igual:

```
mysql> SELECT 0.1 <= 2;
        -> 1
```

- `<`

  Menor que:

```
mysql> SELECT 2 < 2;
        -> 0
```

- `>=`

  Mayor que o igual:

```
mysql> SELECT 2 >= 2;
        -> 1
```

- `>`

Mayor que:

```
mysql> SELECT 2 > 2;
        -> 0
```

- `IS valor booleano`, `IS NOT valor booleano`

  Comprueba si un valor contra un valor booleano, donde *`boolean_value`* puede ser `TRUE`, `FALSE`, o `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
        -> 1, 1, 1
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
        -> 1, 1, 0
```

  `IS [NOT] valor booleano` sintaxis se añadió en MySQL 5.0.2.

- `IS NULL`, `IS NOT NULL`

  Testea si un valor es o no `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1, 1, 0
```

  Para poder trabajar con programas ODBC, MySQL soporta las siguientes características extra al usar `IS NULL`:

  - Puede encontrar el registro que contiene el valor `AUTO_INCREMENT` más reciente realizando un comando de la siguiente forma inmediatamente tras generar el valor:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

    Este comportamiento puede desactivarse asignando `SQL_AUTO_IS_NULL=0`. Consulte Sección 13.5.3, “Sintaxis de SET”.

  - Para columnas `DATE` y `DATETIME` que se declaran como `NOT NULL`, puede encontrar la fecha especial `'0000-00-00'` con un comando como este:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

    Esto es necesario para algunas aplicaciones ODBC, ya que ODBC no soporta un valor de fecha `'0000-00-00'` .

- *`expr`* `BETWEEN` *`min`* `AND` *`max`*

  Si *`expr`* es mayor o igual que *`min`* y *`expr`* es menor o igual a *`max`*, `BETWEEN` retorna `1`, de otro modo retorna `0`. Esto es equivalente a la expresión `(`*`min`* `<=` *`expr`* `AND` *`expr`* `<=` *`max`*`)` si todos los

argumentos son del mismo tipo. De otro modo la conversión de tipos tiene lugar según las reglas descritas al principio de la sección, pero aplicadas a todos los argumentos.

```
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
```

- *`expr`* `NOT BETWEEN` *`min`* `AND` *`max`*

  Esto es lo mismo que `NOT (`*`expr`* `BETWEEN` *`min`* `AND` *`max`*`)`.

- `COALESCE(`*`value`*`,...)`

  Retorna el primer valore no `NULL` de la lista.

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

- `GREATEST(`*`value1`*`,`*`value2`*`,...)`

  Con dos o más argumentos, retorna el argumento mayor (con valor mayor). Los argumentos se comparan usando las mismas reglas que para `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'
```

- *`expr`* `IN (`*`value`*`,...)`

  Retorna `1` si *`expr`* es uno de los valores en la lista `IN` , de lo contrario retorna `0`. Si todos los valores son constantes, se evalúan según el tipo y ordenación de *`expr`* . La búsqueda para el elemento se hace usando búsqueda binaria. Esto significa que `IN` es muy rápido si la lista `IN` consiste enteramente en constantes. Si *`expr`* es una expresión de cadenas de caracteres sensible a mayúsculas, la comparación de cadenas se realiza sensible a mayúsculas.

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
        -> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
        -> 1
```

El número de valores en la lista `IN` sólo está limitado por el valor `max_allowed_packet`.

En MySQL 5.0, para cumplir el estándar SQL `IN` retorna `NULL` no sólo si la expresión de la parte izquierda es `NULL`, también si no encuentra coincidencias en la lista y una de las expresiones en la lista es `NULL`.

La sintaxis de `IN()` puede usarse para escribir algunas subconsultas. Consulte Sección 13.2.8.3, “Subconsultas con ANY, IN y SOME”.

- *`expr`* `NOT IN (`*`value`*`,...)`

  Esto es lo mismo que `NOT (`*`expr`* `IN (`*`value`*`,...))`.

- `ISNULL(`*`expr`*`)`

  Si *`expr`* es `NULL`, `ISNULL()` retorna `1`, sino retorna `0`.

```
mysql> SELECT ISNULL(1+1);
        -> 0
mysql> SELECT ISNULL(1/0);
        -> 1
```

  Una comparación de valores `NULL` usando `=` siempre es falsa.

  La función `ISNULL()` comparte algunos comportamientos especiales con el operador de comparación `IS NULL`, consulte la descripción de `IS NULL` en Sección 12.1.3, “Funciones y operadores de comparación”.

- `INTERVAL(`*`N`*`,`*`N1`*`,`*`N2`*`,`*`N3`*`,...)`

  Retorna `0` if *`N`* < *`N1`*, `1` si *`N`* < *`N2`* y así o `-1` si *`N`* es `NULL`. Todos los argumentos se tratan como enteros. Esto requiere que *`N1`* < *`N2`* < *`N3`* < `...` < *`Nn`* para que la función funcione correctamente. Esto es porque se usa una búsqueda binaria (muy rápida).

```
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`*`,...)`

  Con dos o más argumentos, retorna el argumento menor (con un valor menor). Los argumentos se comparan usando las siguientes reglas:

  - Si el valor retornado se usan en un contexto `INTEGER` o todos los argumentos son enteros, se comparan como enteros.

- Si el valor retornado se usa en un contexto `REAL` o todos los argumentos son reales, se comparan como reales.
- Si algún argumento es una cadena de caracteres sensible a mayúsculas, los argumentos se comparan como cadenas sensibles a mayúsculas.
- En cualquier otro caso, los argumentos se comparan como cadenas no sensibles a mayúsculas.

```
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'
```

Tenga en cuenta que las reglas de conversión precedentes pueden producir resultados extraños en algunos casos extremos:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
    -> -9223372036854775808
```

Esto ocurre porque MySQL lee `9223372036854775808.0` en un contexto entero. La representación entera no es lo bastante buena para tratar el valor, así que lo cambia a entero con signo.

12.1.4. Operadores lógicos

En SQL, todos los operadores lógicos se evalúan a `TRUE`, `FALSE`, o `NULL` (`UNKNOWN`). En MySQL, se implementan como 1 (`TRUE`), 0 (`FALSE`), y `NULL`. La mayoría de esto es común en diferentes servidores de bases de datos SQL aunque algunos servidores pueden retornar cualquier valor distinto a cero para `TRUE`.

- `NOT`, `!`

  NOT lógica. Se evalúa a `1` si el operando es `0`, a `0` si el operando es diferente a cero, y `NOT NULL` retorna `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
```

  El último ejemplo produce `1` porque la expresión se evalúa igual que `(!1)+1`.

- `AND`, `&&`

  AND lógica. Se evalúa a `1` si todos los operandos son distintos a cero y no `NULL`, a 0 si uno o más operandos son `0`, de otro modo retorna `NULL`.

```
mysql> SELECT 1 && 1;
        -> 1
mysql> SELECT 1 && 0;
        -> 0
mysql> SELECT 1 && NULL;
        -> NULL
mysql> SELECT 0 && NULL;
        -> 0
mysql> SELECT NULL && 0;
        -> 0
```

- `OR`, `||`

  OR lógica. Cuando ambos ooperandos son no `NULL`, el resultado es 1 si algún operando es diferente a cero, y 0 de otro modo. Con un operando `NULL` el resultado es 1 si el otro operando no es cero, y `NULL` de otro modo. Si ambos operandos son `NULL`, el resultado es `NULL`.

```
mysql> SELECT 1 || 1;
        -> 1
mysql> SELECT 1 || 0;
        -> 1
mysql> SELECT 0 || 0;
        -> 0
mysql> SELECT 0 || NULL;
        -> NULL
mysql> SELECT 1 || NULL;
        -> 1
```

- `XOR`

  XOR lógica. Retorna `NULL` si algún operando es `NULL`. Para operandos no `NULL` , evalúa a 1 si un número impar de operandos es distinto a cero, sino retorna 0.

```
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` es matemáticamente igual a `(a AND (NOT b)) OR ((NOT a) and b)`.

12.2. Funciones de control de flujo

- `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`

  La primera versión retorna *`result`* donde *`value=compare-value`*. La segunda versión retorna el resultado para la primera condición que es cierta. Si no hay ningún resultado coincidente, el resultado tras `ELSE` se retorna, o `NULL` si no hay parte `ELSE`.

```
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;
        -> NULL
```

El tipo de retorno por defecto de una expresión `CASE` es el tipo agregado compatible de todos los valores de retorno, pero también depende del contexto en el que se usa. Si se usa en un entorno de cadenas de caracteres, el resultado se retorna como cadena de caracteres. Si se usa en un contexto numérico, el resultado se retorna como valor decimal, real o entero.

- `IF(expr1,expr2,expr3)`

  Si *`expr1`* es TRUE (*`expr1 <> 0`* and *`expr1 <> NULL`*) entonces `IF()` retorna *`expr2`*; de otro modo retorna *`expr3`*. `IF()` retorna un valor numérico o cadena de caracteres, en función del contexto en que se usa.

```
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'
```

  Si sólo una de `expr2` o `expr3` es explícitamente `NULL`, el tipo del resultado de la función `IF()` es el mismo tipo que la expresión no `NULL` .

  `expr1` se evalúa como un valor entero, que significa que si esta testeando valores de punto flotante o cadenas de caracteres, debe hacerlo mediante operaciones de comparación.

```
mysql> SELECT IF(0.1,1,0);
        -> 0
mysql> SELECT IF(0.1<>0,1,0);
        -> 1
```

  En el primer caso mostrado, `IF(0.1)` retorna 0 ya que 0.1 se convierte a un valor entero, resultando en un test de `IF(0)`. Puede que esto no ea lo que espera. En el segundo caso, la comparación testea el valor de coma flotante para comprobar que no es cero. El resultado de la comparación se usa como entero.

  El tipo de retorno de `IF()` (que puede ocurrir cuando se almacena en una tabla temporal) se calcula como sigue:

Expresión	Valor Retornado
`expr2` o `expr3` retorna una cadena	cadena de caracteres
`expr2` o `expr3` retorna un valor de coma flotante	coma flotante
`expr2` o `expr3` retorna un entero	entero

  Si `expr2` y `expr3` son cadenas de caracteres, el resultado es sensible a mayúsculas si alguna de las cadenas lo es.

- `IFNULL(expr1,expr2)`

  Si `expr1` no es `NULL`, `IFNULL()` retorna `expr1`, de otro modo retorna `expr2`. `IFNULL()` retorna un valor numérico o de cadena de caracteres, en función del contexto en que se usa.

```
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'
```

  El valor por defecto de retorno de `IFNULL(expr1,expr2)` es el más “general” de las dos expresiones, en el orden `STRING`, `REAL`, o `INTEGER`. Considere el caso de una tabla basada en expresiones o donde MySQL debe almacenar internamente un valor retornado por `IFNULL()` en una tabla temporal:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| test  | varbinary(4) | NO   |     |         |       |
+-------+--------------+------+-----+---------+-------+
```

  En este ejemplo, el tipo de la columna`test` es `VARBINARY(4)`.

- `NULLIF(expr1,expr2)`

  Retorna `NULL` si `expr1 = expr2` es cierto, de otro modo retorna `expr1`. Es lo mismo que `CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END`.

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
        -> 1
```

  Tenga en cuenta que MySQL evalúa `expr1` dos veces si los argumentos no son iguales.

12.3. Funciones para cadenas de caracteres

Las funciones de cadenas de caracteres retornan `NULL` si la longitud del resultado es mayor que el valor de la variable de sistema `max_allowed_packet` . Consulte Sección 7.5.2, “Afinar parámetros del servidor”.

Para funciones que operan en posiciones de cadenas de caracteres, la primera posición es la 1.

- `ASCII(str)`

  Retorna el valor numérico del carácter más a la izquierda de la cadena de caracteres `str`. Retorna 0 si `str` es la cadena vacía. Retorna `NULL` si `str` es `NULL`. `ASCII()` funciona para caracteres con valores numéricos de 0 a 255.

```
mysql> SELECT ASCII('2');
```

```
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

Consulte la función `ORD()` .

- `BIN(N)`

  Retorna una representación de cadena de caracteres del valor binario de *`N`*, donde *`N`* es un número muy largo (`BIGINT`) . Esto es equivalente a `CONV(N,10,2)`. Retorna `NULL` si *`N`* es `NULL`.

```
mysql> SELECT BIN(12);
        -> '1100'
```

- `BIT_LENGTH(str)`

  Retorna la longitud de la cadena de caracteres *`str`* en bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

- `CHAR(N,...)`

  `CHAR()` interpreta los argumentos como enteros y retorna la cadena de caracteres que consiste en los caracteres dados por los códigos de tales enteros. Los valores `NULL` no se tienen en cuenta.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

- `CHAR_LENGTH(str)`

  Retorna la longitud de la cadena de caracteres *`str`*, medida en caracteres. Un carácter de múltiples bytes cuenta como un sólo carácter. Esto significa que para una cadena de caracteres que contiene cinco caracteres de dos bytes, `LENGTH()` retorna `10`, mientras `CHAR_LENGTH()` returna `5`.

- `CHARACTER_LENGTH(str)`

  `CHARACTER_LENGTH()` es sinónimo de `CHAR_LENGTH()`.

- `COMPRESS(string_to_compress)`

Comprime una cadena de caracteres. Esta función necesita que MySQL se compile con una biblioteca de compresión como `zlib`. De otro modo, el valor retornado siempre es `NULL`. La cadena comprimida puede descomprimirse con `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
```

Los contenidos comprimidos se almacenan de la siguiente forma:

- Cadenas vacías se almacenan como cadenas vacías.
- Cadenas no vacías se almacenan como longitud de cuatros bytes de la cadena descomprimida (los bytes más bajos primero), seguido de la cadena comprimida. Si la cadena acaba con un espacio, se añade un carácter '`.`' para evitar problemas con eliminación de espacios finales al almacenar en una columna `CHAR` o `VARCHAR` . (El uso de `CHAR` o `VARCHAR` para almacenar cadenas comprimidas no se recomienda. Es mejor usar una columna `BLOB` .)

- `CONCAT(`*`str1`*`,`*`str2`*`,...)`

  Retorna la cadena resultado de concatenar los argumentos. Retorna `NULL` si algúna argumento es `NULL`. Puede tener uno o más argumentos. Si todos los argumentos son cadenas no binarias, el resultado es una cadena no binaria. Si los argumentos incluyen cualquier cadena binaria, el resultado es una cadena binaria. Un argumento numérico se convierte a su forma de cadena binaria equivalente; si quiere evitarlo puede usar conversión de tipos explícita, como en este ejemplo: `SELECT CONCAT(CAST(int_col AS CHAR), char_col)`

  ```
  mysql> SELECT CONCAT('My', 'S', 'QL');
          -> 'MySQL'
  mysql> SELECT CONCAT('My', NULL, 'QL');
          -> NULL
  mysql> SELECT CONCAT(14.3);
          -> '14.3'
  ```

- `CONCAT_WS(`*`separator`*`,`*`str1`*`,`*`str2`*`,...)`

  `CONCAT_WS()` significa CONCAT With Separator (CONCAT con separador) y es una forma especial de `CONCAT()`. El primer argumento es el separador para el resto de argumentos. El separador se añade entre las cadenas a concatenar. El separador puede ser una cadena como el resto de argumentos. Si el separador es `NULL`, el resultado es `NULL`. La función evita valores `NULL` tras el argumento separador.

  ```
  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'
  ```

  En MySQL 5.0, `CONCAT_WS()` no evita cadenas vacías. (Sin embargo, evita `NULL`s.)

- `CONV(N,from_base,to_base)`

  Convierte números entre diferentes bases numéricas. Retorna una representación de cadena de carácters para el número `N`, convertido de base `from_base` a base `to_base`. Retorna `NULL` si algún argumento es `NULL`. El argumento `N` se interpreta como entero, pero puede especificarse como un entero o cadena. La base mínima es `2` y la máxima es `36`. Su `to_base` es un número negativo, `N` se trata como un número con signo. De otro modo, `N` se trata como sin signo. `CONV()` funciona con precisión de 64-bit.

```
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'+0xa,10,10);
        -> '40'
```

- `ELT(N,str1,str2,str3,...)`

  Retorna `str1` si `N` = `1`, `str2` if `N` = `2`, y así. Retorna `NULL` si `N` es menor que `1` o mayor que el número de argumentos. `ELT()` es el complemento de `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]])`

  Retorna una cadena en que para cada bit del valor `bits`, puede obtener una cadena `on` y para cada bit reasignado obtiene una cadena `off` . Los bits en `bits` se examinan de derecha a izquierda (de bits menores a mayores). Las cadenas se añaden al resultado de izquierda a derecha, separados por la cadena `separator` (siendo el carácter por defecto la coma '`,`'). El número de bits examinados se obtiene por `number_of_bits` (por defecto 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,...)`

  Retorna el índice de `str` en la lista `str1`, `str2`, `str3`, `...` . Retorna `0` si no se encuentra `str`.

  Si todos los argumentos de `FIELD()` son cadenas, todos los argumentos se comparan como cadenas. Si todos los argumentos son números, se comparan como números. De otro modo, los argumentos se comparan como números con doble precisión.

Si *str* es NULL, el valor retornado es 0 porque NULL falla en comparaciones de comparación con cualquier valor. FIELD() es el complemento de 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*)

  Retorna un valor en el rango de 1 a *N* si la cadena *str* está en la lista de cadenas *strlist* consistente de *N* subcadenas. Una lista de cadenas es una cadena compuesta de subcadenas separadas por caracteres ',' . Si el primer argumento es una cadena constante y el segundo es una columna de tipo SET, la función FIND_IN_SET() está optimizada para usar aritmética de bit. Retorna 0 si *str* no está en *strlist* o si *strlist* es la cadena vacía. Retorna NULL si algún argumento es NULL. Esta función no funciona apropiadamente si el primer argumento contiene un carácter de coma (',') .

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

- HEX(*N_or_S*)

  Si *N_OR_S* es un número, retorna una cadena representación del valor hexadecimal de *N*, donde *N* es un número muy grande (BIGINT) . Esto es equivalente a CONV(N,10,16).

  Si *N_OR_S* es una cadena, esta función retorna una cadena hexadecimal de *N_OR_S* caracteres, donde cada carácter en *N_OR_S* se convierte a dos dígitos hexadecimales.

```
mysql> SELECT HEX(255);
        -> 'FF'
mysql> SELECT 0x616263;
        -> 'abc'
mysql> SELECT HEX('abc');
        -> 616263
```

- INSERT(*str*,*pos*,*len*,*newstr*)

  Retorna la cadena *str*, con la subcadena comenzando en la posición *pos* y *len* caracteres reemplazados por la cadena *newstr*. Retorna la cadena original si *pos* no está entre la longitud de la cadena. Replaza el resto de la cadena a partir de la posición *pos* si *len* no está dentro de la longitud del resto de la cadena. Retorna NULL si cualquier argumento es nulo.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
        -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
        -> 'QuWhat'
```

  Esta función está preparada para funcionar con múltiples bytes.

- `INSTR(str,substr)`

  Retorna la posición de la primera ocurrencia de la subcadena *`substr`* en la cadena *`str`*. Es lo mismo que la forma de dos argumentos de `LOCATE()`, excepto que el orden de los argumentos es inverso.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

  Esta función puede trabajar con múltiples bytes. En MySQL 5.0, sólo es sensible a mayúsculas si uno de los argumentos es una cadena binaria.

- `LCASE(str)`

  `LCASE()` es sinónimo de `LOWER()`.

- `LEFT(str,len)`

  Retorna los *`len`* caracteres empezando por la izquierda de la cadena *`str`*.

```
mysql> SELECT LEFT('foobarbar', 5);
        -> 'fooba'
```

- `LENGTH(str)`

  Retorna la longitud de la cadena *`str`*, medida en bytes.Un carácter multi-byte cuenta como múltiples bytes. Esto significa que para cadenas que contengan cinco caracteres de dos bytes, `LENGTH()` retorna 10, mientras que `CHAR_LENGTH()` retorna 5.

```
mysql> SELECT LENGTH('text');
        -> 4
```

- `LOAD_FILE(file_name)`

  Lee el fichero y retorna el contenido como cadena de caracteres. El fichero debe estar localizado en el servidor, debe especificar la ruta completa al fichero, y debe tener el privilegio `FILE` . El fichero debe ser legible por todo el mundo y su tamaño menor a `max_allowed_packet` bytes.

  Si el fichero no existe o no puede ser leído debido a que una de las anteriores condiciones no se cumple, la función retorna `NULL`.

```
mysql> UPDATE tbl_name
          SET blob_column=LOAD_FILE('/tmp/picture')
          WHERE id=1;
```

- `LOCATE(substr,str)` , `LOCATE(substr,str,pos)`

  La primera sintaxis retorna la posición de la primera ocurrencia de la subcadena *`substr`* en la cadena *`str`*. La segunda sintaxis retorna la posición de la primera ocurrencia de la subcadena *`substr`* en la cadena *`str`*, comanzando en la posición *`pos`*. Retorna 0 si *`substr`* no está en *`str`*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar',5);
        -> 7
```

  Esta función trabaja con múltiples bytes. En MySQL 5.0, es sensible a mayúsculas sólo si algún argumento es una cadena binaria.

- `LOWER(str)`

  Retorna la cadena *`str`* con todos los caracteres cambiados a minúsculas según el mapeo del conjunto de caracteres actual (por defecto es ISO-8859-1 Latin1).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

  Esta función funciona con múltiples bytes.

- `LPAD(str,len,padstr)`

  Retorna la cadena *`str`*, alineado a la izquierda con la cadena *`padstr`* a una longitud de *`len`* caracteres. Si *`str`* es mayor que *`len`*, el valor retornado se acorta a *`len`* caracteres.

```
mysql> SELECT LPAD('hi',4,'??');
        -> '??hi'
mysql> SELECT LPAD('hi',1,'??');
        -> 'h'
```

- `LTRIM(str)`

  Retorna la cadena *`str`* con los caracteres en blanco iniciales eliminados.

```
mysql> SELECT LTRIM('  barbar');
        -> 'barbar'
```

  Esta función trabaja con múltiples bytes.

- `MAKE_SET(bits,str1,str2,...)`

Retorna un conjunto de valores (una cadena conteniendo subcadenas separadas por caracteres ',') consistiendo en cadenas que tienen el bit correspondiente en *bits* asignado. *str1* se corresponde al bit 0, *str2* al bit 1, y así. Los valores NULL en *str1*, *str2*, ... no se añaden al resultado.

```
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*) es sinónimo de SUBSTRING(*str*,*pos*,*len*).

- OCT(*N*)

  Retorna una representación en cadena del valor octal de *N*, donde *N* es un número largo (BIGINT). Es equivalente a CONV(N,10,8). Retorna NULL si *N* es NULL.

```
mysql> SELECT OCT(12);
        -> '14'
```

- OCTET_LENGTH(*str*)

  OCTET_LENGTH() es sinónimo de LENGTH().

- ORD(*str*)

  Si el carácter más a la izquierda de la cadena *str* es un carácter multi-byte , retorna el código de ese carácter, calculado a partir del valor numérico de sus bytes usando esta fórmula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  Si el caráctar más a la izquierda no es multi-byte , ORD() retorna el mismo valor que la función ASCII() .

```
mysql> SELECT ORD('2');
        -> 50
```

- POSITION(*substr* IN *str*)

  POSITION(*substr* IN *str*) es sinónimo de LOCATE(*substr*,*str*).

- `QUOTE(str)`

  Acota una cadena para producir un resultado que puede usarse como un valor con caracteres de escape en un comando SQL. La cadena se retorna rodeado por comillas sencillas y con cada instancia de comilla sencilla (`''`), antibarra (`'\'`), ASCII NUL, y Control-Z predecidos por una antibarra. Si el argumento es `NULL`, el valor de retorno es la palabra “NULL” sin comillas alrededor.

```
mysql> SELECT QUOTE('Don\'t!');
        -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
        -> NULL
```

- `REPEAT(str,count)`

  Retorna una cadena consistente de la cadena *`str`* repetida *`count`* veces. Si *`count`* `<= 0`, retorna una cadena vacía. Retorna `NULL` si *`str`* o *`count`* son `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)`

  Retorna la cadena *`str`* con todas las ocurrencias de la cadena *`from_str`* reemplazadas con la cadena *`to_str`*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

  Esta función trabaja con múltiples bytes.

- `REVERSE(str)`

  Retorna la cadena *`str`* con el orden de los caracteres invertido.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

  Esta función trabaja con múltiples bytes.

- `RIGHT(str,len)`

  Retorna los *`len`* caracteres de la derecha de la cadena *`str`*.

```
mysql> SELECT RIGHT('foobarbar', 4);
        -> 'rbar'
```

Esta función trabaja con múltiples bytes.

- `RPAD(str,len,padstr)`

  Retorna la cadena *`str`*, alineada a la derecha con la cadena *`padstr`* con una longitud de *`len`* caracteres. Si *`str`* es mayor que *`len`*, el valor de retorno se corta a *`len`* caracteres.

```
mysql> SELECT RPAD('hi',5,'?');
        -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

  Esta función trabaja con múltiples bytes.

- `RTRIM(str)`

  Retorna la cadena *`str`* con los espacios precedentes eliminados.

```
mysql> SELECT RTRIM('barbar   ');
        -> 'barbar'
```

  Esta función trabaja con múltiples bytes.

- `SOUNDEX(str)`

  Retorna una cadena soudex de *`str`*. Dos cadenas que suenen igual deben tener cadenas soundex idénticas. Una cadena soundex estándar tiene cuatro caracteres de longitud, pero la función `SOUNDEX()` retorna una cadena arbitráriamente larga. Puede usar `SUBSTRING()` en el resultado para obtener una cadena soundex estándar. Todos los caracteres no alfabéticos en *`str`* se ignoran. Todos los caracteres alfabéticos internacionales fuera del rango A-Z se tratan como vocales.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

  Nota: Esta función implementa el algoritmo Soundex original, no la versión más popular (descrita por D. Hnuth). La diferencia es que la versión original descarta vocales primero y luego duplicados, mientras que la versión mejorada descarta primero los duplicados y luego las vocales.

- *`expr1`* `SOUNDS LIKE` *`expr2`*

  Es lo mismo que `SOUNDEX(`*`expr1`*`) = SOUNDEX(`*`expr2`*`)`.

- `SPACE(N)`

  Retorna la cadena consistente en *`N`* caracteres blancos.

```
mysql> SELECT SPACE(6);
        -> '      '
```

- `SUBSTRING(str,pos)` , `SUBSTRING(str FROM pos)`, `SUBSTRING(str,pos,len)` , `SUBSTRING(str FROM pos FOR len)`

  Las formas sin el arguemnto *`len`* retornan una subcadena de la cadena *`str`* comenzando en la posición *`pos`*. Las formas con el argumento *`len`* retornan una subcadena de longitud *`len`* a partir de la cadena *`str`*, comenzando en la posición *`pos`*.Las formas que usan `FROM` son sintaxis SQL estándard. En MySQL 5.0, es posible usar valores negativos para *`pos`*. En este caso, el inicio de la subcadena son *`pos`* caracteres a partir del final de la cadena, en lugar del principio. Un valor negativo puede usarse para *`pos`* en cualquier de las formas de esta función.

  ```
  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'
  ```

  Esta función trabaja con múltiples bytes.

  Tenga en cuenta que si usa un valor menor a 1 para *`len`*, el resultado siempre es una cadena vacía.

  `SUBSTR()` es sinónimo de `SUBSTRING()`.

- `SUBSTRING_INDEX(str,delim,count)`

  Retorna la subcadena de la cadena *`str`* antes de *`count`* ocurrencias del delimitador *`delim`*. Si *`count`* es positivo, todo a la izquierda del delimitador final (contando desde la izquierda) se retorna. Si *`count`* es negativo, todo a la derecha del delimitador final (contando desde la derecha) se retorna.

  ```
  mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
          -> 'www.mysql'
  mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
          -> 'mysql.com'
  ```

  Esta función trabaja con múltiples bytes.

- `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)`, `TRIM(remstr FROM] str)`

  Retorna la cadena *`str`* con todos los prefijos y/o sufijos *`remstr`* eliminados. Si ninguno de los especificadores `BOTH`, `LEADING`, o se da`TRAILING`, `BOTH` se asumen. Si *`remstr`* es opcional y no se especifica, los espacios se eliminan.

```
mysql> SELECT TRIM('  bar   ');
        -> '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'
```

Esta función trabaja con múltiples bytes.

- `UCASE(str)`

  `UCASE()` es sinónimo de `UPPER()`.

- `UNCOMPRESS(string_to_uncompress)`

  Descomprime una cadena comprimida con la función `COMPRESS()` . Si el argumento no es un valor comprimido, el resultado es `NULL`. Esta función necesita que MySQL se compile con una biblioteca de compresión tal como `zlib`. De otro modo, el valor de retorno siempre es `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

- `UNCOMPRESSED_LENGTH(compressed_string)`

  Retorna la longitud de una cadena comprimida antes de la compresión.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
        -> 30
```

- `UNHEX(str)`

  Realiza la operación opuesta a `HEX(str)`. Esto es, interpreta cada par de dígitos hexadecimales en el argumento como números y los convierte al carácter representado por el número. El caracteres resultantes se retornan como cadena binaria.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT 0x4D7953514C;
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

- `UPPER(str)`

  Retorna la cadena *`str`* con todos los caracteres cambiados a mayúsculas según el mapeo del conjunto de caracteres actual (por defecto es ISO-8859-1 Latin1).

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

  Esta función trabaja con múltiples bytes.

12.3.1. Funciones de comparación de cadenas de caracteres

MySQL convierte automáticamente números a cadenas según es necesario y viceversa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

Si quiere convertir un número a cadena explícitamente, use la función `CAST()` :

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
```

Si una función de cadenas da una cadena binaria como argumento, la cadena resultante también es binaria. Un número convertido a cadena se trata como cadena binaria (esto es, es sensible a mayúsculas en comparaciones). Esto afecta sólo a comparaciones.

Normalmente, si una expresión en una comparación de cadenas es sensible a mayúsculas, la comparación se realiza con sensibilidad a mayúsculas.

- *`expr`* `LIKE` *`pat`* `[ESCAPE '`*`escape-char`*`']`

  Coincidencia de patrones usando comparación mediante expresiones regulares SQL. Retorna 1 (`TRUE`) o 0 (`FALSE`). Si *`expr`* o *`pat`* es `NULL`, el resultado es `NULL`.

  El patrón no puede ser una cadena literal. Por ejemplo, puede especificarse como expresión de cadena o columna.

  Con `LIKE` puede usar los siguientes dos caracteres comodín en el patrón:

Carácter	Descrición
`%`	Coincidencia de cualquier número de caracteres, incluso cero caracteres
`_`	Coincide exactemente un carácter

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

  Para testear instancias literales de un carácter comodín, preceda el carácter con el carácter de escape. Si no especifica el carácter `ESCAPE` , se asume '`\`' .

Cadena	Descrición
`\%`	Coincide un carácter '`%`'

`\_`	Coincide un carácter '_'

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

Para especificar un carácter de escape distinto, use la cláusula `ESCAPE` :

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

Los siguientes dos comandos ilustran que la comparación de cadenas no son sensibles a mayúsculas a no ser que uno de los operandos sea una cadena binaria:

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

En MySQL, `LIKE` se permite en expresiones numéricas. (Esta es una extensión del SQL estándar `LIKE`.)

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Nota: Debido a que MySQL usa sintaxis de escape C en cadenas (por ejemplo, '`\n`' para representar carácter de nueva línea), debe doblar cualquier '`\`' que use en cadenas `LIKE` . Por ejemplo, para buscar '`\n`', especifíquelo como '`\\n`'. Para buscar '`\`', especifíquelo como '`\\\\`'; esto es debido a que las antibarras se eliminan una vez por el parser y otra vez cuando la coincidencia con el patrón se realiza, dejando una única antibarra para comparar.

- *`expr`* `NOT LIKE` *`pat`* `[ESCAPE '`*`escape-char`*`']`

  Es lo mismo que `NOT (`*`expr`* `LIKE` *`pat`* `[ESCAPE '`*`escape-char`*`'])`.

- *`expr`* `NOT REGEXP` *`pat`*, *`expr`* `NOT RLIKE` *`pat`*

  Es lo mismo que `NOT (`*`expr`* `REGEXP` *`pat`*`)`.

- *`expr`* `REGEXP` *`pat`*, *`expr`* `RLIKE` *`pat`*

  Realiza una comparación de patrones de una expresión de cadena de caracteres *`expr`* contra un patrón *`pat`*. El patrón puede ser una expresión regular extendida. La sintaxis para expresiones regulares se discute en Apéndice F, *Expresiones regulares en MySQL*. Retorna 1 si *`expr`* coincide con *`pat`*, de otro modo retorna 0. Si *`expr`* o *`pat`* es `NULL`, el resultado es `NULL`. `RLIKE` es un sinónimo de `REGEXP`, debido a compatibilidad con `mSQL`.

  El patrón no necesita ser una cadena literal. Por ejemplo, puede especificarse como una expresión de cadena o columna.

Nota: Debido a que MySQL usa la sintaxis de escape de C en cadenas (por ejemplo, '\n' para representar una nueva línea), de doblar cualquier '\' que use en sus cadenas REGEXP .

REGEXP no es sensible a mayúsculas, excepto cuando se usa con cadenas binarias.

```
mysql> SELECT 'Monty!' REGEXP 'm%y%%';
        -> 0
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 y RLIKE usan el conjunto de caracteres actual (ISO-8859-1 Latin1 por defecto) al decidir el tipo de un carácter. **Atención**: Estos operadores no pueden trabajar con múltiples bytes.

- STRCMP(*expr1*,*expr2*)

  STRCMP() retorna 0 si las cadenas son idénticas, -1 si el primer argumento es menor que el segundo según el orden actual, y 1 en cualquier otro caso.

```
mysql> SELECT STRCMP('text', 'text2');
        -> -1
mysql> SELECT STRCMP('text2', 'text');
        -> 1
mysql> SELECT STRCMP('text', 'text');
        -> 0
```

  En MySQL 5.0, STRCMP() usa el conjunto de caracteres actual cuando realizac comparaciones. Esto hace el comportamiento de comparaciones por defecto insensible a mayúsculas a no ser que alguno de los operandos sea una cadena binaria.

12.4. Funciones numéricas

12.4.1. Operadores aritméticos

Los operadores aritméticos usuales están disponibles. Tenga en cuenta que en el caso de -, +, y *, el resultado se calcula con precisión BIGINT (64-bit) si ambos argumentos son enteros. Si uno de los argumentos es un entero sin signo, y los otros argumentos son también enteros, el resultado es un entero sin signo. Consulte Sección 12.8, “Funciones y operadores de cast”.

- +

  Suma:

```
mysql> SELECT 3+5;
        -> 8
```

- -

Resta:

```
mysql> SELECT 3-5;
        -> -2
```

- `-`

  Menos unario. Cambia el signo del argumento.

  ```
  mysql> SELECT - 2;
          -> -2
  ```

  Nota: Si este operador se usa con `BIGINT`, el valor de retorno es también `BIGINT`. Esto significa que debe eliminar usar `-` con enteros que pueden ser iguales o menores a -2^63.

- `*`

  Multiplicación:

  ```
  mysql> SELECT 3*5;
          -> 15
  mysql> SELECT 18014398509481984*18014398509481984.0;
          -> 324518553658426726783156020576256.0
  mysql> SELECT 18014398509481984*18014398509481984;
          -> 0
  ```

  El resultado de la última expresión es incorrecto ya que el resultado de la multiplicación entera excede el rango de 64-bit de cálculos `BIGINT`. (Consulte Sección 11.2, “Tipos numéricos”.)

- `/`

  División:

  ```
  mysql> SELECT 3/5;
          -> 0.60
  ```

  División por cero produce un resultado `NULL`:

  ```
  mysql> SELECT 102/(1-1);
          -> NULL
  ```

  Una división se calcula con aritmética `BIGINT` sólo en un contexto donde el resultado se convierte a entero.

- `DIV`

  División entera. Similar a`FLOOR()` pero funciona con valores `BIGINT`.

```
mysql> SELECT 5 DIV 2;
        -> 2
```

12.4.2. Funciones matemáticas

Todas las funciones matemáticas retornan `NULL` en caso de error.

- `ABS(X)`

  Retorna el valor absoluto de *X*.

  ```
  mysql> SELECT ABS(2);
          -> 2
  mysql> SELECT ABS(-32);
          -> 32
  ```

  Esta función puede usar valores `BIGINT`.

- `ACOS(X)`

  Retorna el arcocoseno de *X*, esto es, el valor cuyo coseno es *X*. Retorna `NULL` si *X* no está en el rango -1 a 1.

  ```
  mysql> SELECT ACOS(1);
          -> 0
  mysql> SELECT ACOS(1.0001);
          -> NULL
  mysql> SELECT ACOS(0);
          -> 1.5707963267949
  ```

- `ASIN(X)`

  Retorna el arcoseno de *X*, esto es, el valor cuyo seno es *X*. Retorna `NULL` si *X* no está en el rango de -1 a 1.

  ```
  mysql> SELECT ASIN(0.2);
          -> 0.20135792079033
  mysql> SELECT ASIN('foo');
          -> 0
  ```

- `ATAN(X)`

  Retorna la arcotangente de *X*, esto es, el valor cuya tangente es *X*.

  ```
  mysql> SELECT ATAN(2);
          -> 1.1071487177941
  mysql> SELECT ATAN(-2);
          -> -1.1071487177941
  ```

- `ATAN(Y,X)`, `ATAN2(Y,X)`

  Retorna la arcotangente de las variables *X* y *Y*. Es similar a calcular la arcotangente de *Y / X*, excepto que los signos de ambos argumentos se usan para determinar el cuadrante del resultado.

```
mysql> SELECT ATAN(-2,2);
        -> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
        -> 1.5707963267949
```

- `CEILING(X)`, `CEIL(X)`

  Retorna el entero más pequeño no menor a *X*.

```
mysql> SELECT CEILING(1.23);
        -> 2
mysql> SELECT CEIL(-1.23);
        -> -1
```

  Estas dos funciones son sinónimos. Tenga en cuenta que el valor retornado se convierte a `BIGINT`.

- `COS(X)`

  Retorna el coseno de *X*, donde *X* se da en radianes.

```
mysql> SELECT COS(PI());
        -> -1
```

- `COT(X)`

  Retorna la cotangente de *X*.

```
mysql> SELECT COT(12);
        -> -1.5726734063977
mysql> SELECT COT(0);
        -> NULL
```

- `CRC32(expr)`

  Computa un valor de redundancia cíclica y retorna el valor sin signo de 32 bits. El resultado es `NULL` si el argumento es `NULL`. Se espera que el argumento sea una cadena y (si es posible) se trata como una si no lo es.

```
mysql> SELECT CRC32('MySQL');
        -> 3259397556
mysql> SELECT CRC32('mysql');
        -> 2501908538
```

- `DEGREES(X)`

  Retorna el argumento *X*, convertido de radianes a grados.

```
mysql> SELECT DEGREES(PI());
        -> 180
mysql> SELECT DEGREES(PI() / 2);
        -> 90
```

- `EXP(X)`

  Retorna el valor de `e` (la base del logaritmo natural) a la potencia de *X*.

```
mysql> SELECT EXP(2);
        -> 7.3890560989307
mysql> SELECT EXP(-2);
        -> 0.13533528323661
```

- `FLOOR(X)`

  Retorna el valor entero más grande pero no mayor a *X*.

```
mysql> SELECT FLOOR(1.23);
        -> 1
mysql> SELECT FLOOR(-1.23);
        -> -2
```

  Tenga en cuenta que el valor devuelto se convierte a `BIGINT`.

- `LN(X)`

  Retorna el logaritmo natural de *X*, esto es, el logaritmo de *X* base *e*.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

  Esta función es sinónimo a `LOG(X)`.

- `LOG(X)`, `LOG(B,X)`

  Si se llama con un parámetro, esta función retorna el logaritmo natural de *X*.

```
mysql> SELECT LOG(2);
        -> 0.69314718055995
mysql> SELECT LOG(-2);
        -> NULL
```

  Si se llama con dos parámetros, esta función retorna el logaritmo de *X* para una base arbitrária *B*.

```
mysql> SELECT LOG(2,65536);
        -> 16
mysql> SELECT LOG(10,100);
        -> 2
```

`LOG(B,X)` es equivalente a `LOG(X) / LOG(B)`.

- `LOG2(X)`

  Retorna el logaritmo en base 2 de `X`.

```
mysql> SELECT LOG2(65536);
        -> 16
mysql> SELECT LOG2(-100);
        -> NULL
```

`LOG2()` es útil para encontrar cuántos bits necesita un número para almacenamiento. Esta función es equivalente a la expresión `LOG(X) / LOG(2)`.

- `LOG10(X)`

  Retorna el logaritmo en base 10 de `X`.

```
mysql> SELECT LOG10(2);
        -> 0.30102999566398
mysql> SELECT LOG10(100);
        -> 2
mysql> SELECT LOG10(-100);
        -> NULL
```

`LOG10(X)` es equivalente a `LOG(10,X)`.

- `MOD(N,M)`, `N % M`, `N MOD M`

  Operación de módulo. Retorna el resto de `N` dividido por `M`.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
mysql> SELECT 29 MOD 9;
        -> 2
```

Esta función puede usar valores `BIGINT`.

`MOD()` también funciona con valores con una parte fraccional y retorna el resto exacto tras la división:

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

- `PI()`

  Retorna el valor de π (pi). El número de decimales que se muestra por defecto es siete, pero MySQL usa internamente el valor de doble precisión entero.

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.000000000000000000;
        -> 3.141592653589793116
```

- `POW(X,Y)` , `POWER(X,Y)`

  Retorna el valor de *`X`* a la potencia de *`Y`*.

```
mysql> SELECT POW(2,2);
        -> 4
mysql> SELECT POW(2,-2);
        -> 0.25
```

- `RADIANS(X)`

  Retorna el argumento *`X`*, convertido de grados a radianes. (Tenga en cuenta que π radianes son 180 grados.)

```
mysql> SELECT RADIANS(90);
        -> 1.5707963267949
```

- `RAND()`, `RAND(N)`

  Retorna un valor aleatorio en coma flotante del rango de 0 a 1.0. Si se especifica un argumento entero *`N`*, es usa como semilla, que produce una secuencia repetible.

```
mysql> SELECT RAND();
        -> 0.9233482386203
mysql> SELECT RAND(20);
        -> 0.15888261251047
mysql> SELECT RAND();
        -> 0.63553050033332
mysql> SELECT RAND();
        -> 0.70100469486881
mysql> SELECT RAND(20);
        -> 0.15888261251047
```

  Puede usar esta función para recibir registros de forma aleatoria como se muestra aquí:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

  `ORDER BY RAND()` combinado con `LIMIT` es útil para seleccionar una muestra aleatoria de una conjunto de registros:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d
    -> ORDER BY RAND() LIMIT 1000;
```

Tenga en cuenta que `RAND()` en una cláusula `WHERE` se re-evalúa cada vez que se ejecuta el `WHERE`.

`RAND()` no pretende ser un generador de números aleatorios perfecto, pero es una forma rápida de generar números aleatorios ad hoc portable entre plataformas para la misma versión de MySQL.

- `ROUND(X)`, `ROUND(X,D)`

  Retorna el argumento *X*, redondeado al entero más cercano. Con dos argumentos, retorna *X* redondeado a *D* decimales. *D* puede ser negativo para redondear *D* dígitos a la izquierda del punto decimal del valor *X*.

```
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
```

  El tipo de retorno es el mismo tipo que el del primer argumento (asumiendo que sea un entero, doble o decimal). Esto significa que para un argumento entero, el resultado es un entero (sin decimales).

  Antes de MySQL 5.0.3, el comportamiento de `ROUND()` cuando el argumento se encuentra a medias entre dos enteros depende de la implementación de la biblioteca C. Implementaciones distintas redondean al número par más próximo, siempre arriba, siempre abajo, o siempre hacia cero. Si necesita un tipo de redondeo, debe usar una función bien definida como `TRUNCATE()` o `FLOOR()` en su lugar.

  Desde MySQL 5.0.3, `ROUND()` usa la biblioteca de matemática precisa para valores exactos cuando el primer argumento es un valor con decimales:

  - Para números exactos, `ROUND()` usa la regla de "redondea la mitad hacia arriba": Un valor con una parte fracional de .5 o mayor se redondea arriba al siguiente entero si es positivo o hacia abajo si el siguiente entero es negativo. (En otras palabras, se redondea en dirección contraria al cero.) Un valor con una parte fraccional menor a .5 se redondea hacia abajo al siguiente entero si es positivo o hacia arriba si el siguiente entero es negativo.
  - Para números aproximados, el resultado depende de la biblioteca C. En muchos sistemas, esto significa que `ROUND()` usa la regla de "redondeo al número par más cercano": Un valor con una parte fraccional se redondea al entero más cercano.

  El siguiente ejemplo muestra cómo el redondeo difiere para valores exactos y aproximados:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
```

```
| 3          |            2 |
+------------+--------------+
```

Para más información, consulte Capítulo 23, *Matemáticas de precisión*.

- `SIGN(X)`

  Retorna el signo del argumento como `-1`, `0`, o `1`, en función de si *`X`* es negativo, cero o positivo.

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
```

- `SIN(X)`

  Retorna el seno de *`X`*, donde *`X`* se da en radianes.

```
mysql> SELECT SIN(PI());
        -> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
        -> 0
```

- `SQRT(X)`

  Retorna la raíz cuadrada de un número no negativo. *`X`*.

```
mysql> SELECT SQRT(4);
        -> 2
mysql> SELECT SQRT(20);
        -> 4.4721359549996
mysql> SELECT SQRT(-16);
        -> NULL
```

- `TAN(X)`

  Retorna la tangente de *`X`*, donde *`X`* se da en radianes.

```
mysql> SELECT TAN(PI());
        -> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
        -> 1.5574077246549
```

- `TRUNCATE(X,D)`

Retorna el número *X*, truncado a *D* decimales. Si *D* es 0, el resultado no tiene punto decimal o parte fraccional. *D* puede ser negativo para truncar (hacer cero) *D* dígitos a la izquierda del punto decimal del valor *X*.

```
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
```

Todos los números se redondean hacia cero.

12.5. Funciones de fecha y hora

Esta sección describe las funciones que pueden usarse para manipular valores temporales. Consulte Sección 11.3, “Tipos de fecha y hora” para una descripción del rango de los valores que tiene cada fecha y hora y los formatos válidos en que se puedene especificar los valores.

Aquí hay un ejemplo que usa funciones de fecha. La siguiente consulta selecciona todos los registros con un valor `date_col` dentro de los últimos 30 días:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

Tenga en cuenta que la consulta también selecciona registros con fechas futuras.

Las funciones que esperan valores de fecha usualmente aceptan valores de fecha y hora e ignoran la parte de hora. Las funciones que esperan valores de hora usualmente aceptan valores de fecha y hora e ignoran la parte de fecha.

Las funciones que retornan la fecha u hora actuales se evalúan sólo una vez por consulta al principio de la ejecución de consulta. Esto significa que las referencias múltiples a una función tales como `NOW()` en una misma consulta siempre producen el mismo resultado. Este principio también se aplica a `CURDATE()`, `CURTIME()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`, y a cualquiera de sus sinónimos.

En MySQL 5.0, las funciones `CURRENT_TIMESTAMP()`, `CURRENT_TIME()`, `CURRENT_DATE()`, y `FROM_UNIXTIME()` retornan valores en la zona horaria de la conexión, que está disponible como valor de la variable de sistema `time_zone` . Además, `UNIX_TIMESTAMP()` asume que su argumento es un valor de fecha y hora en la zona horaria actual. Consulte Sección 5.9.8, “Soporte de zonas horarias en el servidor MySQL”.

Los rango de retorno en las siguientes descripciones de funciones se aplican en fechas completas. Si la fecha es un valor “cero” o una fecha incompleta como `'2001-11-00'`, las funciones que extraen una parte de la fecha pueden retornar 0. Por ejemplo `DAYOFMONTH('2001-11-00')` retorna 0.

- `ADDDATE(date,INTERVAL expr type)`, `ADDDATE(expr,days)`

  Cuando se invoca con la forma `INTERVAL` del segundo argumento, `ADDDATE()` es sinónimo de `DATE_ADD()`. La función relacionada `SUBDATE()` es sinónimo de `DATE_SUB()`. Para información del argumento `INTERVAL` , consulte la discusión de `DATE_ADD()`.

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
        -> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
        -> '1998-02-02'
```

Si el argumento *`days`* es simplemente un valor entero, entonces MySQL 5.0 lo trata como el número de días a añadir a *`expr`*.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
        -> '1998-02-02'
```

- `ADDTIME(`*`expr`*`,`*`expr2`*`)`

  `ADDTIME()` añade *`expr2`* a *`expr`* y retorna el resultado. *`expr`* es una expresión de fecha u hora y fecha, y *`expr2`* es una expresión temporal.

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999',
    ->                '1 1:1:1.000002');
        -> '1998-01-02 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()` convierte un valor datetime *`dt`* de la zona horaria dada por *`from_tz`* a la zona horaria dada por *`to_tz`* y retorna el valor resultante. Las zonas horarias pueden especificarse como se describe en Sección 5.9.8, “Soporte de zonas horarias en el servidor MySQL”. Esta función retorna `NULL` si los argumentos son inválidos.

  Si el valor se sale del rango soportado por el tipo `TIMESTAMP` al convertirse de *`from_tz`* a UTC, no se realiza ninguna conversión. El rango `TIMESTAMP` se describe en Sección 11.1.2, “Panorámica de tipos de fechas y hora”.

```
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'
```

  Nota: Para usar zonas horarias con nombres tales como `'MET'` o `'Europe/Moscow'`, las tabas de zona horaria deben estar actualizadas correctamente. Consulte Sección 5.9.8, “Soporte de zonas horarias en el servidor MySQL” para instrucciones.

- `CURDATE()`

  Retorna la fecha horaria como valor en formato `'YYYY-MM-DD'` o `YYYYMMDD`, dependiendo de si la fucnión se usa en un contexto numérico o de cadena de caracteres.

```
mysql> SELECT CURDATE();
        -> '1997-12-15'
mysql> SELECT CURDATE() + 0;
```

```
        -> 19971215
```

- `CURRENT_DATE`, `CURRENT_DATE()`

  `CURRENT_DATE` y `CURRENT_DATE()` son sinónimos de `CURDATE()`.

- `CURTIME()`

  Retorna la hora actual como valor en formato `'HH:MM:SS'` o `HHMMSS` dependiendo de si la fucnión se usa en un contexto numérico o de cadena de caracteres.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026
```

- `CURRENT_TIME`, `CURRENT_TIME()`

  `CURRENT_TIME` y `CURRENT_TIME()` son sinónimos de `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP()`

  `CURRENT_TIMESTAMP()` son sinónimos de `NOW()`.

- `DATE(`*`expr`*`)`

  Extrae la parte de fecha de la expresión de fecha o fecha y hora *`expr`*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

- `DATEDIFF(`*`expr`*`,`*`expr2`*`)`

  `DATEDIFF()` retorna el número de días entre la fecha inicial *`expr`* y la fecha final *`expr2`*. *`expr`* y *`expr2`* son expresiones de fecha o de fecha y hora. Sólo las partes de fecha de los valores se usan en los cálculos.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
        -> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
        -> -31
```

- `DATE_ADD(`*`date`*`,INTERVAL `*`expr type`*`)`, `DATE_SUB(`*`date`*`,INTERVAL `*`expr type`*`)`

Estas funciones realizan operaciones aritméticas de fechas. *date* es un valor DATETIME o DATE especificando la fecha de inicio. *expr* es una expresión que especifica el intervalo a añadir o borrar de la fecha de inicio. *expr* es una cadena; puede comenzar con un '-' para intervalos negativos. *type* es una palabra clave que indica cómo debe interpretarse la expresión.

La palabra clave INTERVAL y el especificador type no son sensibles a mayúsculas.

La siguiente tabla muestra cómo se relacionan los argumentos *type* y *expr* :

***type* 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.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

Los valores QUARTER y WEEK están disponibles a partir de MySQL 5.0.0.

MySQL permite cualquier delimitador en el formato *expr* . Los mostrados en la tabla son sugerencias. Si el argumento *date* es un valora DATE y sus cálculos involucarán sólo partes YEAR, MONTH, y DAY (esto es, sin partes de hora), el resultado es un valor DATE . De otro modo, el resultado es un valor DATETIME .

INTERVAL *expr type* se permite en cualquier lado del operador + si la expresión en el otro lado es una fecha o fecha y hora. Para el operador - , INTERVAL *expr type* se permite sólo en la parte derecha, ya que no tiene sentido restar una fecha de un intervalo. (Consulte los ejemplos a continuación.)

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
```

```
        -> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
        -> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1997-12-30 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'
```

Si especifica un intervalo demasiado pequeño (no incluye todas las partes de intervalo que se esperarían de la palabra clave *`type`*), MySQL asume que ha dejado la parte más a la izquierda del valor del intervalo. Por ejemplo, si especifica un `type` de `DAY_SECOND`, se espera que el valor de *`expr`* tenga días, horas, minutos y segundos. Si especifica un valor como '1:10', MySQL asume que las partes de día y hora no se encuentran disponibles y que el valor representa minutos y segundos. En otras palabras, `'1:10' DAY_SECOND` se interpreta de forma que es equivalente a `'1:10' MINUTE_SECOND`. Esto es análogo a la forma en que MySQL interpreta valores `TIME` como representando tiempo transcurrido en lugar de la hora del día.

Si suma o borra de un valor de fecha algo que contenga una parte de fora, el resultado se convierte automáticamente a valor fecha/hora:

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
        -> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
        -> '1999-01-01 01:00:00'
```

Si usa fechas muy mal formadas, el resultado es `NULL`. Si suma `MONTH`, `YEAR_MONTH`, o `YEAR` y la fecha resultante tiene un día mayor que el día máximo para el nuevo mes, el día se ajusta al número máximo del nuevo mes:

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
        -> '1998-02-28'
```

- `DATE_FORMAT(`*`date`*`,`*`format`*`)`

  Formatea el valor *`date`* según la cadena *`format`* . Los siguientes especificadores pueden usarse en la cadena *`format`* :

Especificador	Descripción
`%a`	Día de semana abreviado (`Sun`..`Sat`)
`%b`	Mes abreviado (`Jan`..`Dec`)

%c	Mes, numérico (0..12)
%D	Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd, ...)
%d	Día del mes numérico (00..31)
%e	Día del mes numérico (0..31)
%f	Microsegundos (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minutos, numérico (00..59)
%j	Día del año (001..366)
%k	Hora (0..23)
%l	Hora (1..12)
%M	Nombre mes (January..December)
%m	Mes, numérico (00..12)
%p	AM o PM
%r	Hora, 12 horas (hh:mm:ss seguido de AM o PM)
%S	Segundos (00..59)
%s	Segundos (00..59)
%T	Hora, 24 horas (hh:mm:ss)
%U	Semana (00..53), donde domingo es el primer día de la semana
%u	Semana (00..53), donde lunes es el primer día de la semana
%V	Semana (01..53), donde domingo es el primer día de la semana; usado con %X
%v	Semana (01..53), donde lunes es el primer día de la semana; usado con %x
%W	Nombre día semana (Sunday..Saturday)
%w	Día de la semana (0=Sunday..6=Saturday)
%X	Año para la semana donde domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V
%x	Año para la semana, donde lunes es el primer día de la semana, numérico, cuatro dígitos; usado con %v
%Y	Año, numérico, cuatro dígitos
%y	Año, numérico (dos dígitos)
%%	Carácter '%' literal

Todos los otros caracteres se copian al resultado sin interpretación.

Tenga en cuenta que el carácter '%' se necesita antes de caracteres especificadores de formato.

Los rangos para los especificadores de mes y día comienzan en cero debido a que MySQL permite almacenar fechas incompletas tales como '2004-00-00'.

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
        -> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
```

```
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%D %y %a %d %m %b %j');
        -> '4th 97 Sat 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'
```

- `DAY(date)`

  `DAY()` es sinónimo de `DAYOFMONTH()`.

- `DAYNAME(date)`

  Retorna el nombre del día de la semana para *`date`*.

```
mysql> SELECT DAYNAME('1998-02-05');
        -> 'Thursday'
```

- `DAYOFMONTH(date)`

  Retorna el día del mes para *`date`*, en el rango `1` a `31`.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
        -> 3
```

- `DAYOFWEEK(date)`

  Retorna el índice del día de la semana para *`date`* (1 = domingo, 2 = lunes, ..., 7 = sábado). Estos valores del índice se corresponden con el estándar ODBC.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
        -> 3
```

- `DAYOFYEAR(date)`

  Retorna el día del año para *`date`*, en el rango 1 a 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
        -> 34
```

- `EXTRACT(type FROM date)`

La función `EXTRACT()` usa la misma clase de especificadores de tipo que `DATE_ADD()` o `DATE_SUB()`, pero extrae partes de la fecha en lugar de realizar aritmética de fecha.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
       -> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
       -> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
       -> 20102
mysql> SELECT EXTRACT(MICROSECOND
    ->                FROM '2003-01-02 10:30:00.000123');
        -> 123
```

- `FROM_DAYS(N)`

  Dado un número de día *`N`*, retorna un valor `DATE` .

```
mysql> SELECT FROM_DAYS(729669);
       -> '1997-10-07'
```

  Use `FROM_DAYS()` con precaución en fechas viejas. No se pretende que se use con valores que precedan el calendario Gregoriano (1582). Consulte Sección 12.6, “Qué calendario utiliza MySQL”.

- `FROM_UNIXTIME(`*`unix_timestamp`*`)` , `FROM_UNIXTIME(`*`unix_timestamp`*`,`*`format`*`)`

  Retorna una representación del argumento *`unix_timestamp`* como un valor en formato `'YYYY-MM-DD HH:MM:SS'` o `YYYYMMDDHHMMSS` , dependiendo de si la función se usa en un formato numérico o de cadena de caracteres.

```
mysql> SELECT FROM_UNIXTIME(875996580);
       -> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
       -> 19971004222300
```

  Si se da *`format`*, el resultado se formatea según la cadena *`format`*. *`format`* puede contener los mismos especificadores que los listados en la entrada para la función `DATE_FORMAT()` .

```
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                      '%Y %D %M %h:%i:%s %x');
        -> '2003 6th August 06:22:58 2003'
```

- `GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')`

  Retorna una cadena de formato. Esta función es útil en combinación con las funciones `DATE_FORMAT()` y `STR_TO_DATE()` .

  Los tres valores posibles para el primer argumento y los cinco posibles valores para el segundo argumento resultan en 15 posibles cadenas de formato (para los especificadores usados, consulte la tabla en la descripción de la función `DATE_FORMAT()`).

LLamad a función	Resultado
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'
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'

El formato ISO es ISO 9075, no ISO 8601.

En MySQL 5.0, `TIMESTAMP` puede usarse; `GET_FORMAT()` retorna los mismos valores que para `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'
```

Consulte Sección 13.5.3, “Sintaxis de SET”.

- `HOUR(time)`

  Retorna la hora para *`time`*. El rango del valor de retorno es 0 a 23 para valores de horas del día.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
```

Además, el rango de los valores `TIME` es mucho mayor, así que `HOUR` puede retornar valores mayores que `23`.

```
mysql> SELECT HOUR('272:59:59');
        -> 272
```

- `LAST_DAY(date)`

  Toma una fecha o fecha/hora y retorna el valor correspondiente para el último día del mes. Retorna `NULL` si el argumento es inválido.

```
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` y `LOCALTIME()` son sinónimos de `NOW()`.

- `LOCALTIMESTAMP`, `LOCALTIMESTAMP()`

  `LOCALTIMESTAMP` y `LOCALTIMESTAMP()` son sinónimos de `NOW()`.

- `MAKEDATE(year,dayofyear)`

  Retorna una fecha, dado un año y día del año. *`dayofyear`* debe ser mayor a 0 o el resultado es `NULL`.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
        -> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
        -> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
        -> NULL
```

- `MAKETIME(hour,minute,second)`

  Retorna un valor horario calculado a partir de los argumentos *`hour`*, *`minute`*, y *`second`* .

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

- `MICROSECOND(expr)`

  Retorna los microsegundos a partir del a expresión de hora o fecha/hora *`expr`* como número en el rango de `0` a `999999`.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');
```

```
        -> 10
```

- `MINUTE(time)`

  Retorna el minuto de *`time`*, en el rango `0` a `59`.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
        -> 5
```

- `MONTH(date)`

  Retorna el mes para *`date`*, en el rango `1` a `12`.

```
mysql> SELECT MONTH('1998-02-03');
        -> 2
```

- `MONTHNAME(date)`

  Retorna el nombre completo del mes para *`date`*.

```
mysql> SELECT MONTHNAME('1998-02-05');
        -> 'February'
```

- `NOW()`

  Retorna la fecha y hora actual como valor en formato `'YYYY-MM-DD HH:MM:SS'` o `YYYYMMDDHHMMSS`, dependiendo de si la función se usa en contexto numérico o de cadena de caracteres.

```
mysql> SELECT NOW();
        -> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 19971215235026
```

- `PERIOD_ADD(P,N)`

  Añade *`N`* meses al periodo *`P`* (en el formato `YYMM` o `YYYYMM`). Retorna un valor en el formato `YYYYMM`. Tenga en cuenta que el argumento del periodo *`P`* *no* es una fecha.

```
mysql> SELECT PERIOD_ADD(9801,2);
        -> 199803
```

- `PERIOD_DIFF(P1,P2)`

  Retorna el número de meses entre periodos *`P1`* y *`P2`*. *`P1`* y *`P2`* deben estar en el formato `YYMM` o `YYYYMM`. Tenga en cuenta que los argumentos del periodo *`P1`* y *`P2`* *no* son fechas.

```
mysql> SELECT PERIOD_DIFF(9802,199703);
        -> 11
```

- `QUARTER(date)`

  Retorna el cuarto del año para *`date`*, en el rango `1` a `4`.

```
mysql> SELECT QUARTER('98-04-01');
        -> 2
```

- `SECOND(time)`

  Retorna el segundo para *`time`*, en el rango `0` a `59`.

```
mysql> SELECT SECOND('10:05:03');
        -> 3
```

- `SEC_TO_TIME(seconds)`

  Retorna el argumento *`seconds`* , convertido a horas, minutos y segundos, como un valor en formato `'HH:MM:SS'` o `HHMMSS`, dependiendo de si la función se usa en contexto numérico o de cadena de caracteres.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

- `STR_TO_DATE(str,format)`

- Esta es la inversa de la función `DATE_FORMAT()`. Toma la cadena *`str`* y la cadena de formato *`format`*. `STR_TO_DATE()` retorna un valor `DATETIME` si la cadena de formato contiene parte de fecha y hora, o un valor `DATE` o `TIME` si la cadena contiene sólo parte de fecha u hora.

  Los valores fecha, hora o fecha/hora contenidos en *`str`* deben ser dados en el formato indicado por *`format`*. Para los especificadores que pueden usarse en *`format`*, consulte la tabla en la descripción de la función `DATE_FORMAT()` . Todos los otros caracteres no se interpretan. Si *`str`* contiene un valor fecha, hora o fecha/hora ilegal, `STR_TO_DATE()` retorna `NULL`. A partir de MySQL 5.0.3, un valor ilegal también produce una advertencia.

```
mysql> SELECT STR_TO_DATE('03.10.2003 09.20','%d.%m.%Y %H.%i');
        -> '2003-10-03 09:20:00'
mysql> SELECT STR_TO_DATE('10arp', '%carp');
        -> '0000-10-00 00:00:00'
mysql> SELECT STR_TO_DATE('2003-15-10 00:00:00','%Y-%m-%d %H:%i:%s');
```

```
        -> NULL
```

El chequeo de rango en las partes de los valores de fecha se describe en Sección 11.3.1, “Los tipos de datos DATETIME, DATE y TIMESTAMP”. Esto significa, por ejemplo, que una fecha con una parte de día mayor que el número de días en un mes se permite mientras la parte del día esté en el rango de 1 a 31. También, fechas “cero” o fechas con partes de 0 se permiten.

```
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'
```

- `SUBDATE(`*`date`*`,INTERVAL `*`expr type`*`)`, `SUBDATE(`*`expr`*`,`*`days`*`)`

  Cuando se invoca con la forma `INTERVAL` del segundo argumento, `SUBDATE()` es sinónimo de `DATE_SUB()`. Para información del argumento `INTERVAL` , consulte la discusión para `DATE_ADD()`.

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
```

  La siguiente forma permite el uso de un valor entero para *`days`*. En tales casos, es el número de días a ser borrados de la expresión fecha o fecha/hora *`expr`*.

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
        -> '1997-12-02 12:00:00'
```

  Nota no puede usar formato `"%X%V"` para convertir una cadena año-semana en fecha ya que la combinación de un año y semana no identific unívocamente un año y semana si la semana atraviesa la forntera de un mes. Para convertir un año-semana a fecha, debe especificar el día de la semana:

```
mysql> select str_to_date('200442 Monday', '%X%V %W');
-> 2004-10-18
```

- `SUBTIME(`*`expr`*`,`*`expr2`*`)`

  `SUBTIME()` resta *`expr2`* de *`expr`* y retorna el resultado. *`expr`* es una expresión de hora o fecha/hora, y *`expr2`* es una expresión de hora.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
        -> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

- `SYSDATE()`

  `SYSDATE()` es sinónimo de `NOW()`.

- `TIME(expr)`

  Extrae la parte de hora de la expresión hora o fecha/hora *`expr`*.

```
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(expr,expr2)`

  `TIMEDIFF()` retorna el tiempo entre la hora de inicio *`expr`* y la hora final *`expr2`*. *`expr`* y *`expr2`* son expresiones de hora o de fecha/hora, pero ambas deben ser del mismo tipo.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
    ->                 '1997-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

- `TIMESTAMP(expr)` , `TIMESTAMP(expr,expr2)`

  Con un único argumento, esta función retorna la expresión de fecha o fecha/hora *`expr`* como valor fecha/hora. Con dos argumentos, suma la expresión de hora *`expr2`* a la expresión de fecha o de fecha/hora *`expr`* y retorna el resultado como valor fecha/hora.

```
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(interval,int_expr,datetime_expr)`

  Suma la expresión entera *`int_expr`* a la expresión de fecha o de fecha/hora *`datetime_expr`*. La unidad for *`int_expr`* la da el argumento `interval` , que debe ser uno de los siguientes valores: `FRAC_SECOND`, `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, o `YEAR`.

  El valor *`interval`* puede especificarse usando una de las palabras claves que se muestran, o con un prefijo de `SQL_TSI_`. Por ejemplo, `DAY` o `SQL_TSI_DAY` son legales.

```
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'
```

  `TIMESTAMPADD()` está disponible desde MySQL 5.0.0.

- `TIMESTAMPDIFF(interval,datetime_expr1,datetime_expr2)`

Retorna la diferencia entera entre las expresiones de fecha o de fecha/hora *`datetime_expr1`* y *`datetime_expr2`*. La unidad del resultado se da en el argumento *`interval`*. Los valores legales para *`interval`* son los mismos que los listados en la descripción de la función `TIMESTAMPADD()` .

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
```

`TIMESTAMPDIFF()` está disponible desde MySQL 5.0.0.

- `TIME_FORMAT(`*`time`*`,`*`format`*`)`

  Se usa como la función `DATE_FORMAT()` pero la cadena *`format`* puede contener sólo los especificadores de formato que tratan horas, minutos y segundos. Otros especificadores producen un valor `NULL` o `0`.

  Si el valor *`time`* contiene una parte horaria mayor que `23`, los especificadores de formato horario `%H` y `%k` producen un valor mayor que el rango usual de `0..23`. Los otros especificadores de hora producen la hora modulo 12.

  ```
  mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
          -> '100 100 04 04 4'
  ```

- `TIME_TO_SEC(`*`time`*`)`

  Retorna el argumento *`time`* convertido en segundos.

  ```
  mysql> SELECT TIME_TO_SEC('22:23:00');
          -> 80580
  mysql> SELECT TIME_TO_SEC('00:39:38');
          -> 2378
  ```

- `TO_DAYS(`*`date`*`)`

  Dada la fecha *`date`*, retorna un número de día (el número de dias desde el año 0).

  ```
  mysql> SELECT TO_DAYS(950501);
          -> 728779
  mysql> SELECT TO_DAYS('1997-10-07');
          -> 729669
  ```

  `TO_DAYS()` no está pensado para usarse con valores anteriores al calendario Gregoriano (1582), ya que no tiene en cuenta los días perdidos cuando se cambió el calendario. Consulte Sección 12.6, “Qué calendario utiliza MySQL”.

  Recuerde que MySQL convierte años de dos dígitos en fechas de cuatro dígitos usando las reglas en Sección 11.3, “Tipos de fecha y hora”. Por ejemplo, `'1997-10-07'` y `'97-10-07'` se consideran fechas idénticas:

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');
        -> 729669, 729669
```

Para fechas anteriores a 1582 (y posiblemente un año posterior en otras localizaciones), los resultados de esta función no son fiables. Consulte Sección 12.6, “Qué calendario utiliza MySQL” para más detalles.

- `UNIX_TIMESTAMP()`, `UNIX_TIMESTAMP(`*`date`*`)`

  Si se llama sin argumentos, retorna el timestamp de Unix (segundos desde `'1970-01-01 00:00:00'` GMT) como entero sin signo. Si se llama a `UNIX_TIMESTAMP()` con un argumento *`date`* , retorna el valor del argumento como segundos desde `'1970-01-01 00:00:00'` GMT. *`date`* puede ser una cadena `DATE` , una cadena `DATETIME` , un `TIMESTAMP`, o un número en el formato `YYMMDD` o `YYYYMMDD` en hora local.

  ```
  mysql> SELECT UNIX_TIMESTAMP();
          -> 882226357
  mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
          -> 875996580
  ```

  Cuando se usa `UNIX_TIMESTAMP` en una columna `TIMESTAMP` , la función retorna el valor del timestamp interno directamente, sin conversión implícita “string-to-Unix-timestamp” . Si pasa una fecha fuera de rango a `UNIX_TIMESTAMP()`, retorna `0`, pero tenga en cuenta que sólo se hace un chequeo de rango básico (año de `1970` a `2037`, mes de `01` a `12`, día de `01` a `31`).

  Si quiere restar columnas `UNIX_TIMESTAMP()` puede querer convertir el resultado a enteros sin signo. Consulte Sección 12.8, “Funciones y operadores de cast”.

- `UTC_DATE`, `UTC_DATE()`

  Retorna la fecha UTC actual como valor en formato `'YYYY-MM-DD'` o `YYYYMMDD`, dependiendo si la función se usa en un contexto numérico o de cadenas de caracteres.

  ```
  mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
          -> '2003-08-14', 20030814
  ```

- `UTC_TIME`, `UTC_TIME()`

  Retorna la hora UTC actual como valor en formato `'HH:MM:SS'` or `HHMMSS` dependiendo si la función se usa en un contexto numérico o de cadenas de caracteres.

  ```
  mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
          -> '18:07:53', 180753
  ```

- `UTC_TIMESTAMP`, `UTC_TIMESTAMP()`

Retorna la fecha y hora UTC actual como valor en formato `'YYYY-MM-DD HH:MM:SS'` o `YYYYMMDDHHMMSS` dependiendo si la función se usa en un contexto numérico o de cadenas de caracteres.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
        -> '2003-08-14 18:08:04', 20030814180804
```

- `WEEK(date[,mode])`

  Esta función retorna el número de semana para *`date`*. La forma de dos argumentos de `WEEK()` le permite especificar si la semana comienza en lunes o domingo y si el valor de retorno debe estar en el rango de `0` a `53` o de `1` a `53`. Si el argumento *`mode`* se omite en MySQL 5.0, el valor de la variable de sistema `default_week_format` se usa. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

  La siguiente tabla describe cómo funciona el argumento *`mode`* :

Modo	Primer día de semana	Rango	Semana 1 es la primera semana...
0	Domingo	0-53	con un domingo en este año
1	Lunes	0-53	con más de 3 días este año
2	Domingo	1-53	con un domingo este año
3	Lunes	1-53	con más de 3 días este año
4	Domingo	0-53	con más de 3 días este año
5	Lunes	0-53	con un lunes en este año
6	Domingo	1-53	con más de 3 días este año
7	Lunes	1-53	con un lunes en este año

```
mysql> SELECT WEEK('1998-02-20');
        -> 7
mysql> SELECT WEEK('1998-02-20',0);
        -> 7
mysql> SELECT WEEK('1998-02-20',1);
        -> 8
mysql> SELECT WEEK('1998-12-31',1);
        -> 53
```

Tenga en cuenta que si una fecha cae en la última semana del año prévio, MySQL retorna `0` si no usa `2`, `3`, `6`, o `7` con el argumento opcional `mode` :

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
        -> 2000, 0
```

Se podría argumentar que MySQL debería retornar `52` para la función `WEEK()` , ya que la fecha dada ocurre en la 52a semana de 1999. Decidimos retornar `0` en su lugar porque queríamos que la función devolviera “el número de semana en el año dado.” Esta hace uso de la función `WEEK()` fiable combinada con otras funciones que extraen una parte de fecha de una fecha.

Si prefiere que el resultado a ser evaluado respecto al año que contiene el primer día de la semana para la fecha dada, debe usar `0`, `2`, `5`, o `7` como el argumento `mode` opcional.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

Alternativamente, use la función `YEARWEEK()`:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(date)`

  Retorna el índice de días de la semana para *`date`* (`0` = lunes, `1` = martes, ... `6` = domingo).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
        -> 1
mysql> SELECT WEEKDAY('1997-11-05');
        -> 2
```

- `WEEKOFYEAR(date)`

  Retorna la semana de la fecha como número del rango `1` a `53`. Esta es una función de compatibilidad equivalente a `WEEK(date,3)`.

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
        -> 8
```

- `YEAR(date)`

  Retorna el año para *`date`*, en el rango `1000` a `9999`.

```
mysql> SELECT YEAR('98-02-03');
        -> 1998
```

- `YEARWEEK(date)`, `YEARWEEK(date,start)`

  Retorna año y semana para una fecha. El argumento *`start`* funciona exactamente como el argumento *`start`* de `WEEK()`. El año en el resultado puede ser diferente del año en el argumento fecha para la primera y última semana del año.

```
mysql> SELECT YEARWEEK('1987-01-01');
        -> 198653
```

Tenga en cuenta que el número de semana es diferente de lo que la función `WEEK()` retornaría (`0`) para argumentos opcionales `0` o `1`, como `WEEK()` retorna la semana en el contexto del año dado.

12.6. Qué calendario utiliza MySQL

MySQL usa lo que se conoce como **calendario Gregoriano proleptico**.

Cada país que ha cambiado del calendario Juliano al Gregoriano ha tenido que descartar al menos diez días durante el cambio. Para ver cómo funciona, vamos a mirar el mes de Octubre de 1582, cuando se hizo el primer cambio Juliano-Gregoriano:

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

No hay fechas entre 4 y 15 de Octubre. Esta continuidad se llama el **corte**. Cualquier fecha antes del corte son Julianas, y cualquier fecha posterior es Gregoriana. Las fechas durante el corte no existen.

Un calendario aplicado a fechas cuando no estaban en uso se llama **proleptico**. Por lo tanto, si asumimos que nunca hubo un corte y las reglas Gregorianas funcionaron siempre, tenemos un calendario Gregoriano proleptico. Esto es lo que se usa en MySQL, y es requerido por el estándar SQL. Por esta razón, las fechas anteriores al corte almacenadas como valores MySQL `DATE` o `DATETIME` deben ajustarse para compensar la diferencia. Es importante tener en cuenta que el corte no ocurrió al mismo tiempo en todos los países, y que donde ocurrió más tarde se perdieron más días. Por ejemplo, en Gran Bretaña, tuvo lugar en 1752, cuando el miércoles 2 de Septiembre fue seguido por el jueves 14 de Septiembre; Rusia siguió con el calendario Juliano hasta 1918, perdiendo 13 días en el proceso, y la que se llama popularmente la "Revolución de Octubre" ocurrió en Noviembre según el calendario Gregoriano.

12.7. Funciones de búsqueda de texto completo (Full-Text)

- `MATCH (col1,col2,...) AGAINST (expr [IN BOOLEAN MODE | WITH QUERY EXPANSION])`

  MySQL soporta indexación y búsqueda full-text. Un índice full-text en MySQL es un índice de tipo `FULLTEXT`. Los índices `FULLTEXT` pueden usarse sólo con tablas `MyISAM`; pueden ser creados desde columnas `CHAR`, `VARCHAR`, o `TEXT` como parte de un comando `CREATE TABLE` o añadidos posteriormente usando `ALTER TABLE` o `CREATE INDEX`. Para conjuntos de datos grandos, es mucho más rápido cargar los datos en una tabla que no tenga índice `FULLTEXT` y crear el índice posteriormente, que cargar los datos en una tabla que tenga un índice `FULLTEXT` existente.

  Las restricciones en búsquedas full-text se listan en Sección 12.7.3, “Limitaciones de las búsquedas de texto completo (Full-Text)”.

Las búsquedas full-text se realizan con la función `MATCH()` .

```
mysql> CREATE TABLE articles (
    ->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   title VARCHAR(200),
    ->   body TEXT,
    ->   FULLTEXT (title,body)
    -> );
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)
```

La función `MATCH()` realiza una búsqueda de lenguaje natural para cadenas contra una colección de textos. Una colección es un conjunto de una o más columnas incluídas en un índice `FULLTEXT` . La cadena de búsqueda se da como argumento para `AGAINST()`. Para cada registro en la tabla `MATCH()` retorna un valor de relevancia, esto es, una medida de similaridad entre la cadena de búsqueda y el texto en el registro en las columnas mencionadas en la lista `MATCH()`.

Por defecto, la búsqueda se realiza de forma insensible a mayúsculas. Sin embargo, puede realizar búsquedas sensibles a mayúsculas usando colaciones binarias para columnas indexadas. Por ejemplo, una columna que usa el conjunto de caracteres `latin1` que puede asignarse una colación de `latin1_bin` para hacerla sensible a mayúsculas para búsquedas full-text .

Cuando se usa `MATCH()` en una cláusula `WHERE` , como en el ejemplo precedente, los registros retornados se ordenan automáticamente con la relevancia mayor primero. Los valores relevantes son números en coma flotante no negativos. Relevancia cero significa que no tiene similaridad. La relevancia se computa basada en el número de palabras en el registro, el número de palabras únicas en este registro, el número total de palabras en la colección, y el número de documentos (registros) que contienen una palabra particulas.

Para búsquedas full-text en lenguaje natural, se requiere que las columnas nombradas en la función `MATCH()` sean las mismas columnas incluídas en algún índice `FULLTEXT` en su tabla. Para la consulta precedente, tenga en cuenta que las columnas nombradas en la función `MATCH()` (`title` y `body`) son las mismas que las nombradas en la definición del índice `FULLTEXT` de la tabla `article`. Si quiere buscar el `title` o `body` separadamente, necesitará crear índices `FULLTEXT` para cada columna.

También es posible realizar una búsqueda boolena o una búsqueda con expansión de consulta. Estos tipos de búsqueda se describen en Sección 12.7.1, “Búsquedas booleanas de texto completo (Full-Text)” y Sección 12.7.2, “Búsquedas de texto completo (Full-Text) con expansión de consulta”.

El ejemplo precedente es una ilustración básica mostrando cómo usar la función `MATCH()` donde los registros se retornan para decrementar la relevancia. El siguiente ejemplo muestra cómo recibir los valores de relevancia explícitamente. Los registros retornados no se ordenan debido a que el comando `SELECT` no incluye cláusulas `WHERE` ni `ORDER BY`:

```
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 |
+----+----------------------------------------+
6 rows in set (0.00 sec)
```

El siguiente ejemplo es más complejo. La consulta retorna los valores de relevancia y también ordena los registros en orden decrecente de relevancia. Para conseguir este resultado, debe especificar `MATCH()` dos veces: una vez en la lista `SELECT` y otra en la cláusula `WHERE` . Esto hace que no haya sobrecarga adicional, ya que el optimizador de MySQL se da cuenta que hay dos llamadas `MATCH()` son idénticas y invoca la búsqueda full-text sólo una vez.

```
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)
```

La implementación MySQL de `FULLTEXT` trata cualquier secuencia de caracteres de palabras (letras, dígitos, y subrayados) como una palabra. Esa secuencia puede contener apóstrofes (`'`), poro no más que una en un registro. Esto significa que `aaa'bbb` se trata como una palabra, pero `aaa''bbb` se trata como dos palabras. Los apóstrofoes al principio o fin de una palabra se eliminan por el parser `FULLTEXT` ; `'aaa'bbb'` se parsea como `aaa'bbb`.

El parser `FULLTEXT` determina dónde empiezan y acaban las palabras buscando algunos delimitadores, por ejemplo `' '` (el espacio), `,` (coma), y `.` (punto). Si las palabras no se separan por delmitadores como, por ejemplo, en chino, el parser `FULLTEXT` no puede determinar dónde empieza y acaba una palabra. Para ser capaz de añadir palabras o índices indexados en tales idioomas en un índice `FULLTEXT`, debe preprocesarlos para que se eliminen mediante algún delimitador arbitrario tal como `"`.

Algunas palabras se ignoran en las búsquedas full-text:

- Cualquier palabra demasiado corta se ignora. La longitud mínima de las palabras que se encuentran en búsquedas full-text es de cuatro caracteres por defecto.
- Las palabras en la lista de palabras de parada se ignoran. Una palabra de parada es una palabra tal como “el” o “algún” que es tan común que se considera que no tiene valor semántico. Hay una lista de palabras de parada, pero puede reescribirse con una lista de palabras definidas por el usuario. Consulte Sección 12.7.4, “Afinar búsquedas de texto completo (Full-Text) con MySQL”.

La longitud de palabra mínima y lista de palabras de parada puede cambiarse como se describe en Sección 12.7.4, “Afinar búsquedas de texto completo (Full-Text) con MySQL”.

Cada palabra correcta en la colección y en la consulta se pesa según su significado en la colección o consulta. Esta forma, una palabra que está presente en varios documentos tiene un peso menor (y puede incluso tener peso 0), ya que tiene un valor semántico menor en esta colección particular. De modo similar, si la palabra es rara, recibe un peso mayor. Los pesos de las palabras se combinan para computar la relevancia del registro.

Una técnica de este tipo funciona mejor con colecciones grandes (de hecho, se ajustó con cuidado para funcionar de este modo). Para tablas muy pequeñas, la distribución de palabras no refleja automáticamente su valor semántico, y este modelo puede producir resultados extraños en ocasiones. Por ejemplo, aunque la palabra “MySQL” está presente en cada registro de la tabla `articles`, una búsqueda de esta palabra no da resultados.

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

El resultado de búsqueda es vacío porque la palabra “MySQL” está presente al menos en el 50% de los registros. Como tal, se trata efectivamente como una palabra de parada. Para conjuntos grandes, este es el comportamiento más deseable--una consulta en lenguaje natural no debe retornar cada segundo registro de una tabla de 1GB. Para conjuntos pequeños, puede ser menos deseable.

Una palabra que coíncide la mitad de registros en una tabla es menos deseable para localizar documentos relevantes. De hecho, es más fácil encontrar muchos documentos irrelevantes. Todos sabemos que esto pasa demasiado frecuentemente cuando tratamos de buscar algo en Internet con un motor de búsqueda. Es con este razonamiento que los registros que contienen la palabra se les asigna un valor semántico bajo para *el conjunto particular en que ocurre*. Una palabra dada puede exceder el límite del 50% en un conjunto pero no en otro.

El límite de 50% tiene una implicación significante cuando intenta una primera búsqueda full-text para ver cómo funciona: si crea una tabla e inserta sólo uno o dos registros de texto en ella, cada palabra en el texto aparece al menos en el 50% de los registros. Como resultado, no se retorna ningún resultado. Asegúrese de insertar al menos tres registros, y preferiblemente muchos más.

12.7.1. Búsquedas booleanas de texto completo (Full-Text)

MySQL puede realizar búsquedas full-text booleanas usando el modificador `IN BOOLEAN MODE` :

```
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 ... |
+----+-----------------------+-------------------------------------+
```

Esta consulta recibe todos los registros que contienen la palabra “MySQL” pero que *no* contiene la palabra “YourSQL”.

Búsquedas full-text booleanas tienen estas características:

- No usa el límite del 50%.
- No ordenan registros automáticamente para ordenarlos por relevancia decrecente. Puede verlo en la consulta precedente: El registro con la relevancia mayor es la que contiene “MySQL” dos veces, pero se lista la última, no la primera.
- Pueden funcionar incluso sin un índice `FULLTEXT`, aunque una búsqueda ejecutada de esta forma sería bastante lenta.
- La longitud mínima y máxima de palabra de parámetro de full-text se aplica.
- Se aplica la lista de palabras de parada.

La capacidad de búsqued full-text booleana soporta los siguientes operadores:

- `+`

  Un signo más indica que esta palabra **debe** estar presente en cada registro que se retorne.

- `-`

Un signo menos indica que esta palabra debe **no** estar presente en cualquiera de los registros que se retornan.

- `(sin operador)`

  Por defecto (cuando ni `+` ni `-` se especifica) la palabra es opcional, pero los registros que la contienen se clasifican mejor. Esto mimetiza el comportamiento de `MATCH() ... AGAINST()` sin el modificador `IN BOOLEAN MODE` .

- `> <`

  Estos dos operadores se usan para cambiar la contribución de la palabra al valor de relevancia que se asigna a un registro. El operador `>` incrementa la contribución y el operador `<` lo decrementa. Consulte el ejemplo a continuación.

- `( )`

  Los paréntesis se usan para agrupara palabras en subexpresiones. Los grupos entre paréntesis se pueden anidar.

- `~`

  Una tilde actúa como operador de negación, causando que la contribución de la palabra a la relevancia del registro sea negativa. Esto es útil para marcar palabras “ruidosas”. Un registro que contenga tales palabras se clasifica peor que otros, pero no se excluye, como se haría con el operador `-` .

- `*`

  El asterisco sirve como operador de truncado. A diferencia de otros operadores, debe ser **añadido** a la palabra afectada.

- `"`

  Una frase entre comillas dobles (`'"'`) coincide sólo con registos que contienen la frase *literalmente, como si se hubiera escrito*. El motor de full-text divide la frase en palabras, realiza una búsqueda en el índice `FULLTEXT` de la palabra. Antes de MySQL 5.0.3, el motor realizaba una búsqueda de subcadenas para la frase en el registro en que se encontraban, de forma que la coincidencia debe incluir caracteres no imprimibles en la frase. Desde MySQL 5.0.3, los caracteres no imprimibles no necesitan coincidir exactamente: La búsqueda de frases requiere sólo que las coincidencias contengan exactamente las mismas palabras de la frase y en el mismo orden. Por ejemplo, `"test phrase"` coincide con `"test, phrase"` en MySQL 5.0.3, pero no anteriormente.

  Si la frase no contiene palabras que están en el índice, el resultado es vacío. Por ejemplo, si todas las palabras son palabras de parada o con longitud menor que la mínima para palabras indexadas, el resultado es vacío.

Los siguientes ejemplos demuestran algunas cadenas de búsqueda que usan operadores booleanos full-text:

- `'apple banana'`

  Encuentra registros que contengan al menos una de las dos palabras.

- `'+apple +juice'`

  Encuentra registros que contengan ambas palabras.

- `'+apple macintosh'`

Encuentra registros que contengan la palabra “apple”, pero claficia mejor las que tambíen contengan “macintosh”.

- `'+apple -macintosh'`

  Encuentra registros que contengan la palabra “apple” pero no “macintosh”.

- `'+apple +(>turnover <strudel)'`

  Encuentra registros que contengan las palabras “apple” y “turnover”, o “apple” y “strudel” (en cualquier orden), pero clasifican “apple turnover” mejor que “apple strudel”.

- `'apple*'`

  Encuentra registros que contenga palabras tales como “apple”, “apples”, “applesauce”, o “applet”.

- `'"some words"'`

  Encuentra registros que contienen la frase exacta “some words” (por ejemplo, registros que contengan “some words of wisdom” pero no “some noise words”). Tenga en cuenta que el carácter `'"'` que envuelve la frase son caracteres operadores que delimitan la frase. No son los delimitadores que rodean la cadena de búsqueda en sí.

12.7.2. Búsquedas de texto completo (Full-Text) con expansión de consulta

La búsqueda full-text en MySQL 5.0 soporta expansión de consultas(en particular, su variante “expansión de consultas ciega”). Generalmente esto es útil cuando una frase buscada es demasiado corta, lo que a menudo significa que el usuario se fía de conocimiento implícito que normalemente no tiene el motor de búsqueda full-text. Por ejemplo, uun usuario buscando “database” puede referirse a “MySQL”, “Oracle”, “DB2”, y “RDBMS” todas son frases que deberían coincidir con “databases” y deberían retornarse también. Este es conocimiento implícito.

Expansión de consultas ciega (también conocida como feedback de relevancia automático) se activa añadiendo `WITH QUERY EXPANSION` siguiendo la frase de búsqueda. Funciona realizando la búsqueda dos veces, donde la frase de búsqueda para la segunda búsqueda es la frase de búsqueda original concatenada con los primeros documentos encontrados en la primera búsqueda. Por lo tanto, si uno de estos documentos contiene la palabra “databases” y la palabra “MySQL”, la segunda búsqueda los documentos que contienen la palabra “MySQL” incluso si no contienen la palabra “database”. El siguiente ejemplo muestra esta diferencia:

```
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)
```

Otro ejemplo puede ser buscar libros de Georges Simenon sobre Maigret, cuando un usuario no está seguro de cómo se escribe “Maigret”. Una búsqueda para “Megre and the reluctant witnesses” encuentra sólo “Maigret and the Reluctant Witnesses” sin expansión de consulta. Una búsqueda con expansión de consulta encuentra todos los libros con la palabra “Maigret” en la segunda pasada.

Nota: Debido a que la expansión de búsqueda ciega tiende a incrementar el ruido significativamente retornando documentos no relevantes, sólo tiene sentido cuando una frase de búsqueda es corta.

12.7.3. Limitaciones de las búsquedas de texto completo (Full-Text)

- Las búsquedas full-text las soportan sólo las tablas `MyISAM` .
- En MySQL 5.0, las búsquedas full-text pueden usarse con la mayoría de conjuntos de caracteres multi-byte. La excepción es para Unicode, el conjunto de caracteres `utf8` puede usarse, pero no el conjunto `ucs2` .
- Idiomas ideográficos como Chino y Japonés no tienen delimitadores de palabras. Por lo tanto, el parser `FULLTEXT` *no puede determinar dónde empiezan y acaban las palabras en este y otros idiomas* . Las implicaciones y algunas soluciones del problema se describen en Sección 12.7, “Funciones de búsqueda de texto completo (Full-Text)”.
- Mientras el uso de múltiples conjuntos de caracteres en una misma tabla se soporta, todas las columnas en un índice `FULLTEXT` deben usar el mismo conjunto de caracteres y colación.
- La lista de columnas `MATCH()` debe coincidir exactamente con la lista de columnas en algún índice `FULLTEXT` definido en la tabla, a no ser que `MATCH()` estén `IN BOOLEAN MODE`.
- El argumento de `AGAINST()` debe ser una cadena constante.

12.7.4. Afinar búsquedas de texto completo (Full-Text) con MySQL

La capacidad de búsqueda full-text de MySQL tiene algunos parámetros ajustables por el usuario, aunque añadir más está en la lista de temas pendientes. Puede tener un mayor control en el comportamiento de las búsquedas full-text si tiene una distribución fuente de MySQL ya que algunos cambios requieren modificaciones del código. Consulte Sección 2.8, “Instalación de MySQL usando una distribución de código fuente”.

Tenga en cuenta que la búsqueda full-text está ajustada cuidadosamente para una mayor eficiencia. El modificar el comportamiento por defecto puede decrementarla en la mayoría de los casos. *No cambie el código MySQL a no ser que sepa lo que hace*.

La mayoría de variables full-text que se describen a continuación deben cambiarse en tiempo de arranque del servidor. Se necesita reiniciar el servidor para cambiarlas; no pueden modificarse mientras el servidor está en ejecución.

Algunos cambios de variables requieren que rehaga los índices `FULLTEXT` en sus tablas. Las instrucciones para hacerlo se dan al final de esta sección.

- La longitud mínima y máxima de las palabras a indexar se definen con las variables de sistema `ft_min_word_len` y `ft_max_word_len` . (Consulte Sección 5.3.3, “Variables de sistema del servidor”.) El valor por defecto mínimo es de cuatro caracteres; el valor máximo por defecto depende de la versión de MySQL que use. Si cambia algún valor, debe rehacer los índices `FULLTEXT`. Por ejemplo, si quiere que se puedan buscar palabras de tres caracteres, puede cambiar la variable `ft_min_word_len` poniendo las siguientes líneas en un fichero de opciones:

```
[mysqld]
ft_min_word_len=3
```

A continuación reinicie el servidor y rehaga los índices `FULLTEXT`. Tenga en cuenta las particularidades de `myisamchk` en las instrucciones a continuación.

- Para sobreescribir la lista de palabras de parada por defecto, cambie la variable de sistema `ft_stopword_file`. (Consulte Sección 5.3.3, “Variables de sistema del servidor”.) El valor de la variable debe ser la ruta del fichero que contiene la lista de palabras de parada, o la cadena vacía para desactivar el filtro de palabras de parada. Tras cambiar el valor de esta variable o los contenidos del fichero de palabras de parada, rehaga los índices `FULLTEXT` .

  La lista de palabras de parada es libre, esto es, puede usar cualquier carácter no alfanumérico como el de nueva línea, espacio, o coma para separar las palabras de parada. Las excepción son el subrayado (_) y el apóstrofe sencillo (') que se tratan como parte de una palabra. El conjunto de caracteres de la lista de palabras de parada es el conjunto de caracteres por defecto del servidor; consulte Sección 10.3.1, “Conjunto de caracteres y colación del servidor”.

- El límite del 50% para búsquedas de lenguaje natural está determinada por el esquema de pesos elegido. Para desactivarlo, consulte la siguiente línea en `myisam/ftdefs.h`:

  ```
  #define GWS_IN_USE GWS_PROB
  ```

  Cambie esta línea a:

  ```
  #define GWS_IN_USE GWS_FREQ
  ```

  A continuación recompile MySQL. No hay necesidad de rehacer los índices en este caso. **Nota**: Al hacer esto se decrementa *severamente* la habilidad de MySQL para proporcionar valores apropiados de relevancia para la función `MATCH()`. Si realmente necesita buscar para estas palabras comunes, es mejor buscar usando `IN BOOLEAN MODE` en su lugar, que no observa el límite del 50%.

- Para cambiar los operadores usados para búsquedas booleanas full-text, cambie la variable de sistema `ft_boolean_syntax`. Esta variable también puede cambiarse mientras el servidor está en ejecución, pero debe tener el privilegio `SUPER` para ello. No es necesario rehacer los índices. Variables de sistema del servidor describe las reglas que gobiernan cómo cambiar esta variable.

Si modifica variables full-text que afectan el indexado (`ft_min_word_len`, `ft_max_word_len`, `ft_stopword_file`), o si cambia el fichero de palabras de parada mismo, debe reconstruir los índices `FULLTEXT` tras hacer los cambios y reiniciar el servidor. Para rehacer los índices en este caso, es suficiente hacer una operación de reparación `QUICK` :

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Tenga en cuenta que si usa `myisamchk` para realizar una operación que modifica los índices de tablas (tales como reparar o analizar), los índices `FULLTEXT` se reconstruyen usando los valores por defecto full-text para longitud de palabras mínima y máxima y el fichero de palabras de parada a no ser que especifique otro. Esto puede hacer que las consultas fallen.

El problema ocurre porque estos parametros sólo son conocidos por el servidor. No se almacenan en ficheros índices `MyISAM`. Para evitar este problema si ha modificado la longitud mínima o máxima de palabra o el fichero de palabras de parada en el servidor, especifique los mismos valores de `ft_min_word_len`, `ft_max_word_len`, y `ft_stopword_file` a `myisamchk` que usa para `mysqld`.

Por ejemplo, si ha puesto que la longitud de palabra mínima a 3, puede reparar una tabla con un `myisamchk` como este:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

Para asegurar que `myisamchk` y el servidor usan los mismos valores para parámetros full-text, puede poner cada uno en las secciones `[mysqld]` y `[myisamchk]` de un fichero de opciones:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

Una alternativa a usar `myisamchk` es usar `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, o `ALTER TABLE`. Estos comandos los realiza el servidor, que conoce los valores apropiados de los parámetros full-text a usa.

12.7.5. Cosas por hacer en búsquedas de texto completo (Full-Text)

- Rendimiento mejorado para todas las operaciones `FULLTEXT`.
- Operadores de proximidad.
- Soporte de “always-index words.” Pueden ser cualquier cadena de caracteres que quiera el usuario para tratarlas como palabras, tales como “C++”, “AS/400”, o “TCP/IP”.
- Soporte para búsqueda full-text en tablas `MERGE`.
- Soporte para el conjunto de caracteres `ucs2` .
- Hace que la lista de palabras de parada dependente del idioma del conjunto de datos.
- Stemming
- Pre-parser genérico proporcionado por el usuario UDF .
- Hace el modelo más flexible (añadiendo algunos parámetros ajustables a `FULLTEXT` en los comandos `CREATE TABLE` y `ALTER TABLE`).

12.8. Funciones y operadores de cast

- `BINARY`

  El operador `BINARY` convierte la cadena a continuación a una cadena binaria. Esta es una forma fácil de forzar una comparación de columna byte a byte en lugar de carácter a carácter. Esto hace que la comparación sea sensible a mayúsculas incluso si no está definida como `BINARY` o `BLOB`. `BINARY` también hace que los espacios finales sean significativos.

  ```
  mysql> SELECT 'a' = 'A';
          -> 1
  mysql> SELECT BINARY 'a' = 'A';
          -> 0
  mysql> SELECT 'a' = 'a ';
          -> 1
  mysql> SELECT BINARY 'a' = 'a ';
          -> 0
  ```

  `BINARY` afecta la comparación; puede darse antes de cualquier operando con el mismo resultado.

`BINARY str` es una abreviación de `CAST(str AS BINARY)`.

Tenga en cuenta que en algunos contextos, si cambia el tipo de una columna `BINARY` indexada, MySQL no es capaz de usar el índice eficientemente.

Si quiere comparar un valor `BLOB` u otra cadena binaria de forma no sensible a mayúsculas, puede hacerlo teniendo en cuenta que las cadenas binarias no tienen conjunto de caracteres, y por lo tanto no tienen concepto de mayúsculas. Para una comparación no sensible a mayúsculas, use la función `CONVERT()` para convertir el valor de cadena a un conjunto de caracteres no sensible a mayúsculas. El resultado es una cadena no binaria. El resultado es una cadena no binaria, así que la operación `LIKE` no es sensible a mayúsculas:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

Para usar un conjunto de caracteres diferente, substituya su nombre por `latin1` en el comando precedente.

`CONVERT()` puede usarse más generalmente para comparar cadenas que se representan en distintos conjuntos de caracteres.

- `CAST(expr AS type)`, `CONVERT(expr,type)` , `CONVERT(expr USING transcoding_name)`

  Las funciones `CAST()` y `CONVERT()` pueden usarse para tomar un valor de un tipo y producir un valor de otro tipo.

  `type` puede ser uno de los siguientes valores:

  - `BINARY`
  - `CHAR`
  - `DATE`
  - `DATETIME`
  - `DECIMAL`
  - `SIGNED [INTEGER]`
  - `TIME`
  - `UNSIGNED [INTEGER]`

  `BINARY` produce una cadena binaria. Consulte la entrada para el operador `BINARY` en esta sección para una descripción de cómo afecta esto a las comparaciones.

  El tipo `DECIMAL` está disponible desde MySQL 5.0.8.

  `CAST()` y `CONVERT(... USING ...)` son sintaxis SQL estándar. La forma no-`USING` de `CONVERT()` es sintaxis ODBC .

  `CONVERT()` con `USING` se usa para convertir datos entre distintos conjuntos de caracteres. En MySQL, los nombres transcodificados son los mismos que los nombres de los conjuntos de caracteres

correspondientes. Por ejemplo, este comando convierte la cadena `'abc'` en el conjunto de caracteres por defecto del servidor a la cadena correspondiente en el conjunto de caracteres `utf8` :

```
SELECT CONVERT('abc' USING utf8);
```

Las funciones de conversión son útiles cuando quiere crear una columna con un tipo específico en un comando `CREATE ... SELECT`:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

Las funciones también pueden ser útiles para ordenar columnas `ENUM` en orden léxico. Normalmente ordenar columnas `ENUM` se hace usando el valor numérico interno. Convertir los valores en `CHAR` resulta en orden léxico:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST(str AS BINARY)` es lo mismo que `BINARY str`. `CAST(expr AS CHAR)` trata la expresión como una cadena con el conjunto de caracteres por defecto.

`CAST()` cambia el resultaod si lo usa como parte de una expresión más compleja como `CONCAT('Date: ',CAST(NOW() AS DATE))`.

No debe usar `CAST()` para extraer datos en datos en distintos formatos sino usar funciones de cadenas como `LEFT()` or `EXTRACT()`. Consulte Sección 12.5, “Funciones de fecha y hora”.

Para convertir una cadena en un valor numérico en un contexto numérico, normalmente no tiene que hacer nada más que usar el valor de la cadena como si fuera un número:

```
mysql> SELECT 1+'1';
       -> 2
```

Si usa un número en un contexto de cadenas, el número se convierte automáticamente en una cadena `BINARY` string.

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

MySQL soporta aritmética con valores con y sin signo de 64-bit. Si usa operadores numéricos (tales como `+`) y uno de los operandos es un entero sin signo, el resultado no tiene signo. Puede cambiar este comportamiento usando los operadores `SIGNED` y `UNSIGNED` para cambiar la operación a un entero con o sin signo de 64-bit , respectivamente.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
        -> -1
```

Tenga en cuenta que si un operando es un valor de coma flotante, el resultado es de coma flotante y no está afectado por la regla precedente. (En este contexto, los valores de la columna `DECIMAL` se tratan como valores con punto flotante.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

Si usa una cadena en una operación aritmética, se convierte en número en coma flotante.

12.9. Otras funciones

12.9.1. Funciones bit

MySQL usa aritmética `BIGINT` (64-bit) para operaciones de bit, así que estos operadores tienen un rango máximo de 64 bits.

- `|`

  OR bit a bit:

  ```
  mysql> SELECT 29 | 15;
         -> 31
  ```

  El resultado es un entero de 64 bits.

- `&`

  AND bit a bit:

  ```
  mysql> SELECT 29 & 15;
         -> 13
  ```

  El resultado es un entero de 64 bits.

- `^`

  XOR bit a bit:

  ```
  mysql> SELECT 1 ^ 1;
         -> 0
  mysql> SELECT 1 ^ 0;
         -> 1
  mysql> SELECT 11 ^ 3;
         -> 8
  ```

  El resultado es un entero de 64 bits.

  Desplaza un número largo (`BIGINT`) a la izquierda.

- `<<`

  ```
  mysql> SELECT 1 << 2;
         -> 4
  ```

  El resultado es un entero de 64 bits.

- `>>`

Desplaza un número largo (`BIGINT`) a la derecha.

```
mysql> SELECT 4 >> 2;
        -> 1
```

El resultado es un entero de 64 bits.

- `~`

  Invierte todos los bits.

```
mysql> SELECT 5 & ~1;
        -> 4
```

El resultado es un entero de 64 bits.

- `BIT_COUNT(N)`

  Retorna el número de bits en el argumento *`N`*.

```
mysql> SELECT BIT_COUNT(29);
        -> 4
```

12.9.2. Funciones de cifrado

Las funciones en esta sección cifran y descifran valores. Si quiere almacenar resultados de una función de cifrado que puede contaner valores arbitrarios de bytes, use una columna `BLOB` en lugar de `CHAR` o `VARCHAR` para evitar problemas potenciales con eliminación de espacios finales que pueden cambiar los valores de datos.

- `AES_ENCRYPT(str,key_str)` , `AES_DECRYPT(crypt_str,key_str)`

  Estas funciones permiten el cifrado y descifrado de datos usando el algoritmo oficial AES (Advanced Encryption Standard), conocido anteriormente como "Rijndael." Se usa un cifrado con una clave de 128-bit, pero puede ampliarlo hasta 256 bits modificando las fuentes. Elegimos 128 porque es mucho más rápido y de momento es suficientemente seguro.

  Los argumentos de entrada pueden ser de cualquier longitud. Si algún argumento es `NULL`, el resultado de esta función también es `NULL`.

  Debido a que AES es un algoritmo a nivel de bloques, se usa relleno para cadenas de longitud impar y así la longitud de la cadena resultante puede calcularse como `16 * (trunc(string_length / 16) + 1)`.

  Si `AES_DECRYPT()` detecata datos inválidos o relleno incorrecto, retorna `NULL`. Sin embargo, es posible para `AES_DECRYPT()` retornar un valor no `NULL` (posiblemente basura) si los datos de entrada o la clave son inválidos.

  Puede usar la función AES para almacenar datos de forma cifrada modificando sus consultas:

```
```

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

Puede obtener incluso mejor seguridad si no transfiere la clave a través de la conexión para cada consulta, que puede hacerse almacenando la clave en una variable del servidor al hacer la conexión. Por ejemplo:

```
SELECT @password:='my password';
INSERT INTO t VALUES (1,AES_ENCRYPT('text',@password));
```

`AES_ENCRYPT()` y `AES_DECRYPT()` pueden considerarse las funciones de cifrado criptográficamente más seguras disponibles en MySQL.

- `DECODE(crypt_str,pass_str)`

  Descifra la cadena cifrada *`crypt_str`* usando *`pass_str`* como contraseña. *`crypt_str`* debe ser una cadena retornada de `ENCODE()`.

- `ENCODE(str,pass_str)`

  Cifra *`str`* usando *`pass_str`* como contraseña. Para descifrar el resultado, use `DECODE()`.

  El resultado es una cadena binaria de la misma longitud que *`str`*. Si quiere guardarlo en una columna, use una columna de tipo `BLOB` .

- `DES_DECRYPT(crypt_str[,key_str])`

  Descifra una cadena cifrada con `DES_ENCRYPT()`. En caso de error, esta función retorna `NULL`.

  Tenga en cuenta que esta función funciona sólo si MySQL se configura con soporte SSL. Consulte Sección 5.7.7, “Usar conexiones seguras”.

  Si no se da argumento *`key_str`*, `DES_DECRYPT()` examina el primer byte de la cadena cifrada para determinar el número de clave DES que se usó para cifrar la cadena original, y luego lee la clave del fichero clave DES para descifrar el mensaje. Para que esto funcione, el usuario debe tener el privilegio `SUPER`. El fichero clave puede especificarse con la opción del servidor `--des-key-file` .

  Si le pasa a esta función el argumento *`key_str`* , esta cadena se usa como la clave para descifrar el mensaje.

  Si el argumento *`crypt_str`* no parece una cadena cifrada, MySQL retorna *`crypt_str`*.

- `DES_ENCRYPT(str[,(key_num|key_str)])`

  Cifra la cadena con la clave dada usando el algoritmo triple-DES. En caso de error, retorna `NULL`.

  Tenga en cuenta que esta función funciona sólo si MySQL se configura con soporte SSL. Consulte Sección 5.7.7, “Usar conexiones seguras”.

  La clave de cifrado a usar se elige basada en el segundo argumento de `DES_ENCRYPT()`, si se ha dado uno:

Argumento	Descripción
Sin argumento	Se usa la primera clave del fichero clave DES.
key_num	El número de clave dado (0-9) del fichero clave DES se usa.
key_str	Se usa la cadena clave dada para cifrar *str*.

El fichero clave puede especificarse con la opción de servidor `--des-key-file` .

La cadena retornada es una cadena binaria donde el primer carácter es `CHAR(128 | key_num)`.

Se añade 128 para hacer más sencillo reconocer una clave cifrada. Si usa una cadena clave, *key_num* es 127.

La longitud de la cadena para el resultado es *new_len = orig_len + (8-(orig_len % 8))+1*.

Cada línea en el fichero clave DES tiene el siguiente formato:

```
key_num des_key_str
```

Cada *key_num* debe ser un número en el rango de `0` a `9`. Las líneas en el fichero pueden estar en cualquier orden. *des_key_str* es la cadena que se usa para cifrar el mensaje. Entre el número y la clave debe haber un espacio como mínimo. La primera clave es la clave usada por defecto si no especifica ningún argumento clave para `DES_ENCRYPT()`

Puede decir a MySQL que lea un nuevo valor de clave del fichero clave con el comando `FLUSH DES_KEY_FILE`. Esto necesita el privilegio `RELOAD` .

Un beneficio de tener un conjunto de claves por defecto es que da a las aplicaciones una forma de chequear la existencia de valores de columna cifrados, sin dar al usuario final el derecho de descifrarlos.

```
mysql> SELECT customer_address FROM customer_table
     > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCRYPT(str[,salt])`

  Cifra *str* usando la llamada de sistema Unix `crypt()` . El argumento *salt* debe ser una cadena con al menos dos caracteres. Si no se da argumento *salt* , se usa un valor aleatorio.

  ```
  mysql> SELECT ENCRYPT('hello');
         -> 'VxuFAJXVARROc'
  ```

  `ENCRYPT()` ignora todo excepto los primeros ochos caracteres de *str*, al menos en algunos sistemas. Este comportamiento viene determinado por la implementación de la llamada de sistema `crypt()` subyacente.

  Si `crypt()` no está disponible en su sistema (como pasa en Windows), `ENCRYPT()` siempre retorna `NULL`. Debido a esto, recomandamos que use `MD5()` o `SHA1()` en su lugar, y que estas dos funciones existen en todas las plataformas.

- `MD5(str)`

Calcula una checksum MD5 de 128-bit para la cadena. El valor se retorna como una cadena binaria de dígitos 32 hex ,o `NULL` si el argumento era `NULL`. El valor de retorno puede usarse como clave hash, por ejemplo.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

Este es el "RSA Data Security, Inc. MD5 Message-Digest Algorithm."

Si quiere convertir el valor a mayúsculas, consulte la descripción de conversiones de cadenas binarias dada en la entrada del operador `BINARY` en Sección 12.8, “Funciones y operadores de cast”.

- `OLD_PASSWORD(str)`

  `OLD_PASSWORD()` se añadió en MySQL 4.1,cuando se cambión la implementación de `PASSWORD()` para mejorar la seguridad. `OLD_PASSWORD()` retorna el valor de la implementación pre-4.1 de `PASSWORD()`, y está hecha para permitirle resetear contraseñas para cualquier cliente pre-4.1 que necesite conectar a su versión 4.1 o posterior de MySQL server sin bloquearlo. Consulte Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1”.

- `PASSWORD(str)`

  Calcula y retorna una cadena de contraseña de la contraseña en texto plano *`str`*, o `NULL` si el argumento era `NULL`. Esta es la función que se usa para cifrar contraseñas MySQL para almacenar en la columna `Password` de la tabla `user` .

  ```
  mysql> SELECT PASSWORD('badpwd');
          -> '7f84554057dd964b'
  ```

  El cifrado de `PASSWORD()` es de un sentido (no reversible).

  `PASSWORD()` no realiza el cifrado de contraseña de la misma forma que se cifran las contraseñas Unix. Consulte `ENCRYPT()`.

  Nota: La función `PASSWORD()` se usa por el sistema de autenticación en MySQL Server; *no* no debe usarlo en su propias aplicaciones. Para ese propósito, use `MD5()` o `SHA1()` en su lugar. Consulte RFC 2195 para más información acerca de tratar contraseñas y autenticación de forma segura en su aplicación.

- `SHA1(str)`, `SHA(str)`

  Calcula una checksum SHA1 de 160-bit para la cadena, como se describe en RFC 3174 (Secure Hash Algorithm). El valor se retorna como cadnea de 40 dítigos hexadecimales, o `NULL` si el argumento era `NULL`. Uno de los usos posibles para esta función es una clave hash. También puede usarlo como función criptográficamente segura para almacenar contraseñas

  ```
  mysql> SELECT SHA1('abc');
          -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
  ```

`SHA1()` puede considerarse un equivalente criptográficamente más seguro que `MD5()`. `SHA()` es sinónimo de `SHA1()`.

12.9.3. Funciones de información

- `BENCHMARK(count,expr)`

  La función `BENCHMARK()` ejecuta la expresión *`expr`* repetidamente *`count`* veces. Puede usarse para ver lo rápido que MySQL procesa la expresión. El valor resultado siempre es `0`. El uso pretendido es desde dentro del cliente `mysql`, que reporte tiempos de ejecución de consultas:

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+--------------------------------------------+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+--------------------------------------------+
|                                          0 |
+--------------------------------------------+
1 row in set (4.74 sec)
```

  El tiempo reportado es el tiempo transcurrido en el cliente final no el tiempo de CPU en el servidor. Se recomienda ejecutar `BENCHMARK()` varias veces, y interpretar el resultado teniendo en cuenta la carga de la máquina servidor.

- `CHARSET(str)`

  Retorna el conjunto de caracteres el argumento cadena.

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

- `COERCIBILITY(str)`

  Retorna la coerzabilidad de la colación del argumento.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
```

  El valor de retorno tiene los siguientes significados:

Coerzabilidad	Significado	Ejemplo
`0`	Colación explícita	Valor con la cláusula `COLLATE`
`1`	Sin colación	Concatenación de cadenas con distintas colaciones

2	Colación implícita	Valor de columna
3	Constante de sistema	Valor de retorno `USER()`
4	Coercible	Cadena literal
5	Ignorable	`NULL` o una expresión derivada de `NULL`

Antes de MySQL 5.0.3, los valores de retorno se muestran como sigue, y las funciones tales como `USER()` tienen una coerzabilidad de 2:

Coerzabilidad	Significado	Ejemplo
0	Colación explícita	Valor con la cláusula `COLLATE`
1	Sin colación	Concatenación de cadenas con distintas colaciones
2	Colación implícita	Valor de columna
3	Coercible	Cadena literal

Los valores menores tienen precedencia mayor.

- `COLLATION(str)`

  Retorna la colación para el conjunto de caracteres de la cadena dada.

```
mysql> SELECT COLLATION('abc');
        -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
        -> 'utf8_general_ci'
```

- `CONNECTION_ID()`

  Retorna el ID de la conexión (ID del thread) para la conexión. Cada conexión tiene su propio y único ID.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

- `CURRENT_USER()`

  Retorna la combinación de nombre de usuario y de equipo que tiene la sesión actual. Este valor se corresponde con la cuenta MySQL que determina sus privilegios de aceso. Puede ser distinto al valor de `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'
```

El ejemplo ilustra que aunque el cliente especica un nombre de usuario de `davida` (como se indica por el valor de la función `USER()`), el servidor autentica al cliente usando una cuenta de usuario anónimo (como se ve por la parte de usuario vacía del valor `CURRENT_USER()`). Una forma en que esto puede ocurrir es que no haya cuenta listada en las cuentas de permisos para `davida`.

En MySQL 5.0, la cadena retornada por `CURRENT_USER()` usa el conjunto de caracteres `utf8` .

- `DATABASE()`

  Retorna el nombre de base de datos por defecto (actual). En MySQL 5.0, la cadena tiene el conjunto de caracteres `utf8` .

  ```
  mysql> SELECT DATABASE();
          -> 'test'
  ```

  No hay base de datos por defecto, `DATABASE()` retorna `NULL`.

- `FOUND_ROWS()`

  Un comando `SELECT` puede incluir una cláusula `LIMIT` para restringir el número de registros que el servidor retorna al cliente. En algunos casos, es deseable saber cuántos registos habría retornado el comando sin `LIMIT`, pero sin volver a lanzar el comando. Para obtener este conteo de registros, incluya la opción `SQL_CALC_FOUND_ROWS` en el comando `SELECT` , luego invoque `FOUND_ROWS()` :

  ```
  mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
      -> WHERE id > 100 LIMIT 10;
  mysql> SELECT FOUND_ROWS();
  ```

  El segundo `SELECT` retorna un número indicando cuantos registros habría retornado el primer `SELECT` sin la cláusula `LIMIT` . (Si el comando precendente no incluye la opción `SQL_CALC_FOUND_ROWS` , `FOUND_ROWS()` puede retornar un resultado distinto cuando se usa `LIMIT` y cuando no.)

  Tenga en cuenta que si usa `SELECT SQL_CALC_FOUND_ROWS`, MySQL debe calcular cuántos registros hay en el conjunto de resultdos completo. Sin embargo, esto es más rápido que ejecutar la consulta de nuevo sin `LIMIT`, ya que el conjunto de resultados no necesita ser enviado al cliente.

  `SQL_CALC_FOUND_ROWS` y `FOUND_ROWS()` pueden ser útiles en situaciones donde puede querer restringir el número de registros que retorna una consulta, pero también determinar el número de registros en el conjunto de resultados entero sin ejecutar la consulta de nuevo. Un ejemplo es el script Web que presenta una salida paginada conteniendo enlaces a las páginas que muestran otras secciones de un resultado de búsqueda. Usando `FOUND_ROWS()` puede determinar cuántas páginas necesita para el resto de resultados.

  El uso de`SQL_CALC_FOUND_ROWS` y `FOUND_ROWS()` es más complejo para consultas `UNION` que para comandos `SELECT` simples, ya que `LIMIT` puede ocurrir en varios lugares en una `UNION`. Puede aplicarse a comandos `SELECT` individuales en la `UNION`, o global en el resultado de `UNION` como conjunto.

La intención de SQL_CALC_FOUND_ROWS para UNION es que debe retornar el número de registros que se retornarían sin un LIMIT globar. Las condiciones para uso de SQL_CALC_FOUND_ROWS con UNION son:

- La palabra clave SQL_CALC_FOUND_ROWS debe aparecer en el primer SELECT de la UNION.
- El valor de FOUND_ROWS() es exacto sólo si se usa UNION ALL . Si se usa UNION sin ALL, se eliminan duplicados y el valor de FOUND_ROWS() es sólo aproximado.
- Si no hay LIMIT en UNION, se ignora SQL_CALC_FOUND_ROWS y retorna el número de registros en la tabla temporal que se crea para procesar UNION.

- LAST_INSERT_ID(), LAST_INSERT_ID(*expr*)

  Retorna el último valor generado automáticamente que se insertó en una columna AUTO_INCREMENT.

```
mysql> SELECT LAST_INSERT_ID();
        -> 195
```

  El último ID generado se mantiene en el servidor para cada conexión. Esto significa que el valor de la función retorna a cada cliente el valor AUTO_INCREMENT más reciente generado por ese cliente. Este valor no puede ser afectado por otros clientes, incluso si generan valores AUTO_INCREMENT ellos mismos. Este comportamiento asegura que reciba sus propios IDs sin tener en cuenta la actividad de otros clientes y sin la necesidad de bloqueos o transacciones.

  El valor de LAST_INSERT_ID() no cambia si actualiza la columna AUTO_INCREMENT de un registro con un valor no mágico (esto es, un valor que no es NULL ni 0).

  Si inserta varios registros a la vez con un comando de inserción LAST_INSERT_ID() retorna el valor del primer registro insertado. La razón para esto es hacer posible reproducir fácilmente el mismo comando INSERT contra otro servidor.

  Si usa INSERT IGNORE y el registro se ignora, el contador AUTO_INCREMENT no se incrementa y LAST_INSERT_ID() retorna 0, lo que refleja que no se ha insertado ningún registro. (Antes de MySQL 4.1, AUTO_INCREMENT el contador se incrementa y LAST_INSERT_ID() retorna el nuevo valor.)

  Si se da *expr* como argumento para LAST_INSERT_ID(), el valor del argumento se retorna por la función y se recuerda como el siguiente valor a ser retornado por LAST_INSERT_ID(). Esto puede usarse para simular secuencias:

  - Cree una tabla para guardar el contador de secuencia y inicializarlo:

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

  - Use la tabla para generar números de secuncia como aquí:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

    El comando UPDATE incrementa el contador de secuencia y causa que la siguiente llamada a LAST_INSERT_ID() retorne el valor actualizado. El comando SELECT recibe ese valor. La función

de la API C `mysql_insert_id()` puede usarse para obtener el valor . Consulte Sección 24.2.3.34, “`mysql_insert_id()`”.

Puede generar secuencias sin llamar a `LAST_INSERT_ID()`, pero la utilidad de usar esta función de esta forma es que el valor ID se mantiene en el servidor como el último valor generado automáticamente. Es válido para multi usuarios porque varios clientes pueden realizar el comando `UPDATE` y obtener su propio valor de secuencia con el comando `SELECT` (o `mysql_insert_id()`), sin afectar o ser afectado por otros clientes que generen sus propios valores de secuencia.

Tenga en cuenta que `mysql_insert_id()` sólo se actualiza tras los comandos `INSERT` y `UPDATE` , así que no puede usar la función de la API C para recibir el valor de `LAST_INSERT_ID(`*`expr`*`)` tras ejecutar otros comandos SQL como `SELECT` o `SET`.

- `ROW_COUNT()`

  `ROW_COUNT()` retorna el número de registros actualizados, insertados o borrados por el comando precedente. Esto es lo mismo que el número de registros que muestra el cliente `mysql` y el valor de la función de la API C `mysql_affected_rows()` .

```
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()` se añadió en MySQL 5.0.1.

- `SESSION_USER()`

  `SESSION_USER()` es sinónimo de `USER()`.

- `SYSTEM_USER()`

  `SYSTEM_USER()` es sinónimo de `USER()`.

- `USER()`

  Retorna el nombre de usuario y de equipo de MySQL actual.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

  El valor indica el nombre de usuario especificado al conectar con el servidor, y el equipo cliente desde el que se está conectando. El valor puede ser distinto del de `CURRENT_USER()`.

  Puede extraer la parte de nombre de usuario así:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
        -> 'davida'
```

  En MySQL 5.0, `USER()` retorna un valor en el conjunto de caracteres `utf8`, así que debe asegurarse que la literal `'@'` se interpreta en el conjunto de caracteres:

```
mysql> SELECT SUBSTRING_INDEX(USER(),_utf8'@',1);
        -> 'davida'
```

- `VERSION()`

  Retorna una cadena que indica la versión del servidor MySQL. La cadena usa el conjunto de caracteres `utf8` .

```
mysql> SELECT VERSION();
        -> '5.0.9-standard'
```

  Tenga en cuenta que si su cadena de versión acaba con `-log` significa que el logueo está activado.

12.9.4. Funciones varias

- `DEFAULT(col_name)`

  Retorna el valor por defecto para una columna de tabla. A partir de MySQL 5.0.2, retorna un error si la columna no tiene valor por defecto.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)`

  Formatea el número *`X`* a un formato como `'#,###,###.##'`, redondeado a *`D`* decimales, y retorna el resultado como una cadena. Si *`D`* es `0`, el resultado no tiene punto decimar o parte fraccional.

```
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'
```

- `GET_LOCK(str,timeout)`

  Intenta obtener un bloqueo con un nombre dado por la cadena *`str`*, con un tiempo máximo de *`timeout`* segundos. Retorna `1` si el bloqueo se obtiene con éxito, `0` si el intento supera el tiempo máximo (por ejemplo, debido a que otro cliente haya bloqueado el nombre préviamente), o `NULL` si ocurre un error (tal como quedarse sin memoria o que el flujo acabe con `mysqladmin kill`). Si tiene un bloqueo obtenido con `GET_LOCK()`, se libera cuando ejecuta `RELEASE_LOCK()`, ejecuta un nuevo `GET_LOCK()`, o su conexión termina (normal o anormalmente).

  Esta función puede usarse para implementar bloqueos de aplicaciones o simular bloqueo de registros. Los nombres se bloquean en el servidor. Si un nombre lo ha bloqueado un cliente, `GET_LOCK()` bloquea cualquier petición de otro cliente para bloquear el mismo nombre. Esto permite a los clientes que se pongan con un nombre de bloqueo dado a usar el nombre para realizar advertencias de bloqueo cooperativo. Pero tenga en cuenta que esto permite a un cliente que no esté entre el conjunto de clientes cooperativos bloquear un nombre, de forma deliverada o no, y evitar que ninguno de los clientes cooperativos puedan bloquear dicho nombre. Una forma de reducir la probabilidad que esto pase es usar nombres de bloqueo especificos de bases de datos o de aplicación. Por ejemplo, use nombres de bloqueo de la forma *`db_name.str`* o *`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
```

  Tenga en cuenta que la segunda llamada a `RELEASE_LOCK()` retorna `NULL` debido a que el bloqueo `'lock1'` se libera automáticamente por la segunda llamada `GET_LOCK()` .

- `INET_ATON(expr)`

  Dada la representación de cuatros números separados por puntos de una dirección de red como cadena de caracteres, retorna un entero que representa el valor numérico de la dirección. Las direcciones pueden ser direcciones de 4 o 8 bytes .

```
mysql> SELECT INET_ATON('209.207.224.40');
        -> 3520061480
```

  El número generado siempre tiene orden de bytes de red. Para el ejemplo mostrado anteriormente, el número se calcula como `209*256^3 + 207*256^2 + 224*256 + 40`.

  `INET_ATON()` también entiende direcciones IP de forma corta:

```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
        -> 2130706433, 2130706433
```

  NOTA: Cuando almacene valores generados por `INET_ATON()`, se recomiendoa que use una columna `INT UNSIGNED`. Si usa una columna `INT` (con signo), los valores correspondientes a direcciones IP

para las que el primer octeto es mayor que 127 se truncan a 2147483647 (esto es, el valor retornado por `INET_ATON('127.255.255.255')`). Consulte Sección 11.2, “Tipos numéricos”.

- `INET_NTOA(expr)`

  Dada una dirección de red numérica (4 o 8 bytes), retorna la representación de puntos de la dirección como cadena.

```
mysql> SELECT INET_NTOA(3520061480);
        -> '209.207.224.40'
```

- `IS_FREE_LOCK(str)`

  Chequea si el nombre de bloqueo *`str`* está libre para uso (esto es, no bloqueado). Retorna `1` si el bloqueo está libre (nadie lo usa para bloquear), `0` si el bloqueo está en uso, y `NULL` en errores (tales como argumentos incorrectos).

- `IS_USED_LOCK(str)`

  Chequea si el nombre de bloqueo *`str`* está en uso (esto es, bloqueado). Si es así, retorna el identificador de conexión del cliente que tiene el bloqueo. De otro modo, retorna `NULL`.

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

  Esta función es útil para control de sincronización de maestro/ esclavo. Bloquea hasta que el esclavo ha leído y aplicado todas las actualizaciones hasta la posición especificada en el log del maestro. El valor retornado es el número de eventos logueados que tiene que esperar para llegar a la posición especificada. La función retorna `NULL` si el flujo esclavo SQL no está arrancado, la información maestra del esclavo no está inicializada, los argumentos son incorrectos, u ocurre un error. Retorna `-1` si se agota el tiempo de espera. Si el flujo SQL esclavo para mientras `MASTER_POS_WAIT()` está esperando, la función retorna `NULL`. Si el eslavo pasa la posición especificada, la función retorna inmediatamente.

  Si un valor *`timeout`* se especifica, `MASTER_POS_WAIT()` para de esprar cuando pasan *`timeout`* segundos. *`timeout`* debe ser superior a 0; un cero o *`timeout`* negativo significa que no hay timeout.

- `RELEASE_LOCK(str)`

  Libera el bloqueo nombrado por la cadena *`str`* obtenida con `GET_LOCK()`. Retorna `1` si el bloqueo se libera, `0` si el bloqueo no estaba bloqueado por este flujo (en cuyo caso el bloqueo no se libera), y `NULL` si el bloqueo nombrado no existía. El bloqueo no existe si nunca se obtuvo por una llamada a `GET_LOCK()` o si había sido liberado préviamente.

  El comando `DO` es conveniente para usar con `RELEASE_LOCK()`. Consulte Sección 13.2.2, “Sintaxis de `DO`”.

- `UUID()`

  Retorna un Universal Unique Identifier (UUID) (Identificador Único Universal) generado según la “DCE 1.1: Remote Procedure Call” (Apéndice A) CAE (Common Applications Environment) Especificaciones publicadas por The Open Group en Octubre 1997 (Número de Documento C706).

  Se designa un UUID como número que es único globalmente en espacio y tiempo. Dos llamadas a `UUID()` generan dos valores distintos, incluso si estas llamadas se realizan en dos máquinas separadas y no están conectadas entre ellas.

  Un UUID es un número de 128 bits representado por una cadena de cinco números hexadecimales en formato `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee`:

  - Los primeros tres números se generan de un valor temporal.
  - El cuarto número preserva unicidad temporal en caso de que el valor temporal pierda moniticidad (por ejemplo, debido al cambio horario).
  - El quinto número es un número IEEE 802 de nodo que proporciona unicidad espacial. Un número aleatorio se sustituye si el último no está disponible (por ejemplo, debido a que la máquina no tenga tarjeta Ethernet, o no sabemos encontrar la dirección hardware de una interfaz en el sistema operativo). En este caso, la unicidad espacial no se puede garantizar. Sin embargo, una colisión debe tener una probabilidad *muy* baja.

    Actualmente, la dirección MAC de una interfaz se tiene en cuenta sólo en FreeBSD y Linux. En otros sistemas operativos, MySQL usa un número generado aleatoriamente de 48 bits.

```
mysql> SELECT UUID();
        -> '6ccd780c-baba-1026-9564-0040f4311e29'
```

  Tenga en cuenta que `UUID()` no funciona todavía con replicación.

12.10. Funciones y modificadores para cláusulas GROUP BY

12.10.1. Funciones (de agregación) de GROUP BY

Si usa una función de grupo en un comando sin la cláusula `GROUP BY` , es equivalente a agrupar todos los registros.

- `AVG([DISTINCT] expr)`

  Retorna el valor medio de *`expr`*. La opción `DISTINCT` puede usarse desde MySQL 5.0.3 para retornar la media de los valores distintos de *`expr`*.

```
mysql> SELECT student_name, AVG(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

- `BIT_AND(expr)`

  Retorna el `AND` bit a bit de todos los bits en *`expr`*. Los cálculos se realizan con precisión de 64 bits (`BIGINT`) .

En MySQL 5.0, esta función retoran 18446744073709551615 si no hubieran registros coindicentes. (Este es el valor de un BIGINT sin signo con todos los bits a 1.)

- BIT_OR(*expr*)

  Retorna la OR bit a bit de todos los bits en *expr*. El cálculo se realiza con precisión de 64 bits (BIGINT) .

  Esta función retorna 0 si no hay registros coincidentes.

- BIT_XOR(*expr*)

  Retorna el XOR bit a bit de todos los bits en *expr*. Los cálculos se realizan con precisión de 64 bits (BIGINT) .

  Esta función retorna 0 si no hay registros coincidentes.

- COUNT(*expr*)

  Retorna el contador del número de valores no NULL en los registros recibidos por un comando SELECT.

```
mysql> SELECT student.student_name,COUNT(*)
    ->        FROM student,course
    ->        WHERE student.student_id=course.student_id
    ->        GROUP BY student_name;
```

  COUNT(*) es algo diferente en que retorna un contador del número de registros retornados, si contienen o no valores NULL .

  COUNT(*) está optimizado para retornar muy rápidamente si SELECTretorna de una tabla, no se retornan otras columnas, y no hay cláusula WHERE . Por ejemplo:

```
mysql> SELECT COUNT(*) FROM student;
```

  Esta optimización se aplica sólo a tablas MyISAM, ya que un conteo exacto de registros se almacena para estos tipos de tablas y puede ser accedido muy rápidamente. Para motores de almacenamiento transaccionales (InnoDB, BDB), almacenar un contador de registros es más problemático ya que pueden ocurrir múltiples transacciones, cada una de las cuales puede afectar al contador.

- COUNT(DISTINCT *expr*,[*expr*...])

  Retorna un contador del número de valores no NULL distintos.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

En MySQL, puede obtener el número de combinaciones de distintas expresiones que no contiene NULL dada una lista de expresiones. En SQL estándar, tendría que hacer una concatenación de todas las expresiones dentro de COUNT(DISTINCT ...).

- GROUP_CONCAT(*expr*)

  Esta función retorna una cadena resultado con los valores no NULL concatenados de un grupo. Retorna NULL si no hay valores no NULL . La sintaxis completa es la siguiente:

```
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;
```

  En MySQL, puede obtener los valores concatenados de combinaciones de expresiones. Puede eliminar valores duplicados usando DISTINCT. Si quiere ordenar valores en el resultado, debe usar la cláusula ORDER BY. Para ordenar en orden inverso, añada la palabra clave DESC (descendente) al nombre de la columna que está ordenando con la cláusula ORDER BY . El valor por defecto es orden ascendente; puede especificarse explícitamente usando la palabra clave ASC. SEPARATOR tiene a continuación la cadena que debe insertarse entre los valores del resultado. Por defecto es una coma (','). Puede eliminar el separador especificando SEPARATOR ''.

  Puede especificar la longitud máxima con la variable de sistema group_concat_max_len . La sintaxis para ello en tiempo de ejecución es la siguiente, donde val es un entero sin signo:

```
SET [SESSION | GLOBAL] group_concat_max_len = val;
```

  Si se especifica una longitud máxima, el resultado se trunca a su longitudo máxima.

- MIN([DISTINCT] *expr*), MAX([DISTINCT] *expr*)

  Retornas los valores máximos y mínimos de *expr*. MIN() y MAX() pueden tener un argumento; en tales casos retornan el valor de cadena mínimo y máximo. Consulte Sección 7.4.5, “Cómo utiliza MySQL los índices”. La palabra clave DISTINCT puede usarse en MySQL 5.0 para encontrar el mínimo o máximo de los distintos valores de *expr*; esto es soportado, pero produce el mismo resultado que omitiendo DISTINCT.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
```

```
    ->         FROM student
    ->         GROUP BY student_name;
```

Para MIN(), MAX(), y otras funciones agregadas, MySQL compara columnas ENUM y SET por su valor de cadena de caracteres en lugar que por la posición relativa de la cadena en el conjunto. Esto difiere de cómo los compara ORDER BY. Esto se rectificará en una futura versión de MySQL .

- STD(*expr*), STDDEV(*expr*)

  Retorna la desviación estándar de *expr*. Esta es una extensión del estándar SQL. La forma STDDEV() de esta función se proporciona para compatibilidad con Oracle. Desde MySQL 5.0.3, la función estándar SQL STDDEV_POP() puede usarse en su lugar.

- STDDEV_POP(*expr*)

  Retorna la desviación estándar de *expr* (la raíz cuadrada de VAR_POP()). Esta función se añadió en MySQL 5.0.3. Antes de 5.0.3, puede usar STD() o STDDEV(), que son equivalentes pero no SQL estándar.

- STDDEV_SAMP(*expr*)

  Retorna la muestra de la desviación estándar de *expr* (la raíz cuadrada de VAR_SAMP(). Esta función se añadió en MySQL 5.0.3.

- SUM([DISTINCT] *expr*)

  Retorna la suma de *expr*. Si el conjunto resultado no tiene registros, SUM() retorna NULL. La palabra clave DISTINCT puede usarse en MySQL 5.0 para sumar sólo los valores distintos de *expr*.

- VAR_POP(*expr*)

  Retorna la varianza estándar de *expr*. Considera los registros como la población completa, no como una muestra, así que tiene el número de registros como denominador. Esta función se añadió en MySQL 5.0.3. Antes de 5.0.3, puede usar VARIANCE(), que es equivalente pero no SQL estándar.

- VAR_SAMP(*expr*)

  Retorna la varianza de muestra de *expr*. Esto es, el denominador es el número de registros menos uno. Esta función se añadió en MySQL 5.0.3.

- VARIANCE(*expr*)

  Retorna la varianza estándar de *expr*. Esto es una extensión de SQL estándar. Desde MySQL 5.0.3, la función SQL estándar VAR_POP() puede usarse en su lugar.

12.10.2. Modificadores de GROUP BY

La cláusula GROUP BY permite añadir un modificador WITH ROLLUP que provoca añadir registros extra al resumen de la salida. Estos registros representan operaciones de resumen de alto nivel (o super agregadas) . ROLLUP por lo tanto le permite responder preguntas en múltiples niveles de análisis con una sola consulta. Puede usarse, por ejemplo, para proporcionar soporte para operaciones OLAP (Online Analytical Processing).

Suponga que una tabla llamada sales tiene las columnas year, country, product, y profit para guardar las ventas productivas:

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

Los contenidos de la tabla pueden resumirse por año con un simple GROUP BY como este:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

Esta salida muestra el beneficio total para cada año, pero si quiere determinar el beneficio total registrado durante todos los años, debe añadir los valores individuales usted mismo o ejecutar una consulta adicional.

O puede usar ROLLUP, que proporciona ambos niveles de análisis con una única consulta. Añadir un modificador WITH ROLLUP a la cláusula GROUP BY provoca que la consulta produzca otro registro que muestra el beneficio total sobre todos los valores de año:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

La línea super agregada con la suma total se identifica con el valor NULL en la columna year .

ROLLUP tiene un efecto más complejo cuando hay múltiples columnas GROUP BY . En este caso, cada vez que hay un “break” (cambio en el valor) en cualquiera excepto la última columna de agrupación, la consulta produce registros super agregados extra.

Por ejemplo, sin ROLLUP, un resumen de la tabla sales basado en year, country, y product puede tener este aspecto:

```
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 |
+------+---------+------------+-------------+
```

La salida indica valores resumen sólo en el nivel de análisis year/country/product . Cuando se añade `ROLLUP` , la consulta produce registros extra:

```
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 |
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

Para esta consulta, añadir `ROLLUP` provoca que la salida incluya información resumen en cuatro niveles de análisis, no sólo uno. Aquí se muestra cómo interpretar la salida de `ROLLUP` :

- A continuación de cada conjunto de registros producto de un año dado y un país, un registro resume extra se produce mostrando el total para todos los productos. Estos registros tienen la columna `product` a `NULL`.
- A continuación de cada conjunto de registros para un año dado, se produce un registro resumen extra mostrando el total para todos los países y productos. Estos registros tienen las columnas `country` y `products` a `NULL`.
- Finalmente, a continuación de todos los otros registros, un registro extra resumen se produce mostrando el total para todos los años, paises y productos. Este registro tiene las columnas `year`, `country`, y `products` a `NULL`.

Otras consideraciones usando `ROLLUP`

Los siguientes puntos listan algunos comportamientos específicos a la implementación de MySQL de `ROLLUP`:

Cuando usa ROLLUP, no puede usar una cláusula ORDER BY para ordenar los resultados. En otras palabras, ROLLUP y ORDER BY son mútuamente exclusivas. Sin embargo, puede tener algún control sobre la ordenación. GROUP BY en MySQL ordena los resultados, y puede usar explícitamente ASC y DESC con columnas mostradas en la lista GROUP BY para especificar orden de ordenación para columnas individuales. (Los registros resumen de alto nivel apadidos por ROLLUP todavía aparecen tras los registros para los que son calculados, a pesar del orden de ordenación.)

LIMIT puede usarse para restringir el número de registros retornados al cliente. LIMIT se aplica tras ROLLUP, así que el límite se aplica contra los registros extra añadidos por ROLLUP. Por ejemplo:

```
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 |
+------+---------+------------+-------------+
```

Usar LIMIT con ROLLUP puede producir resultados que son más difíciles de interpretar, ya que tiene menos contexto para entender los registros super agregados.

Los indicadores NULL en cada registro super agregado se preducen cuando los registros se envían al cliente. El servidor busca las columnas llamadas en la cláusula GROUP BY siguiendo la que esté más a la izquierda que ha cambiado un valor. Para cualquier columna en el conjunto de resultados con un nombre que sea una coincidencia léxica para cualquiera de estos nombres, su valor se cambia a NULL. (Si especifica columnas para agrupar con número de columna, el servidor identifica tales columnas para cambiar a NULL por el número.)

Debido a que los valores NULL en los registros super agregados se guardan en el conjunto de resultados en una de las últimas etapas del proceso de la consulta, no puede testearlas como valores NULL dentro de la propia consulta. Por ejemplo, no puede añadir HAVING product IS NULL a la consulta para eliminar de la salida todos los valores menos los registros super agregados.

Por otro lado, los valores NULL aparecen como NULL en la parte del cliente y pueden testearse como tales usando cualquier interfaz de programación de cliente MySQL.

12.10.3. GROUP BY con campos escondidos

MySQL extiende el uso de GROUP BY para que pueda usar columnas o cálculos en la lista SELECT que no aparecen en la cláusula GROUP BY. Esto se aplica a *cualquier valor posible para este grupo* . Puede usarlo para obtener mejor rendimiento al evitar ordenar y agrupar elementos innecesarios. Por ejemplo, no necesita agrupar customer.name en la siguiente consulta:

```
mysql> SELECT order.custid, customer.name, MAX(payments)
    ->        FROM order,customer
    ->        WHERE order.custid = customer.custid
    ->        GROUP BY order.custid;
```

En SQL estándar, puede tener que añadir customer.name a la cláusula GROUP BY . En MySQL, el nombre es redundante si no se ejecuta en modo ANSI.

No use esta característica si las columnas que omite de la parte `GROUP BY` no son únicos en el grupo! Obtendría resultados impredecibles.

En algunos casos, puede usar `MIN()` y `MAX()` para obtener valores específicos de columna incluso si no son únicos. La siguiente da el valor de `column` del registro conteniendo el valor más pequeño en la columna `sort` :

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)
```

Consulte Sección 3.6.4, “Los registros de un grupo que tienen el máximo valor en alguna columna”.

Tenga en cuenta que si trata de seguir SQL estándar, no puede usar expresiones en clásulas `GROUP BY` o `ORDER BY` . Puede solucionar esta limitación usando un alias para la expresión:

```
mysql> SELECT id,FLOOR(value/100) AS val
    -> FROM tbl_name
    -> GROUP BY id, val ORDER BY val;
```

Sin embargo, MySQL le permite usar expresiones en cláusulas `GROUP BY` y `ORDER BY` . Por ejemplo:

```
mysql> SELECT id, FLOOR(value/100) FROM tbl_name ORDER BY RAND();
```

Capítulo 13. Sintaxis de sentencias SQL

Tabla de contenidos

Este capítulo describe la sintaxis para los comandos SQL soportados en MySQL.

13.1. Sentencias de definición de datos (Data Definition Statements)

13.1.1. Sintaxis de ALTER DATABASE

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification [, alter_specification] ...

alter_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

ALTER DATABASE le permite cambiar las características globales de una base de datos. Estas características se almacenan en el fichero db.opt en el directorio de la base de datos. Para usar ALTER DATABASE, necesita el permiso ALTER en la base de datos.

La cláusula CHARACTER SET cambia el conjunto de caracteres por defecto de la base de datos. La cláusula COLLATE cambia la colación por defecto de la base de datos. El conjunto de caracteres y la colación se discuten en Capítulo 10, *Soporte de conjuntos de caracteres*.

En MySQL 5.0, el nombre de base de datos puede omitirse. El comando se aplica a la base de datos por defecto. ALTER SCHEMA puede usarse desde MySQL 5.0.2.

13.1.2. Sintaxis de ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
  | ADD INDEX [index_name] [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [index_name] [index_type] (index_col_name,...)
  | ADD [FULLTEXT|SPATIAL] [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | table_options
```

ALTER TABLE le permite cambiar la estructura de una tabla existente. Por ejemplo, puede añadir o borrar columnas, crear o destruir índices, cambiar el tipo de columnas existentes, o renombrar columnas o la misma tabla. Puede cambiar el comentario de la tabla y su tipo.

La sintaxis para varias de las alteraciones permitidas es similar a cláusulas del comando CREATE TABLE. Esto incluye modificaciones *table_options* , para opciones tales como ENGINE, AUTO_INCREMENT, y AVG_ROW_LENGTH. Consulte Sección 13.1.5, “Sintaxis de CREATE TABLE”.

Algunas operaciones pueden producir advertencias si se intentan en una tabla para que el motor de almacenamiento no soporte la operación. Estas advertencias pueden mostrarse con SHOW WARNINGS. Consulte Sección 13.5.4.22, “Sintaxis de SHOW WARNINGS”.

Si usa ALTER TABLE para cambiar la especificación de una columna pero DESCRIBE *tbl_name* indica que la columna no ha cambiado, es posible que MySQL haya ignorado las modificaciones por alguna de las razones descritas en Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”. Por ejemplo, si intenta cambiar una columna VARCHAR a CHAR, MySQL usa VARCHAR si la tabla contiene otras columnas de longitud variable.

ALTER TABLE funciona creando una copia temporal de la tabla original. La alteración se realiza en la copia, luego la tabla original se borra y se renombra la nueva. Mientras se ejecuta ALTER TABLE la tabla original es legible por otros clientes. Las actualizaciones y escrituras en la tabla se esperan hasta que la nueva tabla esté lista, luego se redirigen automáticamente a la nueva tabla sin ninguna actualización fallida.

Tenga en cuenta que si usa cualquier otra opción en ALTER TABLE distinta a RENAME, MySQL siempre crea una tabla temporal, incluso si los datos no necesitan ser copiados (tales como cuando cambia el nombre de una columna). Planeamos arreglar esto en el futuro, pero debido a que ALTER TABLE no es un comando que se use frecuentemente, no es un tema demasiado urgente. Para tablas MyISAM puede incrementar la velocidad de la operación de recrear índices (que es la parte más lenta del proceso de alteración) mediante la variable de sistema myisam_sort_buffer_size poniendo un valor alto.

- Para usar ALTER TABLE, necesita ALTER, INSERT, y permisos CREATE para la tabla.
- IGNORE es una extensión MySQL a SQL estándar. Controla cómo funciona ALTER TABLE si hay duplicados en las claves primarias en la nueva tabla o si ocuren advertencias cuando está activo el modo STRICT. Si no se especifica IGNORE la copia se aborta y no se ejecuta si hay errores de clave duplicada. Si se especifica IGNORE , entonces para duplicados con clave única, sólo se usa el primer registro. El resto de registros conflicitivos se borran. Los valores erróneos se truncan al valor más cercano aceptable.
- Puede ejecutar múltiples cláusulas ADD, ALTER, DROP, y CHANGE en un único comando ALTER TABLE . Esta es una extensión MySQL al estándar SQL, que permite sólo una de cada cláusula por comando ALTER TABLE . Por ejemplo, para borrar múltiples columnas en un único comando:

  ```
  mysql> ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
  ```

- CHANGE *col_name*, DROP *col_name*, y DROP INDEX son extensiones MySQL al estándar SQL.
- MODIFY es una extensión de Oracle a ALTER TABLE.
- La palabra COLUMN es opcional y puede omitirse.
- Si usa ALTER TABLE *tbl_name* RENAME TO *new_tbl_name* sin ninguna otra opción, MySQL símplemente renombra cualquier fichero que se corresponda a la tabla *tbl_name*. No es necesario crear una tabla temporal. (Puede usar el comando RENAME TABLE para renombrar tablas. Consulte Sección 13.1.9, “Sintaxis de RENAME TABLE”.)
- Las cláusulas *column_definition* usan la misma sintaxis para ADD y CHANGE así como CREATE TABLE. Tenga en cuenta que esta sintaxis incluye el nombre de la columna, no sólo el tipo. Consulte Sección 13.1.5, “Sintaxis de CREATE TABLE”.
- Puede renombrar una columna usando CHANGE *old_col_name column_definition*. Para ello, especifique el nombre de columna viejo y nuevo y el tipo de la columna actual. Por ejemplo, para renombrar una columna INTEGER de a a b, puede hacer:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

Si quiere cambiar el tipo de una columna pero no el nombre, la sintaxis CHANGE necesita un nombre viejo y nuevo de columna, incluso si son iguales. Por ejemplo:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

Puede usar MODIFY para cambiar el tipo de una columna sin renombrarla:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- Si usa CHANGE o MODIFY para acortar una columna para la que existe un índice en la columna, y la longitud de la columna resultante es menor que la del índice, MySQL reduce el índice automáticamente.
- Cuando cambia un tipo de columna usando CHANGE o MODIFY, MySQL intenta convertir valores de columna existentes al nuevo tipo lo mejor posible.
- En MySQL 5.0, puede usar FIRST o AFTER *col_name* para añadir una columna a una posición específica sin un registro de tabla. Por defecto se añade al final. Puede usar FIRST y AFTER en operaciones CHANGE o MODIFY en MySQL 5.0.
- ALTER COLUMN especifica un nuevo valor por defecto para una columna o borra el antiguo valor por defecto. Si el antiguo valor por defecto se borra y la columna puede ser NULL, el nuevo valor por defecto es NULL. Si la columna no puede ser NULL, MySQL asigna un valor por defecto, como se describe en Sección 13.1.5, “Sintaxis de CREATE TABLE”.
- DROP INDEX borra un índice. Es una extensión MySQL al estándar SQL. Consulte Sección 13.1.7, “Sintaxis de DROP INDEX”.
- Si las columnas se borran de una tabla, las columnas también se borran de cualquier índice del que formaran parte. Si todas las columnas que crean un índice se borran, también se borra el índice.
- Si una tabla contiene sólo una columna, la columna no puede borrarse. Si lo que quiere es borrar la tabla, use DROP TABLE .
-
- ORDER BY le permite crear la nueva tabla con los registros en un orden específico. Tenga en cuenta que la tabla no queda en este orden tras las inserciones y borrados. Esta opción es útil cuando sabe que normalmente consultará los registros en el mismo orden; usando esta opción tras grandes cambios en la tabla, puede ser capaz de obtener un mejor rendimiento. En algunos casos, puede hacer la ordenación más fácil para MySQL si la tabla está en el orden de la columna por la que quiere ordenar posteriormente.
- Si usa ALTER TABLE en una tabla MyISAM , todos los índices no únicos se crean en un batch separado (como para REPAIR TABLE). Esto debe hacer ALTER TABLE mucho más rápido cuando tiene muchos índices.

  En MySQL 5.0, esta característica puede activarse explícitamente ALTER TABLE ... DISABLE KEYS le dice a MySQL que pare de actualizar índices no únicos para una tabla MyISAM . ALTER TABLE ... ENABLE KEYS debe usarse para recrear índices perdidos. MySQL lo hace con un algoritmo especial que es mucho más rápido que insertar claves una a una, así que deshabilitar claves antes de realizar operaciones de inserción masivas debería dar una mejora de velocidad. Usar ALTER TABLE ... DISABLE KEYS requiere del permiso INDEX además de los permisos mencionados anteriormente.
-

-
- ALTER TABLE ignora las opciones DATA DIRECTORY y INDEX DIRECTORY.
- Si quiere cambiar el conjunto de caracteres por defecto de la tabla y todas las columnas de caracteres (CHAR, VARCHAR, TEXT) a un nuevo conjunto de caracteres, use un comando como:

  ```
  ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
  ```

  Atención: La operación precedente convierte los valores de columnas entre conjuntos de caracteres. Esto *no* es lo que quiere hacer si tiene una columna en un conjunto de caracteres (como latin1) pero los valores almacenados realmente usan otro conjunto de caracteres incompatible (como utf8). En este caso, tiene que hacer lo siguiente para cada una de tales columnas:

  ```
  ALTER TABLE t1 CHANGE c1 c1 BLOB;
  ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
  ```

  La razón de que esto funcione es que no hay conversión cuando convierte desde o hacia columnas BLOB .

  Si especifica CONVERT TO CHARACTER SET binary, las columnas CHAR, VARCHAR, y TEXT se convierten a sus cadenas de caracteres binarias (BINARY, VARBINARY, BLOB). Esto significa que las columnas no tendrán un conjunto de carácters y que siguientes operaciones CONVERT TO no se les aplicarán.

  Para sólo cambiar el conjunto de caracteres *por defecto* de una tabla, use este comando:

  ```
  ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
  ```

  La palabra DEFAULT es opcional. El conjunto de caracteres por defecto es el que se usa si no especifica uno para una nueva columna que añada a la tabla (por ejemplo, con ALTER TABLE ... ADD column).

  Atención: En MySQL 5.0, ALTER TABLE ... DEFAULT CHARACTER SET y ALTER TABLE ... CHARACTER SET son equivalentes y cambian sólo el conjunto de caracteres por defecto de la tabla.
-
- Con la función mysql_info() de la API de C, puede consultar el número de registros copiados, y (cuando se usa IGNORE) cuántos registros se borraron debido a duplicación de valores de claves única. Consulte Sección 24.2.3.32, “mysql_info()”.

Hay algunos ejemplos que muestran usos de ALTER TABLE. Comienza con una tabla t1 que se crea como se muestra:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

Para renombrar la tabla de t1 a t2:

```
mysql> ALTER TABLE t1 RENAME t2;
```

Para cambiar la columna a desde INTEGER a TINYINT NOT NULL (dejando el mismo nombre), y para cambiar la columna b desde CHAR(10) a CHAR(20) así como dejarla de b a c:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

Para añadir una nueva columna TIMESTAMP llamada d:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

Para añadir índices en las columnas d y a:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

Para borrar la columna c:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

Para añadir una nueva columna entera AUTO_INCREMENT llamada c:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ->     ADD PRIMARY KEY (c);
```

Tenga en cuenta que indexamos c (como PRIMARY KEY), ya que las columnas AUTO_INCREMENT deben indexarse, y también que declaramos c como NOT NULL, ya que las columnas de clave primara no pueden ser NULL.

Cuando añade una columna AUTO_INCREMENT los valores se rellenan con números secuenciales automáticamente. Para tablas MyISAM puede asignar el primer número de secuencia ejecutando SET INSERT_ID=*value* antes de ALTER TABLE o usando la opción de tabla AUTO_INCREMENT=*value*. Consulte Sección 13.5.3, “Sintaxis de SET”.

Desde MySQL 5.0.3, puede usar la opción de tabla ALTER TABLE ... AUTO_INCREMENT=*value* para InnoDB para asignar el número de secuencia de nuevos registros si el valor es mayor que el máximo valor en la columna AUTO_INCREMENT . *Si el valor es menor que el máximo actual en la columna, no se da ningún mensaje de error y el valor de secuencia actual no se cambia.*

Con tablas MyISAM , si no cambia la columna AUTO_INCREMENT , el número de secuencia no se ve afectado. Si elimina una columna AUTO_INCREMENT y luego añade otra columna AUTO_INCREMENT los números se resecuencian comenzando en 1.

Consulte Sección A.7.1, “Problemas con ALTER TABLE”.

13.1.3. Sintaxis de CREATE DATABASE

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification [, create_specification] ...]

create_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

CREATE DATABASE crea una base de datos con el nombre dado. Para usar CREATE DATABASE, necesita el permiso CREATE en la base de datos.

Las reglas para nombres de bases de datos permitidos se dan en Sección 9.2, “Nombres de bases de datos, tablas, índices, columnas y alias”. Ocurre un error si la base de datos existe y no especifica IF NOT EXISTS.

En MySQL 5.0, las opciones create_specification pueden darse para especificar característica de la base de datos. Las características se almacenan en el fichero db.opt en el directorio de la base de datos. La cláusula CHARACTER SET especifica el conjunto de caracteres por defecto de la base de datos. La cláusula COLLATE especifica la colación por defecto de la base de datos. Los nombres de colación y de conjunto de caracteres se discuten en Capítulo 10, *Soporte de conjuntos de caracteres*.

Las bases de datos en MySQL se implementan como directorios que contienen ficheros que se corresponden a tablas en la base de datos. Como no hay tablas en la base de datos cuando se crean inicialmente, el comando CREATE DATABASE en MySQL 5.0 crea sólo un directorio bajo el directorio de datos de MySQL y el fichero db.opt file.

Si crea manualmente un directorio bajo el directorio de datos (por ejemplo, con mkdir), el servidor lo considera como un directorio de base de datos y muestra la salida de SHOW DATABASES.

CREATE SCHEMA puede usarse desde MySQL 5.0.2.

También puede usar el programa mysqladmin para crear bases de datos. Consulte Sección 8.4, "Administrar un servidor MySQL con mysqladmin".

13.1.4. Sintaxis de CREATE INDEX

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [USING index_type]
    ON tbl_name (index_col_name,...)

index_col_name:
    col_name [(length)] [ASC | DESC]
```

En MySQL 5.0, CREATE INDEX se mapea a un comando ALTER TABLE para crear índices. Consulte Sección 13.1.2, "Sintaxis de ALTER TABLE".

Normalmente, crea todos los índices en una tabla cuando se crea la propia tabla con CREATE TABLE. Consulte Sección 13.1.5, "Sintaxis de CREATE TABLE". CREATE INDEX le permite añadir índices a tablas existentes.

Una lista de columnas de la forma (col1,col2,...) crea un índice de múltiples columnas. Los valores de índice se forman al concatenar los valores de las columnas dadas.

Para columnas CHAR y VARCHAR, los índices pueden crearse para que usen sólo parte de una columna, usando *col_name*(*length*) para indexar un prefijo consistente en los primeros *length* caracteres de cada valor de la columna. BLOB t TEXT pueden indexarse, pero se *debe* dar una longitud de prefijo.

El comando mostrado aquí crea un índice usando los primeros 10 caracteres de la columna name :

```
CREATE INDEX part_of_name ON customer (name(10));
```

Como la mayoría de nombres usualmente difieren en los primeros 10 caracteres, este índice no debería ser mucho más lento que un índice creado con la columna name entera. Además, usar columnas parcialmente para índices puede hacer un fichero índice mucho menor, que puede ahorrar mucho espacio de disco y además acelarar las operaciones INSERT .

Los prefijos pueden tener una longitud de hasta 255 bytes. Para tablas MyISAM y InnoDB en MySQL 5.0, pueden tener una longitud de hasta 1000 bytes . Tenga en cuenta que los límites de los prefijos se miden en bytes, mientras que la longitud de prefijo en comandos CREATE INDEX se interpreta como el número

de caracteres. Tenga esto en cuenta cuando especifique una longitud de prefijo para una columna que use un conjunto de caracteres de múltiples bytes.

En MySQL 5.0:

- Puede añadir un índice en una columna que puede tener valores NULL sólo si está usando MyISAM, InnoDB, o BDB .
- Puede añadir un índice en una columna BLOB o TEXT sólo si está usando el tipo de tabla MyISAM, BDB, o InnoDB .

Una especificación *index_col_name* puede acabar con ASC o DESC. Estas palabras se permiten para extensiones futuras para especificar almacenamiento de índice ascendente o descendente. Actualmente se parsean pero se ignoran; los valores de índice siempre se almacenan en orden ascendente.

En MySQL 5.0, algunos motores le permiten especificar un tipo de índice cuando se crea un índice. La sintaxis para el especificador *index_type* es USING *type_name*. Los valores *type_name* posibles soportados por distintos motores se muestran en la siguiente tabla. Donde se muestran múltiples tipos de índice , el primero es el tipo por defecto cuando no se especifica *index_type* .

Motor de almacenamiento	Tipos de índice permitidos
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

Ejemplo:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index USING BTREE ON lookup (id);
```

TYPE *type_name* puede usarse como sinónimo de USING *type_name* para especificar un tipo de índice. Sin embargo, USING es la forma preferida. Además, el nombre de índice que precede el tipo de índice en la especificación de la sintaxis de índice no es opcional con TYPE. Esto es debido a que, en contra de USING, TYPE no es una palabra reservada y se interpreta como nombre de índice.

Si especifica un tipo de índice que no es legal para un motor de almacenamiento, pero hay otro tipo de índice disponible que puede usar el motor sin afectar los resultados de la consulta, el motor usa el tipo disponible.

Para más información sobre cómo MySQL usa índices, consulte Sección 7.4.5, “Cómo utiliza MySQL los índices”.

Índices FULLTEXT en MySQL 5.0 puede indexar sólo columnas CHAR, VARCHAR, y TEXT , y sólo en tablas MyISAM . Consulte Sección 12.7, “Funciones de búsqueda de texto completo (Full-Text)”.

Índices SPATIAL en MySQL 5.0 puede indexar sólo columnas espaciales, y sólo en tablas MyISAM . Los tipo de columna espaciales se describen en Capítulo 18, *Extensiones espaciales de MySQL*.

13.1.5. Sintaxis de CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
```

```
    [table_options] [select_statement]
```

O:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [() LIKE old_tbl_name ()];

create_definition:
    column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  | KEY [index_name] [index_type] (index_col_name,...)
  | INDEX [index_name] [index_type] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX]
        [index_name] [index_type] (index_col_name,...)
  | [FULLTEXT|SPATIAL] [INDEX] [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (index_col_name,...) [reference_definition]
  | CHECK (expr)

column_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value]
        [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
        [COMMENT 'string'] [reference_definition]

type:
    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
  | CHAR(length) [BINARY | ASCII | UNICODE]
  | VARCHAR(length) [BINARY]
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT [BINARY]
  | TEXT [BINARY]
  | MEDIUMTEXT [BINARY]
  | LONGTEXT [BINARY]
  | ENUM(value1,value2,value3,...)
  | SET(value1,value2,value3,...)
  | spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]

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 [COLLATE collation_name]
  | CHECKSUM = {0 | 1}
  | COMMENT = 'string'
  | MAX_ROWS = value
  | MIN_ROWS = value
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'string'
  | DELAY_KEY_WRITE = {0 | 1}
  | ROW_FORMAT = {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | RAID_TYPE = { 1 | STRIPED | RAID0 }
        RAID_CHUNKS = value
        RAID_CHUNKSIZE = value
  | UNION = (tbl_name[,tbl_name]...)
  | INSERT_METHOD = { NO | FIRST | LAST }
  | DATA DIRECTORY = 'absolute path to directory'
  | INDEX DIRECTORY = 'absolute path to directory'

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

`CREATE TABLE` crea una tabla con el nombre dado. Debe tener el permiso `CREATE` para la tabla.

Las reglas para nombres de tabla permitidos se dan en Sección 9.2, “Nombres de bases de datos, tablas, índices, columnas y alias”. Por defecto, la tabla se crea en la base de datos actual. Ocurre un error si la tabla existe, si no hay base de datos actual o si la base de datos no existe.

En MySQL 5.0, el nombre de tabla puede especificarse como *`db_name.tbl_name`* para crear la tabla en la base de datos específica. Esto funciona haya una base de datos actual o no. Si usa identificadores entre comillas, entrecomille el nombre de base de datos y de tabla por separado. Por ejemplo, `` `mydb`.`mytbl` `` es legal, pero `` `mydb.mytbl` `` no.

Puede usar la palabra `TEMPORARY` al crear una tabla. Una tabla `TEMPORARY` es visible sólo para la conexión actual, y se borra automáticamente cuando la conexión se cierra. Esto significa que dos conexiones distintas pueden usar el mismo nombre de tabla temporal sin entrar en conflicto entre ellas ni con tablas no `TEMPORARY` con el mismo nombre. (La tabla existente se oculta hasta que se borra la tabla temporal.) En MySQL 5.0, debe tener el permiso `CREATE TEMPORARY TABLES` para crear tablas temporales.

MySQL 5.0 soporta las palabras `IF NOT EXISTS` para que no ocurra un error si la tabla existe. Tenga en cuenta que no hay verificación que la tabla existente tenga una estructura idéntica a la indicada por el comando `CREATE TABLE` . *Nota*: Si usa `IF NOT EXISTS` en un comando `CREATE TABLE ... SELECT` ,cualquier registro seleccionado por la parte `SELECT` se inserta si la tabla existe o no.

MySQL representa cada tabla mediante un fichero `.frm` de formato de tabla (definición) en el directorio de base de datos. El motor para la tabla puede crear otros ficheros también. En el caso de tablas `MyISAM` , el motor crea ficheros índice y de datos. Por lo tanto, para cada tabla `MyISAM` *`tbl_name`*, hay tres ficheros de disco:

Fichero	Propósito
`tbl_name``.frm`	Fichero de formato de tabla (definición)
`tbl_name``.MYD`	Fichero de datos
`tbl_name``.MYI`	Fichero índice

Los ficheros creados por cada motor de almacenamiento para representar tablas se describen en Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

Para información general de las propiedades de los diversos tipos de columna, consulte Capítulo 11, *Tipos de columna*. Para información acerca de tipos de columna espaciales, consulte Capítulo 18, *Extensiones espaciales de MySQL*.

- Si no se especifica NULL ni NOT NULL, la columna se trata como si se especificara NULL .
- Una columna entera puede tener el atributo adicional AUTO_INCREMENT. Cuando inserta un valor de NULL (recomendado) o 0 en una columna AUTO_INCREMENT autoindexada, la columna se asigna al siguiente valor de secuencia. Típicamente esto es *value*+1, donde *value* es el mayor valor posible para la columna en la tabla. Secuencias AUTO_INCREMENT comienzan con 1. Tales columnas deben definirse como uno de los tipos enteros como se describe en Sección 11.1.1, “Panorámica de tipos numéricos”. (El valor 1.0 **no** es un entero.) Consulte Sección 24.2.3.34, “mysql_insert_id()”.

  En MySQL 5.0, especificar NO_AUTO_VALUE_ON_ZERO para la opción de servidor --sql-mode o la variable de sistema sql_mode le permite almacenar 0 en columnas AUTO_INCREMENT como 0 sin generar un nuevo valor de secuencia. Consulte Sección 5.3.1, “Opciones del comando mysqld”.

  Nota: Sólo puede haber una columna AUTO_INCREMENT por tabla, debe estar indexada, y no puede tener un valor DEFAULT . Una columna AUTO_INCREMENT funciona correctamente sólo si contiene sólo valores positivos. Insertar un número negativo se trata como insertar un número positivo muy grande. Esto se hace para evitar problemas de precisión cuando los números “cambian” de positivos a negativos y asegura que no obtiene accidentalmente una columna AUTO_INCREMENT que contenga 0.

  Para tablas MyISAM y BDB , puede especificar una columna AUTO_INCREMENT secundaria en una clave de múltiples columnas. Consulte Sección 3.6.9, “Utilización de AUTO_INCREMENT”.

  Para hacer MySQL compatible con otras aplicaciones ODBC , puede encontrar el valor AUTO_INCREMENT para el último registro insertado con la siguiente consulta:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

- En MySQL 5.0, las definiciones de columnas de caracteres puede incluir un atributo CHARACTER SET para especificar el conjunto de caracteres y, opcionalmente, una colación para la columna. Para detalles, consulte Capítulo 10, *Soporte de conjuntos de caracteres*. CHARSET es sinónimo de CHARACTER SET.

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

  MySQL 5.0 interpreta las especificaciones de longitud en definiciones de columna en caracteres. (Algunas versiones anteriores los interpretan en bytes.)

-
-

  Un comentario para una columna puede especificarse en MySQL 5.0 con la opción COMMENT . El comentario se muestra con los comandos SHOW CREATE TABLE y SHOW FULL COLUMNS .

- En MySQL 5.0, el atributo SERIAL puede usarse como un alias para BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.
- KEY normalemente es sinónimo para INDEX. En MySQL 5.0, el atributo clave PRIMARY KEY puede especificarse como KEY cuando se da en una definición de columna. Esto se implementó por compatibilidad con otros sistemas de bases de datos.

- En MySQL, un índice UNIQUE es uno en que todos los valores en el índice deben ser distintos. Ocurre un error si intenta añadir un nuevo registro con una clave que coincida con un registro existente. La excepción es que una columna en el índice puede contener valores NULL , puede contener valores NULL múltiples. Esta excepción no se aplica a tablas BDB , en las que una columna indexada le permita un único NULL.
- Una PRIMARY KEY es una KEY única donde todas las columnas de la clave deben definirse como NOT NULL. Si no se declaran explícitamente como NOT NULL, MySQL las declara implícitamente (y sin decirlo) . Una tabla puede tener sólo una PRIMARY KEY. Si no tiene una PRIMARY KEY y una aplicación pide una PRIMARY KEY en sus tablas, MySQL retorna el primer índice UNIQUE que no tenga columnas NULL como la PRIMARY KEY.
- En la tabla creada, una PRIMARY KEY se guarda en primer lugar, seguida por todos los índices UNIQUE, y luego los índices no únicos. Esto ayuda al optimizador MySQL a priorizar qué indice usar y también detectar más rápido claves UNIQUE duplicadas.
- Una PRIMARY KEY puede ser un índice de múltiples columnas. Sin embargo, no puede crear un índice de múltiples columnas usando el atributo de clave PRIMARY KEY en una especificación de columna. Hacerlo sólo marca la columna como primaria. Debe usar una cláusula PRIMARY KEY(index_col_name, ...) separada.
- Si un índice PRIMARY KEY o UNIQUE consite sólo de una columna que tenga un tipo entero, puede referirse a la columna como _rowid en comandos SELECT.
- En MySQL, el nombre de una PRIMARY KEY es PRIMARY. Para otros índices, si no asigna un nombre, el índice tieen el mismo nombre que la primera columna indexada, con un sufijo opcional (_2, _3, ...) para hacerlo único. Puede ver los nombres de índice para una tabla usando SHOW INDEX FROM *tbl_name*. Consulte Sección 13.5.4.11, “Sintaxis de SHOW INDEX”.
- A partir de MySQL 5.0, algunos motores de almacenamiento le permiten especificar un tipo de índice al crear el índice. Consulte Sección 13.1.4, “Sintaxis de CREATE INDEX”.

  Para más información acerca de cómo usa los índices MySQL, consulte Sección 7.4.5, “Cómo utiliza MySQL los índices”.
-
- Con sintaxis *col_name(length)* en una especificación de índice, puede crear un índice que use sólo los primeros *length* caracteres de una columna CHAR o VARCHAR . Indexar sólo un prefijo de valores de columna como este puede hacer el fichero de índice mucho más pequeño. Consulte Sección 7.4.3, “Índices de columna”.

  En MySQL 5.0, los motores MyISAM y InnoDB soportan indexación en columnas BLOB y TEXT . Al indexar columnas BLOB o TEXT *debe* especificar una longitud de prefijo para el índice. Por ejemplo:

  ```
  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  ```

  En MySQL 5.0, los prefijos pueden tener hasta 1000 bytes de longitud para tablas MyISAM y InnoDB y 255 bytes para otros tipos de tabla. Tenga en cuenta que los límites de prefijo se miden en bytes, mientras que la longitud de prefijo en comandos CREATE TABLE se interpretan como el número de caracteres. Asegúrese de tener esto en cuenta al especificar una longitud de prefijo para una columna que use un conjunto de caracteres multi-byte .
- Una especificación *index_col_name* puede acabar con ASC o DESC. Estas palabras clave se permiten para extensiones futuras para especificar almacenamiento de índices ascendente o descendentemente. Actualmente se parsean pero se ignoran; los valores de índice siempre se almacenan en orden ascendente.

- Cuando usa ORDER BY o GROUP BY en una columna TEXT o BLOB en un SELECT, el servidor ordena los valores usando sólo el número inicial de bytes indicados por la variable de sistema max_sort_length . Consulte Sección 11.4.3, “Los tipos BLOB y TEXT”.

- En MySQL 5.0, puede crear índices especiales FULLTEXT , que se usan para índices full-text . Sólo las tablas MyISAM soportan índices FULLTEXT . Pueden crearse sólo desde columnas CHAR, VARCHAR, y TEXT. La indexación siempre se hace sobre la columna entera; la indexación parcial no se soporta y cualquier longitud de prefijo se ignora. Consulte Sección 12.7, “Funciones de búsqueda de texto completo (Full-Text)” para más detalles.

- En MySQL 5.0, puede crear índices SPATIAL en tipos de columna espaciales. Los tipos espaciales se soportan sólo para tablas MyISAM y las columnas indexadas deben declararse como NOT NULL. Consulte Capítulo 18, *Extensiones espaciales de MySQL*.

- En MySQL 5.0, las tablas InnoDB soportan el chequeo de restricciones de claves foráneas . Consulte Capítulo 15, *El motor de almacenamiento InnoDB*. Tenga en cuenta que la sintaxis FOREIGN KEY en InnoDB es más restrictiva que la sintaxis presentada para el comando CREATE TABLE al inicio de esta sección: las columnas en la tabla referenciada debe siempre nombrarse explícitamente. InnoDB soporta tanto acciones ON DELETE como ON UPDATE en MySQL 5.0. Para la sintaxis precisa, consulte Sección 15.6.4, “Restricciones (constraints) FOREIGN KEY”.

  Para otros motores de almacenamiento, MySQL Server parsea la sintaxis FOREIGN KEY y REFERENCES en comandos CREATE TABLE , pero no hace nada. La cláusula CHECK se parsea paro se ignora en todos los motores de almacenamiento. Consulte Sección 1.7.5.5, “Claves foráneas (foreign keys)”.

- Para tablas MyISAM cada columna NULL ocupa un bit extra, redondeado al byte más próximo. La máxima longitud de registro en bytes puede calcularse como sigue:

```
row length = 1
             + (sum of column lengths)
             + (number of NULL columns + delete_flag + 7)/8
             + (number of variable-length columns)
```

  delete_flag es 1 para tables con formato de registro estático. Las tablas estáticas usan un bit en el registro para un flag que indica si el registro se ha borrado. *delete_flag* es 0 para tablas dinámicas ya que el flag se almacena en una cabecera de registro dinámica.

  Estos cálculos no se aplican en tablas InnoDB , en las que el tamaño de almacenamiento no es distinto para columnas NULL y NOT NULL .

La parte *table_options* de la sintaxis CREATE TABLE puede usarse desde MySQL 3.23.

Las opciones ENGINE y TYPE especifican el motor de almacenamiento para la tabla. ENGINE es el nombre preferido para la opción en MySQL 5.0, y TYPE está obsoleto. El soporte para la palabra TYPE usada en este contexto desaparecerá en MySQL 5.1.

Las opciones ENGINE y TYPE pueden tener los siguientes valores:

Motor de almacenamiento	Descripción
ARCHIVE	El motor de almacenamiento para archivar. Consulte Sección 14.7, “El motor de almacenamiento ARCHIVE”.
BDB	Tablas transaccionales con bloqueo de página. Conocidas como BerkeleyDB. Consulte Sección 14.4, “El motor de almacenamiento BDB (BerkeleyDB)”.

CSV	Tablas que almacenan registros en valores separados por comas. Consulte Sección 14.8, “El motor de almacenamiento CSV”.
EXAMPLE	Motor de ejemplo. Consulte Sección 14.5, “El motor de almacenamiento EXAMPLE”.
FEDERATED	Motor que accede a tablas remotas. Consulte Sección 14.6, “El motor de almacenamiento FEDERATED”.
HEAP	Consulte Sección 14.3, “El motor de almacenamiento MEMORY (HEAP)”.
(*OBSOLETE*) ISAM	No disponible en MySQL 5.0. Si está actualizando a MySQL 5.0 desde una versión prévia, debe convertir cualquier tabla ISAM existente a MyISAM *antes* de la actualización. Consulte Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.
InnoDB	Tablas transaccionales con bloqueo de registro y claves foráneas. Consulte Capítulo 15, *El motor de almacenamiento InnoDB*.
MEMORY	Los datos de este tipo de tabla se almacenan sólo en memoria. (Conocido anteriormente como HEAP.)
MERGE	Colección de tablas MyISAM usadas como una sola tabla. También conocido como MRG_MyISAM. Consulte Sección 14.2, “El motor de almacenamiento MERGE”.
MyISAM	Motor binario portable que es el motor por defecto usado en MySQL. Consulte Sección 14.1, “El motor de almacenamiento MyISAM”.
NDBCLUSTER	Clusterizado, tolerante a errores, tablas en memoria. También conocido como NDB. Consulte Capítulo 16, *MySQL Cluster*.

Para más información acerca de motores MySQL, consulte Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

Si un motor no está disponible, MySQL usa en su lugar MyISAM . Por ejemplo, si una definición de tabla incluye la opción ENGINE=BDB pero el servidor MySQL no soporta tablas BDB , la tabla se crea como MyISAM . Esto hace posible tener un entorno de replicación donde tiene tablas transaccionales en el maestro pero tablas no transaccionales en el esclavo (para tener más velocidad). En MySQL 5.0, aparece una advertencia si la especificación del motor no es correcta.

Las otras opciones de tabla se usan para optimizar el comportamiento de la tabla. En la mayoría de casos, no tiene que especificar ninguna de ellas. La opción funciona para todos los motores a no ser que se indique lo contrario:

- AUTO_INCREMENT

  El valor inicial para AUTO_INCREMENT para la tabla. En MySQL 5.0, sólo funciona para tablas MyISAM y MEMORY. También se soporta para InnoDB desde MySQL 5.0.3. Para inicializar el primer valor de auto incremento para motores que no soporten esta opción, inserte un registro de prueba con un valor que sea uno menor al deseado tras crear la tabla, y luego borre este registro.

  Para motores que soportan la opción de tabla AUTO_INCREMENT en comandos CREATE TABLE puede usar ALTER TABLE *tbl_name* AUTO_INCREMENT = *n* para resetear el valor AUTO_INCREMENT .

- AVG_ROW_LENGTH

  Una aproximación de la longitud media de registro para su tabla. Necesita inicializarla sólo para tablas grandes con registros de longitud variable.

Cuando crea una tabla MyISAM , MySQL usa el producto de las opciones MAX_ROWS y AVG_ROW_LENGTH para decidir el tamaño de la tabla resultante. Si no las especifica, el tamaño máximo para la tabla es 65,536TB de datos (4GB antes de MySQL 5.0.6). (Si su sistema operativo no soporta ficheros de este tamaño, los tamaños de fichero se restringen al límite del sistema operativo.) Si quiere mantener bajos los tamaños de los punteros para que el índice sea pequeño y rápido y no necesita realmente ficheros grandes, puede decrementar el tamaño de puntero por defecto mediante la variable de sistema myisam_data_pointer_size que se añadió en MySQL 4.1.2. (Consulte Sección 5.3.3, “Variables de sistema del servidor”.) Si quiere que todas sus tablas sean capaces de crecer por encima del límite por defecto y quiere mantener sus tablas ligeramente más lentas y más grandes de lo necesario, puede incrementar el tamaño de punter por defecto cambiando esta variable.

- [DEFAULT] CHARACTER SET

  Especifica el conjunto de caracteres para la tabla. CHARSET es un sinónimo.

  para CHARACTER SET.

- COLLATE

  Especifica la colación por defecto de la tabla.

- CHECKSUM

  Póngalo a 1 si quiere que MySQL mantenga un checksum para todos los registros (un checksum que MySQL actualiza automáticamente según cambia la tabla). Esto hace que la tabla tenga actualizaciones más lentas, pero hace más fácil encontrar tablas corruptas. El comando CHECKSUM TABLE muestra el checksum (sólo para MyISAM).

- COMMENT

  Un comentario para su tabla, hasta 60 caracteres.

- MAX_ROWS

  Máximo número de registros que planea almacenar en la tabla. No es un límite absoluto, sino un indicador que la tabla debe ser capaz de almacenar al menos estos registros.

- MIN_ROWS

  Mínimo número de registros que planea almacenar en la tabla.

- PACK_KEYS

  Ponga esta opción a 1 si quiere tener índices más pequeños. Esto hace normalmente que las actualizaciones sean más lentas y las lecturas más rápidas. Poner esta opción a 0 deshabilita la compresión de claves. Ponerla a DEFAULT le dice al motor que comprima sólo columnas CHAR/VARCHAR largas (MyISAM y ISAM sólo).

  Si no usa PACK_KEYS, por defecto se comprimen sólo cadenas, no números. Si usa PACK_KEYS=1, también se empaquetan números.

  Al comprimir claves de números binarios, MySQL usa compresión de prefijo:

  - Cada clave necesita un byte extra para indicar cuántos bytes de la clave previa son los mismos para la siguiente clave.

- El puntero al registro se almacena en orden de el-mayor-byte-primero directamente tras la clave, para mejorar la compresión.

Esto significa que si tiene muchas claves iguales en dos registros consecutivos, todas las “mismas” claves siguientes usualmente sólo ocupan dos bytes (incluyendo el puntero al registro). Comparar esto con el caso ordinario donde las siguente claves ocupan `storage_size_for_key + pointer_size` (donde el tamaño del puntero es usualmente 4). Obtiene un gran beneficio a partir de la compresión de prefijos sñolo si tiene muchos números que sean el mismo. Si todas las claves son totalmente distintas, usa un byte más por clave, si la clave no es una clave que pueda tener valores `NULL` . (En ese caso, el tamaño empaquetado de la clave se almacena en el mismo byte que se usa para marcar si una clave es `NULL`.)

- `PASSWORD`

  Cifra el fichero `.frm` con una contraseña. Esta opción no hace nada en la versión estándar de MySQL.

- `DELAY_KEY_WRITE`

  Póngalo a 1 si quiere retardar actualizaciones de clave para la tabla hasta que la tabla se cierra (sólo en `MyISAM`).

- `ROW_FORMAT`

  Define cómo deben almacenarse los registros. Actualmente esta opción sólo funciona con tablas `MyISAM`. El valor de la opción puede ser `FIXED` o `DYNAMIC` para formato de longitud estática o variable. `myisampack` cambia el tipo a `COMPRESSED`. Consulte Sección 14.1.3, “Formatos de almacenamiento de tablas MyISAM”.

  Desde MySQL/InnoDB-5.0.3, los registros de InnoDB se almacenan de forma más compacta (`ROW_FORMAT=COMPACT`) por defecto. El antiguo formato puede usarse espeficicando `ROW_FORMAT=REDUNDANT`.

- `RAID_TYPE`

  Tenga en cuenta que el soporte para `RAID` se ha eliminado desde MySQL 5.0. Para información sobre `RAID`, consulte Manual de referencia de MySQL 4.1.

- `UNION`

  `UNION` se usa cuando quiere usar una colección de tablas idénticas como una. Funciona sólo con tablas `MERGE` . Consulte Sección 14.2, “El motor de almacenamiento MERGE”.

  En MySQL 5.0, debe tener permisos `SELECT`, `UPDATE`, y `DELETE` para las tablas mapeadas en una tabla `MERGE` . (*Nota*: Originalmente, todas las tablas usadas tenían que estar en la misma base de datos que la tabla `MERGE`. Esta restricción se ha eliminado.)

- `INSERT_METHOD`

  Si quiere insertar datos en una tabla `MERGE` debe especificarlo con `INSERT_METHOD` en la tabla en que se debe insertar el registro. `INSERT_METHOD` es una opción útil para tablas `MERGE` sólo. Use un valor dee `FIRST` o `LAST` para que las inserciones vayan a la primera o última tabla, o un valor de `NO` para evitar inserciones. Consulte Sección 14.2, “El motor de almacenamiento MERGE”.

- `DATA DIRECTORY`, `INDEX DIRECTORY`

Usando DATA DIRECTORY='*directory*' o INDEX DIRECTORY='*directory*' puede especificar dónde debe el moto MyISAM buardar un fichero de datos e índice de una tabla. Tenga en cuenta que el directorio debe ser una ruta completa al directorio (no una ruta relativa).

Estas opciones sólo funcionan cuando no usa la opción --skip-symbolic-links . Su sistema operativo debe tener una llamada realpath() que funcione bien. Consulte Sección 7.6.1.2, “Utilización de enlaces simbólicos para tablas en Unix” para más información.

En MySQL 5.0, puede crear una tabla de otra añadiendo un comando SELECT al final del comando CREATE TABLE :

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQL crea nuevas columnas para todos los elementos en un SELECT. Por ejemplo:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        TYPE=MyISAM SELECT b,c FROM test2;
```

Esto crea una tabla MyISAM con tres columnas, a, b, y c. Tenga en cuenta que las columnas para el comando SELECT se añaden a la derecha de la tabla, no se sobreescriben en la misma. Consulte el siguiente ejemplo:

```
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)
```

Para cada registro en la tabla foo, se inserta un registro en bar con los valores de foo y valores por defecto para las nuevas columnas:

Si hay cualquier error al copiar los datos a la tabla, se borra automáticamente y no se crea.

CREATE TABLE ... SELECT no crea ningún índice automáticamente. Se hace a propósito para hacer el comando lo más flexible posible. Si quiere tener índices en la tabla creada, debe especificarlo antes del comando SELECT :

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Algunas conversiones de tipos de columnas pueden ocurrir. Por ejemplo, el atributo AUTO_INCREMENT no se preserva, y las columnas VARCHAR pueden ser CHAR .

Al crear una tabla con CREATE ... SELECT, asegurése de poner un alias para cualquier llamada a función o expresión en la consulta. Si no lo hace, el comando CREATE puede fallar o crear nombres de columnas no deseados.

```
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;
```

Puede especificar explícitamente el tipo de una columna generada:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

En MySQL 5.0, use LIKE para crear una tabla vacía basada en la definición de otra tabla, incluyendo cualquier atributo de columna e índice definido en la tabla original:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

CREATE TABLE ... LIKE no copia ninguna opción de tabla DATA DIRECTORY o INDEX DIRECTORY especificadas en la tabla original, ni ninguna definición de clave foránea.

Puede preceder SELECT con IGNORE o REPLACE para indicar cómo tratar registros que dupliquen claves únicas. Con IGNORE, los nuevos registros que duplican un registro único existente se descartan. Con REPLACE, los nuevos registros reemplazan a los antiguos con el mismo valor. Si ni IGNORE ni REPLACE se indican, los valores únicos duplicados dan un error.

Para asegurar que el log de update o binario puede usarse para recrear tablas originales, MySQL no permite inserciones concurrentes durante CREATE TABLE ... SELECT.

13.1.5.1. Cambios tácitos en la especificación de columnas

En algunos casos, MySQL cambia especificaciones de columnas silencioasmente de las dadas en un comando CREATE TABLE o ALTER TABLE . Pueden ser cambiso a un tipo de datos, a atributos asociados con un tipo de datos o a una especificación de índice.

Los posibles cambios de tipos de datos se dan en la siguiente lista. Ocurren antes de MySQL 5.0.3. Desde 5.0.3, ocurre un error si no se puede crear una columna usando el tipo de datos especificado.

- Columnas VARCHAR con una longitudo menor que cuatro se cambian a CHAR.
- Si cualquier columna en una tabla tiene una longitud variable, el registro entero pasa a tener longitud variable. Por lo tanto, si una tabla contiene cualquier columna de longitud variable (VARCHAR, TEXT, o BLOB), toda columna CHAR con más de tres caracteres se cambia a columna VARCHAR . Esto no afecta cómo usa las columnas en ningún modo; en MySQL, VARCHAR es sólo un modo distinto de almacenar caracteres. MySQL realiza esta conversión porque ahorra espacio y hacer las operaciones de tabla más rápidas. Consulte Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.
- Antes de MySQL 5.0.3, una columna CHAR o VARCHAR con una longitud mayor a 255 se convierte al tipo TEXT más pequeño que puede contener valores de la longitud dada. Por ejemplo, VARCHAR(500) se convierte en TEXT, y VARCHAR(200000) se convierte en MEDIUMTEXT. Tenga en cuenta que esta conversión resulta en un cambio de comportamiento del tratamiento de espacios finales.

  Conversiones similares ocurren para BINARY y VARBINARY, excepto que se convierten en tipo BLOB.

  Desde MySQL 5.0.3, una columna CHAR o BINARY con una longitud mayor a 255 no se convierte silenciosamente . En su lugar, ocurre un error. Desde MySQL 5.0.6 , la conversión silenciosa de

columnas VARCHAR y VARBINARY con una longitudo mayor a 65,535 no ocurre si el modo estricto SQL está activado. En su lugar, ocurre un error.

- Para una especificación de DECIMAL(*M*,*D*), si *M* no es mayor que *D*, se ajusta por encima. Por ejemplo DECIMAL(10,10) pasa a ser DECIMAL(11,10).

Otros cambios de columna incluyen cambios de atributos o especficación de índice:

- Los tamaños de muestra de TIMESTAMP se descartan. Tenga en cuenta que columnas TIMESTAMP han cambiado considrablemente en versiones recientes de MySQL anteriores a 5.0; para una descripción, consulte Manual de referencia de MySQL 4.1.
- Las columnas que son parte de PRIMARY KEY son NOT NULL incluso si no se declaran como tales.
- Los espacios finales se borran automáticamente para ENUM y SET cuando se crea la tabla.
- MySQL mapea ciertos tipos de columna usados por otras bases de datos SQL a tipos MySQL . Consulte Sección 11.7, “Usar tipos de columnas de otros motores de bases de datos”.
- Si incluye una cláusula USING para especificar un tipo de índice que no sea legal para un motor de almacenamiento dado, pero hay otro tipo de índice disponible que puede usar el motor sin afectar el resultado de la consulta, el motor usa el tipo disponible.

Para ver si MySQL usa un tipo de columna distinto al especificado, realice un comando DESCRIBE o SHOW CREATE TABLE tras crear o alterar la tabla.

Otros cambios de tipo de columna pueden ocurrir si comprime una tabla usando myisampack. Consulte Sección 14.1.3.3, “Características de las tablas comprimidas”.

13.1.6. Sintaxis de DROP DATABASE

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE borrar todas las tablas en la base de datos y borrar la base de datos. Sea *muy* cuidadoso con este comando! Para usarDROP DATABASE, necesita el permiso DROP en la base de datos.

IF EXISTS se usa para evitar un error si la base de datos no existe.

DROP SCHEMA puede usarse desde MySQL 5.0.2.

Si usa DROP DATABASE en una base de datos enlazada simbólicamente, tanto el enlace como la base de datos se borran.

DROP DATABASE retorna el número de tablas que se eliminan. Se corresponde con el número de ficheros .frm borrados.

El comando DROP DATABASE borrar del directorio de base de datos los ficheros y directorios que MySQL puede crear durante operaciones normales:

- Todos los ficheros con estas extensiones:

.BAK	.DAT	.HSH	
.MRG	.MYD	.ISD	
.MYI	.db	.frm	

- Todos los subdirectorios con nombres que tienen dos dígitos hexadecimales 00-ff. Son subdirectorios usados por tablas RAID . (Estos directorios no se borran desde MySQL 5.0, cuando se eliminó el soporte para tablas RAID . Debe convertir las tablas RAID y eliminar estos directorios manualmente antes de actualizar a MySQL 5.0. Consulte Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0”.)
- El fichero db.opt , si existe.

Si permanecen otros ficheros o directorios en el directorio de la base de datos tras que MySQL borre los ficheros listados, el directorio de base de datos no puede borrarse. En este caso, debe borrar cualquier fichero restante manualmente y realizar el comando DROP DATABASE de nuevo.

Puede borrar bases de datos con mysqladmin. Consulte Sección 8.4, “Administrar un servidor MySQL con mysqladmin”.

13.1.7. Sintaxis de DROP INDEX

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX borra el índice llamado *index_name* de la tabla *tbl_name*. En MySQL 5.0, DROP INDEX se mapea a comando ALTER TABLE para borrar el índice. Consulte Sección 13.1.2, “Sintaxis de ALTER TABLE”.

13.1.8. Sintaxis de DROP TABLE

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

DROP TABLE borra una o más tablas. Debe tener el permiso DROP para cada tabla. Todos los datos de la definición de tabla son *borrados*, así que *tenga cuidado* con este comando!

Use IF EXISTS para evitar un error para tablas que no existan. Un NOTE se genera para cada tabla no existente cuando se usa IF EXISTS. Consulte Sección 13.5.4.22, “Sintaxis de SHOW WARNINGS”.

RESTRICT y CASCADE se permiten para hacer la portabilidad más fácil. De momento, no hacen nada.

Nota: DROP TABLE hace un commit automáticamente con la transacción activa,a no ser que use la palabra TEMPORARY.

La palabra TEMPORARY tiene el siguiente efecto:

- El comando sólo borra tablas TEMPORARY.
- El comando no acaba una transacción en marcha.
- No se chequean derechos de acceso. (Una tabla TEMPORARY es visible sólo para el cliente que la ha creado, así que no es necesario.)

Usar TEMPORARY es una buena forma de asegurar que no borra accidentalmente una tabla no TEMPORARY.

13.1.9. Sintaxis de RENAME TABLE

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

Este comando renombra una o más tablas.

La operación de renombrar se hace automáticamente, lo que significa que ningún otro flujo puede acceder a ninguna de las tablas mientras se ejecuta el renombrado. Por ejemplo, si tiene una tabla existente `old_table`, puede crear otra tabla `new_table` con la misma estructura pero vacía, y luego reemplazar la tabla existente con la vacía como sigue:

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

Si el comando renombra más de una tabla, las operaciones de renombrado se realizan de izquierda a derecha. Si quiere intercambiar dos nombres de tablas, puede hacerlo así (asumiendo que no existe ninguna tabla llamada `tmp_table`):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

Mientras haya dos bases de datos en el mismo sistema de ficheros puede renombrar una tabla para moverla de una base de datos a otra:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

Cuando ejecuta `RENAME`, no puede tener ninguna tabla bloqueada o transacciones activas. Debe tener los permisos `ALTER` y `DROP` en la tabla original, y los permisos `CREATE` y `INSERT` en la nueva tabla.

Si MySQL encuentra cualquier error en un renombrado múltiple, hace un renombrado inverso para todas las tablas renombradas para devolver todo a su estado original.

13.2. Sentencias de manipulación de datos (Data Manipulation Statements)

13.2.1. Sintaxis de `DELETE`

Sintaxis para una tabla:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

Sintaxis para múltiples tablas:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
tbl_name[.*] [, tbl_name[.*] ...]
FROM table_references
[WHERE where_definition]
```

O:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
```

```
    FROM tbl_name[.*] [, tbl_name[.*] ...]
    USING table_references
    [WHERE where_definition]
```

DELETE borra los registros de *tbl_name* que satisfacen la condición dada por *where_definition*, y retorna el número de registros borrados.

Si realiza un comando DELETE sin cláusula WHERE se borran todos los registros. Una forma más rápida de hacerlo, cuando no quiere saber el número de registros borrados, se usa TRUNCATE TABLE. Consulte Sección 13.2.9, “Sintaxis de TRUNCATE”.

Si borra el registro conteniendo el máximo valor para una columna AUTO_INCREMENT , el valor se reúsa para una tabla BDB , pero no para tablas MyISAM o InnoDB . Si borra todos los registros en la tabla con DELETE FROM *tbl_name* (sin cláusula WHERE) en modo AUTOCOMMIT , la secuencia comienza para todos los tipos de tabla excepto para InnoDB y MyISAM. Hay algunas excepciones para este comportamiento para tablas InnoDB , como se discute en Sección 15.6.3, “Cómo funciona una columna AUTO_INCREMENT en InnoDB”.

Para tablas MyISAM y BDB , puede especificar una columna AUTO_INCREMENT secundaria en una clave de múltiples columnas. En este caso, el reúso de valores borrados del inicio de la secuencia se realiza incluso para tablas MyISAM . Consulte Sección 3.6.9, “Utilización de AUTO_INCREMENT”.

El comando DELETE soporta los siguientes modificadores:

- Si especifica LOW_PRIORITY, la ejecución de DELETE se retarda hasta que no hay más clientes leyendo de la tabla.
- Para tablas MyISAM , si usa la palabra QUICK , el motor de almacenamiento no mezcla las hojas del índice durante el borrado, que puede acelerar algunos tipos de operaciones de borrado.
- En MySQL 5.0, la palabra clave IGNORE hace que MySQL ignore todos los errores durante el proceso de borrar registros. (Los errores encontrados durante la etapa de parseo se procesan de la forma habitual.) Los errores que se ignoran debido al uso de esta opción se retornan como advertencias.

La velocidad de las operaciones de borrado pueden verse afectadas por factores discutidos en Sección 7.2.16, “Velocidad de sentencias DELETE”.

En tablas MyISAM , los registros borrados se mantienen en una lista enlazada y las operaciones INSERT siguientes reúsan antiguas posiciones de registro. Para reclamar espacio no usado y reducir tamaño de fichero, use el comando OPTIMIZE TABLE o la utilidad myisamchk para reorganizar las tablas. OPTIMIZE TABLE es más sencillo, pero myisamchk es más rápido. Consulte Sección 13.5.2.5, “Sintaxis de OPTIMIZE TABLE” y Sección 5.8.3.10, “Optimización de tablas”.

El modificador QUICK afecta si las hojas del índice es mezclan en operaciones de borrado. DELETE QUICK es más útil para aplicaciones en que los valores del índice para registros borrados se replazan con valores similares de registros insertados posteriormente. En este caso, los agujeros dejados por los valores borrados se reúsan.

DELETE QUICK no es útil cuando los valores borrados conducen a bloques de índices no rellenos con un rango de valores índice para el que vuelven a ocurrir nuevas inserciones. En este caso, el uso de QUICK puede conducir a un gasto de espacio que queda sin reclamar. Aquí hay un ejemplo de este escenario:

1. Cree una tabla que contenga una columna AUTO_INCREMENT indexada.
2. Inserta varios registros en la tabla. Cada inserción resulta en un valor índice que se añade al final del índice.

3. Borra un bloque de registros al final del rango de la columna usando DELETE QUICK.

En este escenario, los bloques de índice asociados con los valores de índice borrado quedan sin rellenar pero no se mezclan con otros bloques de índice debido al uso de QUICK. Quedan sin rellenar cuando hay nuevas inserciones, ya que los nuevos registros no tienen valores índice en el rango borrado. Además, quedan sin rellenar incluso si luego usa DELETE sin QUICK, a no ser que algunos de los valores de índice borrados estén en los bloques de índice dentro o adyacentes a los bloques no rellenos. Para reclamar el espacio de índice sin usar bajo estas circunstancias use OPTIMIZE TABLE.

Si va a borrar varios registros de una tabla, puede ser más sencillo usar DELETE QUICK seguido por OPTIMIZE TABLE. Esto reconstruye el índice en lugar de realizar varias operaciones de mezcla de bloques de índice.

La opción de MySQL LIMIT *row_count* para DELETE le dice al servidor el máximo número de registros a borrar antes de retornar el control al cliente. Esto puede usarse para asegurar que un comando DELETE específico no tarada demasiado tiempo. Puede simplemente repetir el comando DELETE hasta que el número de registros afectados sea menor que el valor LIMIT .

Si el comando DELETE incluye una cláusula ORDER BY , los registros se borran en el orden especificado por la cláusula. Esto es muy útil sólo en conjunción con LIMIT. Por ejemplo, el siguiente ejemplo encuentra registros coincidentes con la cláusula WHERE ordenados por timestamp_column, y borra el primero (el más viejo).

```
DELETE FROM somelog
WHERE user = 'jcole'
ORDER BY timestamp_column
LIMIT 1;
```

Puede especificar múltiples tablas en un comando DELETE para borrar registros de una o más tablas dependiendo de una condición particular en múltiples tablas. Sin embargo, no puede usar ORDER BY o LIMIT en un DELETE de múltiples tablas.

La parte *table_references* lista las tablas involucradas en el join. Esta sintaxis se describe en Sección 13.2.7.1, “Sintaxis de JOIN”.

Para la primera sintaxis, sólo los registros coincidentes de las tablas listadas antes de la cláusula FROM se borran. Para la segunda sintaxis, sólo los registros coincidentes de las tablas listadas en la cláusula FROM (antes de la cláusula USING) se borran. El efecto es que puede borrar registros para varias tablas al mismo tiempo y tienen tablas adicionales que se usan para buscar:

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

O:

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

Estos comandos usan las tres tablas al buscar registros a borrar, pero borrar los registros coincidentes sólo para las tablas t1 y t2.

Los ejemplos anteriores muestran inner joins usando el operador coma, pero comandos DELETE de varias tablas pueden usar cualquier tipo de join permitido por comandos SELECT tales como LEFT JOIN.

La sintaxis permite .* tras los nombres de tabla para compatibilidad con Access.

Si usa un comando DELETE de varias tablas incluyendo tablas InnoDB para las que hay restricciones de clave foránea, el optimizador MySQL puede procesar tablas en un orden ditinto del de su relación padre/

hijo. En este caso, el comando falla y se deshace. En su lugar, debe borrar de una tabla úncia y confiar en la capacidad de ON DELETE que proporciona InnoDB para hacer que las otras tablas se modifiquen correctamente.

Nota: En MySQL 5.0, debe usar el alias (si se dió) al referirse a un nombre de tabla:

En MySQL 4.1:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Borrados cruzados entre bases de datos se soportan para borrados de varias tablas, pero en este caso, debe referirse a las tablas sin usar alias. Por ejemplo:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

Actualmente, no puede borrar desde una tabla y seleccionar de la misma tabla en una subconsulta.

13.2.2. Sintaxis de DO

```
DO expr [, expr] ...
```

DO ejecuta la expresión pero no retorna ningún resultado. Esto es una abreviación de SELECT *expr, ...*, pero tiene la ventaja que es más rápido cuando no le importa el resultado.

DO es útil principalmente con funciones que tienen efectos colaterales, tales como RELEASE_LOCK().

13.2.3. Sintaxis de HANDLER

```
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
```

El comando HANDLER proporciona acceso directo a las interfaces del motor de la tabla. En MySQL 5.0, está disponible para tablas MyISAM y InnoDB .

El comando HANDLER ... OPEN abre una tabla, haciéndola accesible mediante posteriores comandos HANDLER ... READ . Este objeto de tabla no se comparte con otros flujos y no se cierra hasta que el flujo llama HANDLER ... CLOSE o el flujo termina. Si abre la tabla usando un alias, referencias posteriores a la tabla con otros comandos HANDLER deben usar el alias en lugar del nombre de la tabla.

La primera sintaxis HANDLER ... READ recibe un registro donde el índice especificado satisface los valores dados y la condición WHERE se cumple. Si tiene un índice de múltiples columnas, especifique los valores de la columna índice como una lista separada por comas. Los valores epecificados para todas las columnas en el índice, o los valores específicos para un prefijo a la izquierda de las columnas índice. Suponga que un índice incluye tres columnas llamadas col_a, col_b, y col_c, en ese orden. El comando HANDLER puede especificar valores para las tres columnas en el índice, o para las columnas en el prefijo a la izquierda. Por ejemplo:

```
HANDLER ... index_name = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... index_name = (col_a_val,col_b_val) ...
HANDLER ... index_name = (col_a_val) ...
```

La segunda sintaxis `HANDLER ... READ` recibe un registro de la tabla en orden del índice que cumple la condición `WHERE` .

La tercera sintaxis `HANDLER ... READ` recibe un registro de la tabla en orden de registro natural que cumple la condición `WHERE` . Es más rápido que `HANDLER tbl_name READ index_name` cuando se desea un escaneo completo de tabla. El orden de registro natural es el orden en que se almacenan los registros en un fichero de datos de una tabla `MyISAM`. Este comando funciona para tablas `InnoDB` también, pero no hay tal concepto porque no hay un fichero de datos separado.

Sin una cláusula `LIMIT` , todas las formas de `HANDLER ... READ` reciben un único registros si una está disponible. Para retornar un número específico de registros, incluya una cláusula `LIMIT` . Tiene la misma sintaxis que para el comando `SELECT` . Consulte Sección 13.2.7, “Sintaxis de SELECT”.

`HANDLER ... CLOSE` cierra una tabla que se abrió con `HANDLER ... OPEN`.

Nota: Para emplear la interfaz `HANDLER` para referirse a una tabla `PRIMARY KEY`, use el identificador `` `PRIMARY` `` entrecomillado:

```
HANDLER tbl_name READ `PRIMARY` > (...);
```

`HANDLER` es un comando de bajo nivel. Por ejemplo, no proporciona consistencia. Esto es, `HANDLER ... OPEN` *no* toma una muestra de la tabla, y *no* bloquea la tabla. Esto significa que tras un comando `HANDLER ... OPEN` realizado, los datos de la tabla pueden ser modificados (por este o por otro flujo) y estas modificaciones pueden aparecer sólo parcialmente en escaneos `HANDLER ... NEXT` o `HANDLER ... PREV` .

Hay varias razones para usar la interfaz `HANDLER` en lugar de comandos `SELECT` normales:

- `HANDLER` es más rápido que `SELECT`:
  - Un objeto de tratamiento de motor de almacenamiento designado se reserva para `HANDLER ... OPEN`. El objeto se reúsa para posteriores comandos `HANDLER` para esa tabla; no necesita reinicializarse para cada una.
  - Hay menos parseo.
  - No hay sobrecarga del chequeo de consultas ni optimizador.
  - La tabla no tiene que estar bloqueada entre peticiones.
  - La interfaz del handler no tiene que propocionar una vista de los datos consistente (por ejemplo, se permiten dirty reads), así que el motor puede usar optimización que `SELECT` no permite.
- `HANDLER` hace mucho más fácil portar aplicaciones que usen una interfaz tipo `ISAM` a MySQL.
- `HANDLER` le permite consultar una base de datos de forma difícil o imposible de realizar con `SELECT`. La interfaz de `HANDLER` es una forma más natural de consultar los datos cuando se trabaja con aplicaciones que proporcionan una interfaz de usuario interactiva a la base de datos.

13.2.4. Sintaxis de `INSERT`

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
```

```
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

O:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

O:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

`INSERT` inserta nuevos registros en una tabla existente. Las formas `INSERT ... VALUES` y `INSERT ... SET` del comando insertan registros basados en valores explícitamente especificados. La forma `INSERT ... SELECT` inserta registros seleccionados de otra tabla o tablas. `INSERT ... SELECT` se discute en Sección 13.2.4.1, “Sintaxis de `INSERT ... SELECT`”.

`tbl_name` es la tabla en que los registros deben insertarse. Las columnas para las que el comando proporciona valores pueden especificarse como sigue:

- La lista de nombres de columna o la cláusula `SET` indican las columnas explícitamente.
- Si no especifica la lista de columnas para `INSERT ... VALUES` o `INSERT ... SELECT`, los valores para cada columna en la tabla deben proporcionarse en la lista `VALUES` o por el `SELECT`. Si no sabe el orden de las columnas en la tabla, use `DESCRIBE tbl_name` para encontrarlo.

Los valores de columna pueden darse de distintos modos:

- Si no está ejecutando el modo estricto, cualquier columna que no tenga un valor asignado explícitamente recibe su valor por defecto (explícito o implícito). Por ejemplo, si especifica una lista de columnas que no nombra todas las columnas en la tabla, las no nombradas reciben sus valores por defecto. Los valores por defecto asignados se describen en Sección 13.1.5, “Sintaxis de `CREATE TABLE`”. Consulte Sección 1.7.6.2, “Restricciones (constraints) sobre datos inválidos”.

  Si quiere que un comando `INSERT` genere un error a no ser que especifique explícitamente valores para todas las columnas que no tienen un valor por defecto, debe usar modo `STRICT` . Consulte Sección 5.3.2, “El modo SQL del servidor”.

- Use `DEFAULT` para asignar a una columna explícitamente su valor por defecto. Esto hace más fácil escribir comandos `INSERT` que asignan valores a todas las columnas excepto unas pocoas, ya que le permite evitar la escritura de una lista de valores `VALUES` incompleta. De otro modo, tendría que escribir la lista de los nombres de columna correspondientes a cada valor en la lista `VALUES` .

  En MySQL 5.0, puede usar `DEFAULT(col_name)` como forma más general que puede usarse en expresiones para producir un valor por defecto de una columna.

- Si la lista de columnas y la lista `VALUES` están vacías, `INSERT` crea un registro con cada conjunto de columnas con sus valores por defecto:

  ```
  mysql> INSERT INTO tbl_name () VALUES();
  ```

En modo STRICT obtendrá un error si una columna no tiene un valor por defecto. De otro modo, MySQL usará el valor implícito para cualquier columna sin un valor explícito por defecto definido.

- Puede especificar una expresión *expr* para proporcionar un valor de columna. Esto puede involucar convesión de tipos si el tipo de la expresión no coincide con el tipo de la columna, y la conversión de un valor dado puede resultar en distintos valores insertados dependiendo del tipo de columna. Por ejmplo, insertar la cadena '1999.0e-2' en una columna INT, FLOAT, DECIMAL(10,6), o YEAR resulta en los valores 1999, 19.9921, 19.992100, y 1999 insertados, respectivamente. La razón de que el valor almacenado en las columnas INT y YEAR sea 1999 es que la conversión cadena-a-entero consulta sólo el trozo de la parte inicial de la cadena que se puede considerar como un entero válido o año. Para las columnas de coma flotante o punto fijo, la conversión cadena-a-coma-flotante considera la cadena entera un valor válido.

  Una expresión *expr* puede referirse a cualquier columna que se haya asignado antes en una lista de valores. Por ejemplo, puede hacer esto porque el valor para col2 se refiere a col1, que se ha asignado préviamente:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  Pero lo siguiente no es legal, ya que el valor para col1 se refiere a col2, que se asigna tras col1:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

  Una excepción involucra a columnas que contienen valores AUTO_INCREMENT . Como el valor AUTO_INCREMENT se genera tras otras asignaciones de valores, cualquier referencia a una columna AUTO_INCREMENT en la asignación retorna un 0.

El comando INSERT soporta los siguientes modificadores:

- Si usa la palabra DELAYED, el servidor pone el registro o registros a ser insertados en un búffer, y el cliente realizando el comando INSERT DELAYED puede continuar. Si la tabla está en uso, el servidor trata los registros. Cuando la tabla se libera, el servidor comienza a insertar registros, chequeando periódicamente para ver si hay alguna petición de lectura para la tabla. Si la hay, la cola de registros retardados se suspende hasta que la tabla se libera de nuevo. Consulte Sección 13.2.4.2, “Sintaxis de INSERT DELAYED”.

- Si usa la palabra LOW_PRIORITY , la ejecución de INSERT se retrasa hasta que no hay otros clientes leyendo de la tabla. Esto incluye a otros clientes que comiencen a leer mientras que los clientes existentes están leyendo, y meintras el comando INSERT LOW_PRIORITY está en espera. Es posible, por lo tanto, para un cliente que realice un comando INSERT LOW_PRIORITY esperar durante mucho tiempo (o incluso para siempre) en un entorno de muchas lecturas. (Esto es un contraste de INSERT DELAYED, que deja al cliente continuar. Consulte Sección 13.2.4.2, “Sintaxis de INSERT DELAYED”.) Tenga en cuenta que LOW_PRIORITY no debe usarse normalmente con tablas MyISAM y que hacerlo deshabilita inserciones concurrentes. Consulte Sección 14.1, “El motor de almacenamiento MyISAM”.

- Si especifica HIGH_PRIORITY, deshabilita el efecto de la opción --low-priority-updates si el servidor se arrancó con esa opción. Hace que las insecionces concurrentes no se usen.

- Los valores afectados por un INSERT pueden usarse usando la función mysql_affected_rows() de la API de C. Consulte Sección 24.2.3.1, “mysql_affected_rows()”.

- Si usa la palabra IGNORE en un comando INSERT , los errores que ocurren mientras se ejecuta el comando se tratan como advertencias. Por ejemplo, sin IGNORE, un registro que duplique un índice UNIQUE existente o valor PRIMARY KEY en la tabla hace que un error de clave duplicada en el

comando se aborte. Con IGNORE, el registro todavía no se inserta, pero no se muestra error. Las conversionse de datos dispararían errores y abortarían el comando si no se sepecificara IGNORE . Con IGNORE, los valores inválidaos se ajustan al valor más cercano y se insertan; las advertencias se producen pero el comando no se aborta. Puede determinar con la función mysql_info() de la API de C cuántos registros se insertan realmente en la tabla.

Si especifica ON DUPLICATE KEY UPDATE, y se inserta un registro que duplicaría un valor en un índice UNIQUE o PRIMARY KEY, se realiza un UPDATE del antiguo registro. Por ejemplo, si la columna a se declara como UNIQUE y contiene el valor 1, los siguientes dos comandos tienen efectos idénticos:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
-> ON DUPLICATE KEY UPDATE c=c+1;

mysql> UPDATE table SET c=c+1 WHERE a=1;
```

El valor de registros afectados es 1 si el registros se inserta como un nuevo registro y 2 si un valor existente se actualiza.

Nota: Si la columna b es única, el INSERT sería equivalente a este comando UPDATE :

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

Si a=1 OR b=2 se cumple para varios registros, sólo *un* registro se actualiza. En general, debería intentar evitar usar una cláusula ON DUPLICATE KEY en tablas con claves únicas múltiples.

MySQL 5.0 permite el uso de la función VALUES(col_name) en la cláusula UPDATE que se refiere a los valores de columna de la porción INSERT del comando INSERT ... UPDATE . En otras palabras, VALUES(col_name) en la cláusula UPDATE se refiere al valor de *col_name* que se insertarían, no ocurre conflicto de clave duplicada. Esta función es especialmente útil en inserciones de múltiples registros. La función VALUES() tiene sentido sólo en comandos INSERT ... UPDATE y retorna NULL de otro modo.

Ejemplo:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
-> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

Este comando es idéntico a los siguientes dos comandos:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
-> ON DUPLICATE KEY UPDATE c=3;
mysql> INSERT INTO table (a,b,c) VALUES (4,5,6)
-> ON DUPLICATE KEY UPDATE c=9;
```

Cuando usa ON DUPLICATE KEY UPDATE, la opción DELAYED se ignora.

Puede encontrar el valor usado para una columna AUTO_INCREMENT usando la función SQL LAST_INSERT_ID() . Desde la API C, use la función mysql_insert_id() . Sin embargo, debe tener en cuenta que las dos funciones no siempre se comportan idénticamente. El comportamiento de comandos INSERT respecto a columnas AUTO_INCREMENT se discute en Sección 12.9.3, “Funciones de información” y Sección 24.2.3.34, “mysql_insert_id()”.

Si usa un comando INSERT ... VALUES con listas de múltiples valores o INSERT ... SELECT, el comando retorna una cadena de información en este formato:

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records indica el número de registros procesados por el comando. (Este no es necesariamente el número de registros realmente insertados, ya que Duplicates puede ser distinto a cero.) Duplicates indica el número de registros que no pueden insertarse ya que duplicarían algunos valores de índice únicos existentes Warnings indicata el número de intentos para insertar valores de columna que fueron problemáticos por algo. Las advertencias pueden ocurrir bajo cualquiera de las siguientes condiciones:

- Insertar NULL en una columna que se ha declarado NOT NULL. Para comandos INSERT de múltiples columnas o comandos INSERT INTO... SELECT, la columna se asigna con el valor por defecto para el tipo de datos de la columna. Este es 0 para tipos numéricos, la cadena vacía ('') para tipos de cadenas, y el valor “cero” para tipos de fecha y hora. Los comandos INSERT INTO ... SELECT se tratan del mismo modo que inserciones de múltiples registros porque el servidor no examina el resultado del SELECT para ver si retorna o no un único registro. (para un único registro INSERT, no hay ninguna advertencia cuando NULL se inserta en una columna NOT NULL . En lugar de eso, el comando falla con un error.)
- Poner en una columna numérica un valor fuera del rango de la columna. El valor se redondea al punto final del rango más cercano.
- Asigne un valor tal como '10.34 a' a una columna numérica. El texto final se elimina y la parte numérica se inserta. Si el valor de cadena no tiene parte inicial numérica, la columna se pone a 0.
- Insertar una cadena en una columna de cadena (CHAR, VARCHAR, TEXT, o BLOB) que excede la maxima longitud de la columna. El valor se trunca a la máxima longitud de la columna.
- Insertar un valor en una columna de fecha u hora que es ilegal para el tipo de la columna. La columna se asigna con el valor cero apropiado para el tipo.

Si usa la API de C, la cadena de información puede obtenerse invocando la función mysql_info() Consulte Sección 24.2.3.32, “mysql_info()”.

13.2.4.1. Sintaxis de INSERT ... SELECT

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Con INSERT ... SELECT, puede insertar rápidamente varios registros en un atabla desde una o varias tablas.

Por ejemplo:

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

La siguiente condición sirve para un comando INSERT ... SELECT :

- En MySQL 5.0, especifique IGNORE explícitamente para ignorar registros que causarían violaciones de clave duplicada.
- No use DELAYED con INSERT ... SELECT.
- En MySQL 5.0, la tabla objetivo del comando INSERT puede aparecer en la cláusula FROM de la parte SELECT de la consulta. (Esto no era posible en algunas versiones antiguas de MySQL.)

- Las columnas AUTO_INCREMENT funcionan normalmente.
- Para asegurar que el log binario puede usarse para recrear las tablas originales, MySQL no permite inserciones concurrentes durante INSERT ... SELECT.
- Actualmente, no puede insertar en una tabla y seleccionar de la misma tabla en una subconsulta.

En las partes de valores de ON DUPLICATE KEY UPDATE puede referirse a una columna en otras tablas, mientras no use GROUP BY en la parte SELECT . Un efecto lateral es que debe calificar los nombres de columna no únicos en la parte de valores.

Puede usar REPLACE en lugar de INSERT para sobreescribir registros antiguos REPLACE es la contraparte de INSERT IGNORE en el tratamiento de nuevos registros que contienen valores de clave única que duplican registros antiguos: Los nuevos registros se usan para reemplazar los antiguos registros en lugar de descartarlos.

13.2.4.2. Sintaxis de INSERT DELAYED

```
INSERT DELAYED ...
```

La opción DELAYED para el comando INSERT es una extensión de MySQL del estándar SQL muy útil si tiene clientes que no pueden esperar a que se complete el INSERT . Este es un problema común cuando usa MySQL para loguear y periódicamente ejecuta comandos SELECT y UPDATE que tardan mucho tiempo en completarse.

Cuando un cliente usa INSERT DELAYED, obtiene un ok del servidor una vez, y el registro se encola para insertarse cuando la tabla no está en uso por otro flujo.

Otro beneficio de usar INSERT DELAYED es que las inserciones desde varios clientes se tratan juntas y se escriben en un bloque. Esto es mucho más rápido que realizar inserciones separadas.

Hay algunas restricciones al uso de DELAYED:

- En MySQL 5.0, INSERT DELAYED funciona sólo con tablas MyISAM y MEMORY . Para tablas MyISAM , si no hay bloques libres en medio del fichero de datos, se soportan comandos SELECT y INSERT concurrentes. Bajo estas circunstáncias, muy raramente necesitará usar INSERT DELAYED con MyISAM. Consulte Sección 14.1, “El motor de almacenamiento MyISAM” y Sección 14.3, “El motor de almacenamiento MEMORY (HEAP)”.
- En MySQL 5.0, INSERT DELAYED debe usarse sólo para comandos INSERT que especifiquen una lista de valores. El servidor ignora DELAYED para comandos INSERT DELAYED ... SELECT .
- El servidor ignora DELAYED para comandos INSERT DELAYED ... ON DUPLICATE UPDATE.
- Debido a que el comando retorna inmediatamente antes que los registros se inserten, no puede usar LAST_INSERT_ID() para obtener el valor AUTO_INCREMENT que el comando genera.
- Los registros DELAYED no son visibles por los comandos SELECT hasta que se hayan insertado realmente.
- DELAYED se ignora en la replicación de esclavos porque puede causar que el esclavo tenga distintos datos que el maestro.

Tenga en cuenta que los registros encolados se tratan sólo en memoria hasta que se insertan en la tabla. Esto significa que si termina mysqld forzadamente (por ejemplo, con kill -9) o si mysqld muere inesperadamente, cualquier registro encolado que no se escriba en disco se pierde.

A continuación se describe en detalle qué ocurre cuando usa la opción DELAYED con INSERT o REPLACE. En esta descriión, el “flujo” es el flujo que recibe un comando INSERT DELAYED y “handler” es el flujo que trata todos los comandos INSERT DELAYED para una tabla particular.

- Cuando un flujo ejecuta un comando DELAYED para una tabla, un flujo handler se crea para procesar todos los comandos DELAYED para la tabla, si tal handler no existía préviamente.
- El flujo chequea si el handler ha adquirido préviamente un bloqueo DELAYED ; si no, le dice al flujo handler que lo haga. El bloqueo DELAYED puede obtenerse incluso si otros flujos tienen el bloqueo READ o WRITE en la tabla. Sin embargo, el handler espera a todos los bloqueos ALTER TABLE o FLUSH TABLES para asegurar que la estructura de tabla está actualizada.
- El flujo ejecuta el comando INSERT, pero en lugar de escribir el registro en la tabla, pone una copia del registro final en una cola administrada por el flujo handler. Cualquier error de sintaxis es detectado por el flujo y se reporta al programa cliente.
- El cliente no puede obtener del servidor el número de registros duplicados o el valor AUTO_INCREMENT del registro resultante, ya que INSERT retorna antes que se complete la operación de inserción. (Si usa la API C, la función mysql_info() no retorna nada inteligible por la misma razón.)
- El log binario se actualiza por parte del flujo handler cuando el registro se inserta en la tabla. En caso de inserciones de múltiples registros, el log binario se actualiza cuando el primer registro se inserta.
- Tras cada delayed_insert_limit los registros se escriben, el handler chequea si algún comando SELECT todavía está pendiente. Si es así, les permite ejecutarse antes de continuar.
- Cuando el handler no tiene más registros en su cola, la tabla se desbloquea. Si no se reciben nuevos comandos INSERT DELAYED en delayed_insert_timeout segundos, el handler termina.
- Si más de delayed_queue_size registros están pendientes en una cola de handler específica, el flujo que pida el INSERT DELAYED espera hasta que haya espacio en la cola. Esto se hace para asegurar que mysqld no usa toda la memoria para la cola de memoria retrasada.
- El flujo handler se muestra en l lista de procesos MySQL con delayed_insert en la columna Command . Si muere si ejecuta un comando FLUSH TABLES o puede matarlo con KILL thread_id. Sin embargo, antes de salir, primero almacena todos los registros encolados en la tabla. Durante esta operación no acepta ningún nuevo comando INSERT de otros flujos. Si ejecuta un comando INSERT DELAYED a continuación, se crea un nuevo flujo handler.

  Tenga en cuenta que esto significa que comandos INSERT DELAYED tienen mayor prioridad que comandos INSERT normales si hay un handler INSERT DELAYED en ejecución. Otros comandos de actualización tienen que esperar hast que la cola INSERT DELAYED está vacía, alguien termine el flujo handler (con KILL thread_id), o alguien ejecute un FLUSH TABLES.
- Las siguientes variables de estado proporcionan información acerca de comandos INSERT DELAYED :

Variable de estado	Significado
Delayed_insert_threads	Número de flujos handler
Delayed_writes	Número de registros escritos con INSERT DELAYED
Not_flushed_delayed_rows	Número de registros esperando a ser escritos

  Puede ver estas variables ejecutando un comando SHOW STATUS o mysqladmin extended-status.

Tenga en cuenta que INSERT DELAYED es más lento que un INSERT normal si la tabla no está en uso. También hay una sobrecarga adicional para el servidor debido a que tiene que tratar un flujo separado

para cada tabla en que haya registros retardados. Esto significa que debe usar INSERT DELAYED sólo cuando esté realmente seguro que lo necesita.

13.2.5. Sintaxis de LOAD DATA INFILE

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [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,...)]
```

El comando LOAD DATA INFILE lee registros desde un fichero de texto a una tabla a muy alta velocidad. El nombre de fichero debe darse como una cadena literal.

Para más información acerca de la eficiencia de INSERT contra LOAD DATA INFILE y acelerar LOAD DATA INFILE, consulte Sección 7.2.14, “Velocidad de la sentencia INSERT”.

En MySQL 5.0, el conjunto de caracteres indicado por la variable de sistema character_set_database se usa para interpretar la información en el fichero. SET NAMES y el valor de character_set_client no afecta la interpretación de la entrada.

Puede cargar ficheros de datos usando la utilidad mysqlimport ; opera enviando un comando LOAD DATA INFILE al servidor. La opción --local hace que mysqlimport lea ficheros de datos desde el equipo cliente. Puede especificar la opción --compress para obtener un mejor rendimiento en redes lentas si el cliente y el servidor soportan el protocolo comprimido. Consulte Sección 8.9, “El programa para importar datos mysqlimport”.

Si usa LOW_PRIORITY, la ejecución del comando LOAD DATA se retarda hasta que no haya más clientes leyendo de la tabla.

Si especifica CONCURRENT con una tabla MyISAM que satisfaga la condición para inserciones concurrentes (esto es, no contiene bloques libres en medio), entonces otros flujos pueden recibir datos desde la tabla mientras se ejecuta LOAD DATA . Usar esta opción afecta al rendimiento de LOAD DATA ligeramente, incluso si no hay otro flujo usando la tabla al mismo tiempo.

Si se especifica LOCAL, se interpreta respecto al cliente final de la conexión:

- Si especificamos LOCAL, el programa cliente lee el fichero en el equipo cliente y lo envía al servidor. Podemos indicar la ruta completa del fichero para especificar su localización exacta. Si indicamos la ruta relativa, el fichero se interpreta relativo al directorio en el que el cliente se inició.
- Si no se especifica LOCAL , el fichero tiene que estar en el equipo servidor y el servidor lo lee directamente.

Al localizar ficheros en el equipo servidor, el servidor usa las siguientes reglas:

- Si se da una ruta absoluta, el servidor usa la ruta como tal.

- Si se da una ruta relativa con uno o más componentes, el servidor busca este fichero relativo al directorio de datos del servidor.
- Si se da un nombre de fichero sin ninguna ruta, el servidor busca el fichero en el directorio de base de datos de la base de datos por defecto.

Tenga en cuenta que estas reglas significan que un fichero llamado `./myfile.txt` se lee del directorio de datos del servidor, mientras que el mismo fichero llamado como `myfile.txt` se lee desde el directorio de base de datos de la base de datos por defecto. Por ejemplo, el siguiente comando `LOAD DATA` lee el fichero `data.txt` del directorio de la base de datos para `db1` porque `db1` es la base de datos actual, incluso si el comando carga explícitamente el fichero en una tabla en la base de datos `db2`:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Tenga en cuenta que las rutas de windows se especifican usando barras en lugar de antibarras. Si usa barras, debe doblarlas.

Por razones de seguridad, al leer ficheros de texto localizados en el servidor, los ficheros deben residir en el directorio de la base de datos o ser leíbles por todo el mundo. Además, para usar `LOAD DATA INFILE` en ficheros del servidor, debe tener el permiso `FILE` .

Consulte Sección 5.6.3, “Privilegios de los que provee MySQL”.

Usar `LOCAL` es un poco más lento que dejar al servidor acceder al fichero directamente, porque el contenido del fichero debe enviarse por la conexión desde el cliente al servidor . Por otra parte, no necesita el permiso `FILE` para cargar ficheros locales.

En MySQL 5.0, `LOCAL` funciona sólo si su servidor y su cliente lo tienen activado. Por ejemplo, si `mysqld` se arranca con `--local-infile=0`, entonces `LOCAL` no funciona. Consulte Sección 5.5.4, “Cuestiones relacionadas con la seguridad y `LOAD DATA LOCAL`”.

Si necesita `LOAD DATA` para leer desde un pipe, puede usar la siguiente técnica (aquí cargamos el listado del directorio `/` en una tabla):

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
find / -ls > /mysql/db/x/x
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

Las palabaras `REPLACE` y `IGNORE` controlan el tratamiento de registros de entrada que duplican registros existentes en claves únicas.

Si especifica `REPLACE`, los registros de entrada reemplazan registros existentes (en otras palabras, los registros que tienen el mismo valor para una clave primaria o única que un registro existente). Consulte Sección 13.2.6, “Sintaxis de `REPLACE`”.

Si especifica `IGNORE`, los registros de entrada que dupliquen un registro existente en una clave única se ignoran. Si no especifica ninguna opción, el comportamiento depende de si la palabra `LOCAL` se ha especificado o no. Sin `LOCAL`, ocurre un error cuando se encuentra un valor de clave duplicado, y el resto del fichero de texto se ignora. Con `LOCAL`, el comportamiento por defecto es el mismo que si se especifica `IGNORE`, esto es porque el servidor no tiene forma de parar la transmisión del fichero en medio de la operación.

Si quiere ignorar restricciones de clave foránea durante la operación de carga, puede realizar un comando `SET FOREIGN_KEY_CHECKS=0` antes de ejecutar `LOAD DATA`.

Si usa LOAD DATA INFILE en una tabla vacía MyISAM , todos los índices no únicos se crean en batch separados (como para REPAIR TABLE). Esto hace LOAD DATA INFILE mucho más rápido cuando tiene varios índices. Normalmente esto es muy rápido, pero en algunos casos extremos, puede crear los índices incluso más rápido desactivándolos con ALTER TABLE ... DISABLE KEYS antes de cargar el fichero en la tabla y usar ALTER TABLE ... ENABLE KEYS para recrear los índices tras cargar el fichero. Consulte Sección 7.2.14, “Velocidad de la sentencia INSERT”.

LOAD DATA INFILE es el complemento de SELECT ... INTO OUTFILE. (Consulte Sección 13.2.7, “Sintaxis de SELECT”.) Para escribir datos de una tabla en un fichero use SELECT ... INTO OUTFILE. Para leer el fichero de nuevo en una tabla, use LOAD DATA INFILE. La sintaxis de las cláusulas FIELDS y LINES es la misma para ambos. Ambas son opcionales, pero FIELDS debe preceder a LINES si se especifican ambas.

Si especifica una cláusula FIELDS , cada una de sus subcláusulas (TERMINATED BY, [OPTIONALLY] ENCLOSED BY, y ESCAPED BY) también es opcional, excepto que debe especificar al menos una de ellas.

Si no especifica una cláusula FIELDS, por defecto es como si hubiera escrito esto:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

Si no especifica una cláusula LINES, por defecto es como si hubiera escrito esto:

```
LINES TERMINATED BY '\n' STARTING BY ''
```

En otras palabras, por defecto LOAD DATA INFILE actúa como sigue al leer la entrada:

- Busca delimitadores de línea como nuevas líneas.
- No ignora ningún prefijo de línea.
- Rompe las líneas en campos con los tabuladores.
- No espera campos entrecomillados dentro de ningún carácter delimitador.
- Interpreta las ocurrencias de tabuladores, nuevas líneas o '\' precedidas por '\' como caracteres literales que son parte de valores de campos.

Por defecto SELECT ... INTO OUTFILE actúa como sigue al escribir la salida:

- Escribe tabuladores entre campos.
- No entrecomilla los campos.
- Usa '\' para escapar las instancias de tabuladores, nuevas líneas o '\' que ocurren entre valores de campos.
- Escribe nuevas líneas al final de las líneas.

Tenga en cuenta que para escribir FIELDS ESCAPED BY '\\', debe escribir dos antibarras para que se interprete como una única antibarra.

Nota: Si ha generado el fichero de texto en un sistema Windows , puede tener que usar LINES TERMINATED BY '\r\n' para leer correctamente el fichero, ya que los programas de Windows típicamente usan dos caracteres como terminadores de línea . Algunos programas como WordPad, pueden usar \r como terminador de línea al escribir ficheros. Para leer tales ficheros, use LINES TERMINATED BY '\r'.

Si todas las líneas que quiere leer tienen un prefijo común que quiere ignorar, puede usar `LINES STARTING BY 'prefix_string'` para ignorar el prefijo (y cualquier cosa antes del mismo). Si una línea no incluye el prefijo, la línea entera se ignora. **Nota** `prefix_string` puede ocurrir en medio de una línea.

Ejemplo:

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
-> INTO TABLE test LINES STARTING BY "xxx";
```

Con esto puede leer en un fichero que contenga algo como:

```
xxx"row",1
something xxx"row",2
```

Y obtener los datos `("row",1)` y `("row",2)`.

La opción `IGNORE number LINES` puede usarse para ignorar líneas al inicio del fichero. Por ejemplo, puede usar `IGNORE 1 LINES` para ignorar una cabecera inicial que contenga los nombres de las columnas:

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
-> INTO TABLE test IGNORE 1 LINES;
```

Cuando usa `SELECT ... INTO OUTFILE` junto con `LOAD DATA INFILE` para escribir datos desde una base de datos en un fichero y luego lee datos del fichero de nuevo en la base de datos, las opciones de tratamiento de fichero y de línea para ambos comandos deben coincidir. De otro modo, `LOAD DATA INFILE` no interpreta los contenidos del fichero correctamente. Suponga que usa `SELECT ... INTO OUTFILE` para escribir un fichero con campos delimitados por comas:

```
mysql> SELECT * INTO OUTFILE 'data.txt'
->          FIELDS TERMINATED BY ','
->          FROM table2;
```

Para leer el fichero delimitado por comas, el comando correcto sería:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
->           FIELDS TERMINATED BY ',';
```

Si en lugar de esto trata de leer en el fichero con el comando mostrado aquí, no funcionaría porque le dice a `LOAD DATA INFILE` que busque tabuladores entre campos:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
->           FIELDS TERMINATED BY '\t';
```

El resultado esperado es que cada línea de entrada se interprete como un único campo.

`LOAD DATA INFILE` puede usarse para leer ficheros obtenidos de fuentes externas. Por ejemplo, un fichero en formato dBASE tiene campos separados por comas y entrecomillados por comillas dobles. Si las líneas en el fichero se terminan con nuevas líneas, el comando mostrado aquí ilustra las opciones de campo y línea que debería usar para cargar el fichero:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
->           FIELDS TERMINATED BY ',' ENCLOSED BY '"'
```

```
->           LINES TERMINATED BY '\n';
```

Cualquiera de las opciones de tratamiento de campo o línea pueden especificarse como una cadena vacía (`''`). Si no está vacía, los valores `FIELDS [OPTIONALLY] ENCLOSED BY` y `FIELDS ESCAPED BY` deben ser un único carácter. Los valores `FIELDS TERMINATED BY`, `LINES STARTING BY`, y `LINES TERMINATED BY` pueden tener más de un carácter . Por ejemplo, para escribir líneas terminadas por parejas de retorno de carro y nueva línea, o para leer un fichero conteniendo tales líneas, especifique una cláusula `LINES TERMINATED BY '\r\n'` .

Para leer un fichero que contenga bromas separadas por líneas consistentes de `%%`, puede hacer lo siguiente

```
mysql> CREATE TABLE jokes
->     (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->     joke TEXT NOT NULL);
mysql> LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
->     FIELDS TERMINATED BY ''
->     LINES TERMINATED BY '\n%%\n' (joke);
```

`FIELDS [OPTIONALLY] ENCLOSED BY` controla el entrecomillado de los campos. Para la salida (`SELECT ... INTO OUTFILE`), si omite la palabra `OPTIONALLY`, todos los campos se delimitan por el carácter `ENCLOSED BY`. Un ejemplo de tal salida (usando coma como el delimitador de campo) se muestra aquí:

```
"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"
```

Si especifica `OPTIONALLY`, el carácter `ENCLOSED BY` se usa sólo para delimitar valores en columnas que tienen datos de cadenas (tales como `CHAR`, `BINARY`, `TEXT`, o `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
```

Tenga en cuenta que la ocurrencias del carácter `ENCLOSED BY` dentro de un campo se escapan mediante un prefijo del carácter `ESCAPED BY`. También tenta en cuenta que si especifica un valor `ESCAPED BY` vacío, es posible generar salida que no puede leerse correctamente con `LOAD DATA INFILE`. Por ejemplo, la salida precedente tendría la siguiente apariencia si el carácter de escape estuviera vacío. Observe que el segundo campo en la cuarta línea contiene una coma siguiendo la delimitación, que (erróneamente) parece que termine el campo:

```
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
```

Para entrada, el carácter `ENCLOSED BY` , si está presente, se elimina del final de los valores de campos . (Esto es cierto se especifique `OPTIONALLY` o no; `OPTIONALLY` no tiene efecto en la interpretación de la entrada.) Las ocurrencias del carácter `ENCLOSED BY` prececdidas por el carater `ESCAPED BY` se interpretan como parte del campo actual.

Si el campo comienza con el carácter `ENCLOSED BY` , las instancias del mismo se reorganizan como terminadores del campo sólo si van seguidas por el campo o la secuencia `TERMINATED BY` . Para evitar

ambigüedad, las ocurrencias del carácter ENCLOSED BY dentro de un campo se pueden doblar y se interpretan como una única instancia del carácter. Por ejemplo, si se especifica ENCLOSED BY '"' , la delimitación se trata como se muestra aquí:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY controla cómo escribir o leer caracteres especiales. Si el carácter FIELDS ESCAPED BY no está vacío, se usa como prefijo para los siguientes caracteres de salida:

- El carácter FIELDS ESCAPED BY
- El carácter FIELDS [OPTIONALLY] ENCLOSED BY
- El primer carácter de los valores FIELDS TERMINATED BY y LINES TERMINATED BY
- ASCII 0 (lo que realmente se escribe a continuación del carácter de escape es '0' en ASCCI, no un byte con valor cero)

Si el carácter FIELDS ESCAPED BY está vacío, no se escapan caracteres y NULL se muestra como NULL, no \N. Probablemente no es una buena idea especificar un carácter de escape vacío, particularmente si los valores de campos en sus datos contienen cualquiera de los caracteres en la lista dada.

Para entrada, si el carácter FIELDS ESCAPED BY no está vacío, las ocurrencias del mismo se eliminan y el siguiente carácter se toma literalmente como parte del campo. Las excepciones son un '0' escapado o 'N' (por ejemplo, \0 o \N si el carácter de escape es '\'). Estas secuencias se interpretan como ASCII NUL (un byte con valor cero) y NULL. Las reglas para tratamiento de NULL se describen posteriormente.

Para más infomación de la sintaxis de escape '\' consulte Sección 9.1, “Valores literales”.

En ciertos casos, las opciones de tratamiento de campos y línea interactúan:

- Si LINES TERMINATED BY es una cadena vacío y FIELDS TERMINATED BY no está vacío, las líneas se terminan con FIELDS TERMINATED BY.
- Si los valores FIELDS TERMINATED BY y FIELDS ENCLOSED BY están vacíois (''), se usa un formato fijo de registro (no delimitado). Con este formato, no se usan delimitadores entre campos (pero puede tener un terminador de línea). En su lugar, los valores de columna se escriben y leen usando los anchos de muestra de las columnas. Por ejemplo, si una columna se declara como INT(7), los valores para la columna se escriben usando campos de siete caracteres. En la entrada, los valores para la columna se obtienen leyendo siete caracteres.

  LINES TERMINATED BY se usa para separar líneas. Si una línea no contiene todos los campos, el resto de columnas se asignan con sus valores por defecto. Si no tiene un terminador de línea, debe asignarlo a ''. En este caso, el fichero de texto debe contener todos los campos para cada registro.

  El formato fijo de registro también afecta al tratamiento de valores NULL, como se describe posteriormente. Tenga en cuenta que el formato de tamaño fijo no funciona si está usando un conjunto de caracteres multi byte.

El tratamiento de valores NULL varía en función de las opciones FIELDS y LINES en uso:

- Para los valores FIELDS y LINES por defecto, NULL se escribe como \N para la salida, y \N para la entrada se lee como NULL (considerando que el carácter ESCAPED BY es '\').

- Si FIELDS ENCLOSED BY no está vacílo, un campo que contenga el literal NULL como valor se lee como el valor NULL . Esto difiere de la palabra NULL delimitada por caracteres FIELDS ENCLOSED BY , que se lee como la cadena 'NULL'.
- Si FIELDS ESCAPED BY está vacío, NULL se escribe como la palabra NULL.
- Con formato fijo de registro (lo que ocurre cuando FIELDS TERMINATED BY y FIELDS ENCLOSED BY están vacíos), NULL se escribe como una cadena vacía. Teng en cuenta que esto hace que ambos valores NULL y cadenas vacías en la tabla sean indistinguibles cuando se escriben en el fichero ya que ambos se escriben como cadenas vacías. Si necesita distinguir entre ambos al leer del fichero, no debe usar el formato de registro fijo.

Algunos casos no son soportados por LOAD DATA INFILE:

- Registros de tamaño fijo (FIELDS TERMINATED BY y FIELDS ENCLOSED BY ambos vacíos) y columnas BLOB o TEXT .
- Si especifica un separador que es igual o prefijo de otro, LOAD DATA INFILE no será capaz de interpretar la entrada correctamente. Por ejemplo, la siguiente cláusula FIELDS causaría problemas:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- Si FIELDS ESCAPED BY está vacío, un valor que contenga una ocurrencia de FIELDS ENCLOSED BY o LINES TERMINATED BY seguido por el valor FIELDS TERMINATED BY causa que LOAD DATA INFILE pare de leer un campo o línea demasiado rápido. Esto ocurre porque LOAD DATA INFILE no puede determinar apropiadamente dónde acaba el campo o línea.

El siguiente ejemplo carga todas las columnas de la tabla persondata :

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

Por defecto, cuando no se proporciona una lista al final de un comando LOAD DATA INFILE, las líneas de entrada se espera que contengan un campo para cada columna de la tabla. Si quiere cargar sólo algunas columnas de una tabla, especifique una lista de columnas:

```
mysql> LOAD DATA INFILE 'persondata.txt'
->           INTO TABLE persondata (col1,col2,...);
```

Debe especificar una lista de columnas si el orden de los campos del fichero de entrada difiere del orden de las columnas en la tabla. De otro modo, MySQL no puede decir cómo hacer coincidir los campos de entrada con las columnas de la tabla.

Antes de MySQL 5.0.3, la lista de columnas debe contener sólo nombres de columnas en la tabla que se carga, y la cláusula SET no se soporta. Desde MySQL 5.0.3, la lista de columnas puede contener nombres de columna o variables y la cláusula SET se soporta. Esto le permite asignar valores de entrada a variables de usuario, y luego realizar transformaciones on estos valores antes de asignar los resultados a las columnas.

Las variables de usuario en la cláusula SET puede usarse de distintos modos. El siguiente ejemplo usa la primera columna en el fichero de datos directamente para el valor de t1.column1, y asigna la segunda columna a una variable de usuario que está sujeta a una operación de división antes de ser usada por el valor de t2.column2:

```
LOAD DATA INFILE 'file.txt'
```

```
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

La cláusula SET puede usarse para proporcionar valores no derivados del fichero de entrada. Los siguientes comandos actualizan column3 con la fecha y hora actuales:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

También puede descartar un valor de entrada asignándolo a una variable de usuario y no asignando la variable a una columna de tabla:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

El uso de la lista de columnas/variables y la cláusula SET está sujeto a las siguientes restricciones:

- Las asignaciones en la cláusula SET deben tener sólo nombres de columna en el lado izquierdo del operador de asignación.
- Puede usar subconsultas en la parte derecha de la asignación de SET. Una subconsulta que retorne un valor a ser asignado a otra coluimna sólo puede ser una subconsulta escalar. Además, no puede usar una subconsulta para seleccionar desde la tabla que se está cargando.
- Las líneas ignoradas por un cláusula IGNORE no se procesan por parta de la lista de columnas/variables o por la cláusula SET .
- Las variables de usuario no pueden usarse al cargar datos con formato de registo ya que las variables de usuario no tienen un ancho de muestra.

Al procesar una línea de entrada, LOAD DATA la divide en campos y usa los valores según la lista de columnas/ variables y la cláusula SET , si están presentes. A continuación se inserta el registro resultante en la tabla. Si hay disparadores BEFORE INSERT o AFTER INSERT para la tabla, se activan antes o después de insertar el registro, respectivamente.

Si una línea de entrada tiene demasiados campos, los campos extra se ignoran y el número de advertencias se incrementa.

Si una línea de entrada no tiene suficientes campos, las columnas de la tabla que no tienen entrada adquieren su valor por defecto. Los valores por defecto se describen en Sección 13.1.5, “Sintaxis de CREATE TABLE”.

Un valor de campo vacío se interpreta de forma distinta que si el valor no está presente:

- Para tipos de cadenas, la columna adquiere la cadena vacía.
- Para tipos numéricos, la columna recibe el valor 0.
- Para tipos de fecha y hora, la columna obtiene el valor “cero” apropiado para el tipo. Consulte Sección 11.3, “Tipos de fecha y hora”.

Estos son los mismos valores que resultan si asigna una cadena vacía explícitamente a un tipo de cadena de caracteres, numérico o de fecha u hora en un comando INSERT o UPDATE statement.

Las columnas TIMESTAMP obtienen la fecha y hora actuales sólo si hay un valor NULL para la columna (esto es, \N), o (para la primera columna TIMESTAMP únicamente) si se omite TIMESTAMP de la lista de campos cuando se especifica una.

LOAD DATA INFILE trata todas las entradas como cadenas, asi que no puede usar valores numéricos para columnas ENUM o SET del modo en que puede hacerlo con comandos INSERT . Todos los valores ENUM y SET deben especificarse como cadenas.

Cuando acaba el comando LOAD DATA INFILE, retorna una cadena de información con el siguiente formato:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Si usa la API de C, puede obtener información acerca del comando mediante la función mysql_info(). Consulte Sección 24.2.3.32, “mysql_info()”.

Las advertencias se producen bajo las mismas circunstancias que cuando los valores se insertan mediante el comando INSERT (consulte Sección 13.2.4, “Sintaxis de INSERT”), excepto que LOAD DATA INFILE también genera advertencias cuando hay muy pocos o demasiados campos en el registro de entrada. Las advertencias no se almacenan en ningún lugar; el número de las mismas puede usarse sólo como indicación de si todo ha ido bien.

En MySQL 5.0, puede usar SHOW WARNINGS para obtener una lista de las primeras max_error_count advertencias como información acerca de qué ha fallado. Consulte Sección 13.5.4.22, “Sintaxis de SHOW WARNINGS”.

13.2.6. Sintaxis de REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

O:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
```

O:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
SELECT ...
```

REPLACE funciona exactamente como INSERT, excepto que si un valor de la tabla tiene el mismo valor que un nuevo registro para un índice PRIMARY KEY o UNIQUE , el antiguo registro se borra antes de insertar el nuevo. Consulte Sección 13.2.4, “Sintaxis de INSERT”.

Tenga en cuenta que a menos que la tabla tenga un índice PRIMARY KEY, o UNIQUE usar un comando REPLACE no tiene sentido. Es equivalente a INSERT, ya que no hay índice para determinar si un nuevo registro duplica otro.

Los valores para todas las columnas se toman de los valores especificados en el comando REPLACE. Cualquier columna no presente adquiere su valor por defecto, como ocurre con INSERT. No puede referirse a valores del registro actual y usarlos en el nuevo registro. Si usa un comando tal como SET

col_name = *col_name* + 1, la referencia al nombre de columna en la parte derecha se trata como DEFAULT(*col_name*), así que es equivalente a SET *col_name* = DEFAULT(*col_name*) + 1.

Para ser capaz de usar REPLACE, debe tener los permisos INSERT y DELETE para la tabla.

El comando REPLACE retorna un contador con el número de registros afectados. Esta es la suma de registros borrados e insertados. Si el contador es 1 para REPLACE de un único registro, se inserta un registro y no se borra ninguno. Si el contador es mayor que 1, uno o más registros se borraron antes de insertar el nuevo. Es posible para un único registro reemplazar más de un registro antiguo si la tabla contiene múltiples índices únicos y el nuevo registro duplica valores para distintos registros antiguos en distintos índices únicos.

El contador de registros afectados hace fácil determinar si REPLACE sólo añadió un registro o si también reemplazo alguno: Compruebe si el contador es 1 (añadido) o mayor (reemplazados).

Si usa la API de C, el contador de registros afectados puede obtenerse usando la función mysql_affected_rows().

Actualmente, no puede reemplzar en una tabla y seleccionar de la misma en una subconsulta.

Aquí sigue en más detalle el algoritmo usado (también se usa con LOAD DATA ... REPLACE):

1. Intenta insertar el nuevo registro en la tabla
2. Mientras falle la inserción debido a error de clave duplicada por clave única o primaria:
   a. Borra de la tabla el registro conflictivo que tiene el valor de clave duplicada
   b. Intenta insertar de nuevo el registro en la tabla

13.2.7. Sintaxis de SELECT

```
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, ...
[INTO OUTFILE 'file_name' export_options
| INTO DUMPFILE 'file_name']
[FROM table_references
[WHERE where_definition]
[GROUP BY {col_name | expr | position}
[ASC | DESC], ... [WITH ROLLUP]]
[HAVING where_definition]
[ORDER BY {col_name | expr | position}
[ASC | DESC] , ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[PROCEDURE procedure_name(argument_list)]
[FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT se usa para recibir registros seleccionados desde una o más tablas. MySQL 5.0 incluye soporte para comandos UNION y subconsultas. Consulte Sección 13.2.7.2, “Sintaxis de UNION” y Sección 13.2.8, “Sintaxis de subconsultas”.

- Cada *select_expr* indicata una columna que quiere recibir.
- *table_references* indicata la tabla o tablas desde la que recibir registros. Su sintaxis se describe en Sección 13.2.7.1, “Sintaxis de JOIN”.

- *where_definition* consiste en la palabra clave WHERE seguida por una expresión que indica la condición o condiciones que deben satisfacer los registros para ser seleccionados.

SELECT también puede usarse para recuperar registros computados sin referencia a ninguna tabla.

Por ejemplo:

```
mysql> SELECT 1 + 1;
-> 2
```

Todas las cláusulas usadas deben darse exactamente en el orden mostrado en la descripción de la sintaxis. Por ejemplo, una cláusula HAVING debe ir tras cualquier cláusula GROUP BY y antes de cualquier cláusula ORDER BY .

-

- No se permite usar un alias de columna en una cláusula WHERE, ya que el valor de columna puede no estar determinado cuando se ejecuta la cláusula WHERE . Consulte Sección A.5.4, “Problemas con alias de columnas”.

- La cláusula FROM *table_references* indica la tabla desde la que recibir registros. Si nombra más de una tabla, está realizando un join, Para información sobre la sintaxis de join, consulte Sección 13.2.7.1, “Sintaxis de JOIN”. Para cada tabla especificada, puede opcionalmente especificar un alias.

  ```
  tbl_name [[AS] alias]
  [[USE INDEX (key_list)]
  | [IGNORE INDEX (key_list)]
  | [FORCE INDEX (key_list)]]
  ```

  El uso de USE INDEX, IGNORE INDEX, FORCE INDEX para dar al optimizador pistas acerca de cómo escoger los indices se describe en Sección 13.2.7.1, “Sintaxis de JOIN”.

  En MySQL 5.0, puede usar SET max_seeks_for_key=*value* como alternativa para forzar a MySQL a que realice escaneos de claves en lugar de escaneos de tabla.

- Puede referirse a una tabla dentro de la base de datos actual como *tbl_name* (dentro de la base de datos actual) , o como *db_name.tbl_name* para referirse a una base de datos explícitamente. Puede referirse a una columna como *col_name*, *tbl_name.col_name*, o *db_name.tbl_name.col_name*. No necesita especificar un prefijo *tbl_name* o *db_name.tbl_name* para una referencia de columna a no ser que la referencia fuese ambígua. Consulte Sección 9.2, “Nombres de bases de datos, tablas, índices, columnas y alias” para ejemplos de ambigüedad que requieran las formas de referencia de columna más explícitas.

- En MySQL 5.0, puede especificar DUAL como nombre de tabla falso en siguaciones donde no se referencian tablas:

  ```
  mysql> SELECT 1 + 1 FROM DUAL;
  -> 2
  ```

  DUAL es una característica puramente de compatibilidad. Otros servidores requieren esta sintaxis.

- Una referencia de tabla puede tener un alias usando *tbl_name* AS *alias_name* o *tbl_name* *alias_name*:

  ```
  mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  ->     WHERE t1.name = t2.name;
  ```

```
mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
->     WHERE t1.name = t2.name;
```

- En la cláusula WHERE , puede usar cualquiera de las funciones que soporta MySQL, escepto para funciones agregadas (resumen). Consulte Capítulo 12, *Funciones y operadores*.

- Las columnas seleccionadas para la salida pueden ser referidas en cláusulas ORDER BY y GROUP BY usando nombres de columnas, alias, o posiciones. Las posiciones de columnas son enteros y comienzan con 1:

```
mysql> SELECT college, region, seed FROM tournament
->     ORDER BY region, seed;
mysql> SELECT college, region AS r, seed AS s FROM tournament
->     ORDER BY r, s;
mysql> SELECT college, region, seed FROM tournament
->     ORDER BY 2, 3;
```

  Para ordenar en orden inverso, añada la palabra clave DESC (descendiente) al nombre de la columna en la cláusula ORDER BY por la que está ordenando. Por defecto es orden ascendente; puede especificarse explícitamente usando la palabra clave ASC.

  El uso de posiciones de columna está obsoleto ya que la sintaxis se ha eliminado del estándar SQL.

- Si usa GROUP BY, los registros de salida se ordenan según las columnas GROUP BY como si tuviera un ORDER BY para las mismas columnas. MySQL 5.0 extiende la cláusula GROUP BY para que pueda especificar ASC y DESC tras las columnas nombradas en la cláusula:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC
```

- MySQL extiende el uso de GROUP BY para permitir seleccionar campos que no se mencionan en la cláusula GROUP BY . Si no obtiene los resultados que espera de la consulta, por favor lea la descripción de GROUP BY en Sección 12.10, “Funciones y modificadores para cláusulas GROUP BY”.

- En MySQL 5.0, GROUP BY permite un modificador WITH ROLLUP . Consulte Sección 12.10.2, “Modificadores de GROUP BY”.

- La cláusula HAVING se aplica casi al final, justo antes de que los elementos se envíen al cliente, sin optimización. (LIMIT se aplica tras HAVING.)

  Antes de MySQL 5.0.2, una cláusula HAVING podía referirse a cualquier columna o alias nombrado en una *`select_expr`* en la lista SELECT o en subconsultas externas, y para funciones agregadas. Sin embargo, el estándar SQL requiere que HAVING debe referirse sólo a columnas en la cláusula GROUP BY o columnas usadas en funciones agregadas. Para acomodar ambos estándars SQL y el comportamiento específico de MySQL en que es capaz de referirse a columnas en la lista SELECT , MySQL 5.0.2 y posteior permite a HAVING referirse a columnas en la lista SELECT , en la cláusula GROUP BY , en subconsultas externas y en funciones agregadas.

  Por ejemplo, el siguiente comando funciona en MySQL 5.0.2 pero produce un error en versiones aneriores:

```
mysql> SELECT COUNT(*) FROM t GROUP BY col1 HAVING col1 = 2;
```

  Si la cláusula HAVING se refiere a una columna ambígua, se muestra una advertencia. En el siguiente comando, col2 es ambíguo porque se usa tanto para un alias como para un nombre de columna:

```
mysql> SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

Se da preferencia al comportamiento SQL estándar, así que si un nombre de columna HAVING se usa en un GROUP BY y como alias de columna en la lista de columnas de salida, se da preferencia a la columna en GROUP BY .

- No use HAVING para elementos que deban estar en la cláusula WHERE . Por ejemplo, no escriba lo siguiente:

```
mysql> SELECT col_name FROM tbl_name HAVING col_name > 0;
```

Escriba esto en su lugar:

```
mysql> SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- La cláusula HAVING puede referirse a funciones de agregación, algo que no puede hacer la cláusula WHERE:

```
mysql> SELECT user, MAX(salary) FROM users
->     GROUP BY user HAVING MAX(salary)>10;
```

(Esto no funciona en versiones antiguas de MySQL.)

- La cláusula LIMIT puede usarse para restringir el número de registros retornados por el comando SELECT. LIMIT tiene uno o dos argumentos numéricos, que deben ser enteros positivos (incluyendo cero).

Con dos argumentos, el primer argumento especifica el desplazamiento del primer registro a retornar. El desplazamiento del registro inicial es 0 (no 1):

```
mysql> SELECT * FROM table LIMIT 5,10;  # Retrieve rows 6-15
```

Por compatibilidad con PostgreSQL, MySQL también soporta la sintaxis LIMIT *row_count* OFFSET *offset*.

Para recibir todos los registros de un desplazamiento hasta el final del conjunto de resultados, puede usar algún número grande para el segundo parámetro. Ete comando recibe todos los registros desde el 96th hasta el último:

```
mysql> SELECT * FROM table LIMIT 95,18446744073709551615;
```

Con un argumento, el valor especifica el número de registros a retornar desde el comienzo del conjunto de resultados:

```
mysql> SELECT * FROM table LIMIT 5;     # Retrieve first 5 rows
```

En otras palabras, LIMIT n es equivalente a LIMIT 0,n.

- La forma SELECT ... INTO OUTFILE 'file_name' de SELECT escribe los registros seleccionados en un fichero. El fichero se crea en el equipo servidor, así que debe tener el permiso FILE para usar esta sintaxis. El fichero no puede existir, que entre otras cosas evita destruir ficheros cruciales tales como /etc/passwd y tablas de la base de datos.

El comando SELECT ... INTO OUTFILE existe principalmente para dejarle volcar una tabla rápidamente en la máquina servidor. Si quiere crear el fichero resultante en un equipo cliente distinto

al equipo servidor, no puede usar SELECT ... INTO OUTFILE. En tal caso, debería usar algún comando como mysql -e "SELECT ..." > file_name en el equipo cliente para generar el fichero.

SELECT ... INTO OUTFILE es el complemento de LOAD DATA INFILE; la sintaxis para la parte export_options del comando consiste en las mismas cláusulas FIELDS y LINES usadas con el comando LOAD DATA INFILE . Consulte Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.

FIELDS ESCAPED BY controla cómo escribir caracteres especiales. Si el carácter FIELDS ESCAPED BY no está vacío, se usa como prefijo para los siguientes caracteres en la salida:

- El carácter FIELDS ESCAPED BY
- El carácter FIELDS [OPTIONALLY] ENCLOSED BY
- El primer carácter de FIELDS TERMINATED BY y LINES TERMINATED BY
- ASCII 0 (que se escribe siguiendo el carácter de escape ASCII '0', no un byte con valor cero)

Si el carácter FIELDS ESCAPED BY está vacío, no hay ningún carácter de escape y NULL se muestra por salida como NULL, no \N. Probablemente no es buena idea especificar un carácter de escape vacío, particularmente si los valores de los campos de sus datos contienen cualquiera de los caracteres en la lista dada.

La razón de lo anterior es que *debe* escapar cualquier carácter FIELDS TERMINATED BY, ENCLOSED BY, ESCAPED BY, o LINES TERMINATED BY para ser capaz de volver a leer el fichero correctamente. ASCII NUL se escapa para hacer más fácil visualizarlo con algunos visores.

El fichero resultante no tiene que estar conforme a la sintaxis SQL, así que nada más debe escaparse.

Este es un ejemplo que produce un fichero en formato de valores separados por comas usado por varios programas:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

- Si usa INTO DUMPFILE en lugar de INTO OUTFILE, MySQL escribe sólo un registro en el fichero, sin ninguna terminación de línea o columna y sin realizar ningún proceso de escape. Esto es útil si quiere almacenar un valor BLOB en un fichero.
- **Nota**: Cualquier fichero creado por INTO OUTFILE o INTO DUMPFILE es modificable por todos los usuarios en el equipo servidor. La razón es que el servidor MySQL no puede crear un fichero con un propietario distinto al usuario que está en ejecución (nunca debe ejecutar mysqld como root por esta y otras razones). El fichero debe ser modificable por todo el mundo para que pueda maminpular sus contenidos.
- Una cláusula PROCEDURE nombra a un procedimiento que debe procesar los datos en el conjunto de resultados. Para un ejemplo, consulte Sección 27.3.1, “Procedimiento Analyse”.
- Si usa FOR UPDATE en un motor de almacenamiento que usa bloqueo de páginas o registros, los registros examinados por la consulta se bloquean para escritura hasta el final de la transacción actual. Usar LOCK IN SHARE MODE crea un bloqueo compartido que evita a otras transacciones actualizar o borrar los registros examinados. Consulte Sección 15.10.5, “Bloquear lecturas SELECT ... FOR UPDATE y SELECT ... LOCK IN SHARE MODE”.

Tras la palabra clave SELECT , puede usar un número de opciones que afectan la operación del comando.

Las opciones ALL, DISTINCT, and DISTINCTROW especifican si deben retornarse los registros duplicados. Si no se da ninguna de estas opciones, por defecto es ALL (se retornan todos los registros coincidentes). DISTINCT y DISTINCTROW son sinónimos y especifican que los registros duplicados en el conjunto de resultados deben borrarse.

HIGH_PRIORITY, STRAIGHT_JOIN, y opciones que comiencen con SQL_ son extensiones de MySQL al estándar SQL.

- HIGH_PRIORITY da a SELECT prioridad más alta que un comando que actualice una tabla. Debe usar esto sólo para consultas que son muy rápidas y deben realizarse una vez. Una consulta SELECT HIGH_PRIORITY que se realiza mientras la tabla está bloqueada para lectura se ejectua incluso si hay un comando de actualización esperando a que se libere la tabla.

  HIGH_PRIORITY no puede usarse con comandos SELECT que sean parte de una UNION.

- STRAIGHT_JOIN fuerza al optimizador a hacer un join de las tablas en el orden en que se listan en la cláusula FROM . Puede usarlo para acelerar una consulta si el optimizador hace un join con las tablas en orden no óptimo. Consulte Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)”. STRAIGHT_JOIN también puede usarse en la lista *table_references* . Consulte Sección 13.2.7.1, “Sintaxis de JOIN”.

- SQL_BIG_RESULT puede usarse con GROUP BY o DISTINCT para decir al optimizador que el conjunto de resultados tiene muchos registros. En este caso, MySQL usa directamente tablas temporales en disco si son necesarias con una clave en los elementos GROUP BY .

- SQL_BUFFER_RESULT fuerza a que el resultado se ponga en una tabla temporal . Esto ayuda a MySQL a liberar los bloqueos de tabla rápidamente y ayuda en casos en que tarda mucho tiempo en enviar el resultado al cliente.

- SQL_SMALL_RESULT puede usarse con GROUP BY o DISTINCT para decir al optimizador que el conjunto de resultados es pequeño. En este caso, MySQL usa tablas temporales rápidas para almacenar la tabla resultante en lugar de usar ordenación. En MySQL 5.0, esto no hará falta normalmente.

- SQL_CALC_FOUND_ROWS le dice a MySQL que calcule cuántos registros habrán en el conjunto de resultados, sin tener en cuenta ninguna cláusula LIMIT. El número de registros pueden encontrarse con SELECT FOUND_ROWS(). Consulte Sección 12.9.3, “Funciones de información”.

- SQL_CACHE le dice a MySQL que almacene el resultado de la consulta en la caché de consultas si está usando un valor de query_cache_type de 2 o DEMAND. Para una consulta que use UNION o subconsultas, esta opción afecta a cualquier SELECT en la consulta. Consulte Sección 5.12, “La caché de consultas de MySQL”.

- SQL_NO_CACHE le dice a MySQL que no almacene los resultados de consulta en la caché de consultas. Consulte Sección 5.12, “La caché de consultas de MySQL”. Para una consulta que use UNION o subconsultas esta opción afecta a cualquier SELECT en la consulta.

13.2.7.1. Sintaxis de JOIN

MySQL soporta las siguientes sintaxis de JOIN para la parte *table_references* de comandos SELECT y DELETE y UPDATE de múltiples tablas:

```
table_reference, table_reference
table_reference [INNER | CROSS] JOIN table_reference [join_condition]
```

```
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference join_condition
table_reference NATURAL [LEFT [OUTER]] JOIN table_reference
{ ON table_reference LEFT OUTER JOIN table_reference
ON conditional_expr }
table_reference RIGHT [OUTER] JOIN table_reference join_condition
table_reference NATURAL [RIGHT [OUTER]] JOIN table_reference
```

table_reference se define como:

```
  tbl_name [[AS] alias]
  [[USE INDEX (key_list)]
  | [IGNORE INDEX (key_list)]
  | [FORCE INDEX (key_list)]]
```

join_condition se define como:

```
ON conditional_expr | USING (column_list)
```

Generalmente no debería tener ninguna condición en la parte ON que se usa para restringir qué registros desea en el conjunto de resultados, pero en su lugar especificar esas condiciones en la cláusula WHERE . Hay excepciones a esta regla.

La sintaxis { OJ ... LEFT OUTER JOIN ...} mostrada en la lista precedente existe sólo por compatibilidad con ODBC.

- Puede poner un alias en una referencia de tabla usando *tbl_name* AS *alias_name* o *tbl_name alias_name*:

  ```
  mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  ->        WHERE t1.name = t2.name;
  mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
  ->        WHERE t1.name = t2.name;
  ```

- El condicional ON es cualquier expresión condicional de la forma que puede usarse en una cláusula WHERE .

- Si no hay ningún registro coincidiente para la tabla de la derecha en la parte ON o USING en un LEFT JOIN, se usa un registro con todos las columnas a NULL para la tabla de la derecha. Puede usar este hecho para encontrar registros en una tabla que no tengan contraparte en otra tabla:

  ```
  mysql> SELECT table1.* FROM table1
  ->        LEFT JOIN table2 ON table1.id=table2.id
  ->        WHERE table2.id IS NULL;
  ```

  Este ejemplo encuentra todos los registros en table1 con un valor id no presente en table2 (esto es, todos los registros en table1 sin registro correspondiente en table2). Esto asume que table2.id se declara NOT NULL. Consulte Sección 7.2.9, “Cómo optimiza MySQL los LEFT JOIN y RIGHT JOIN”.

- La cláusula USING (*column_list*) muestra una lista de columnas que deben existir en ambas tablas. Las siguientes dos cláusulas son semánticamente idénticas:

  ```
  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
  ```

- El NATURAL [LEFT] JOIN de dos tablas se define semánticamente equivalente a un INNER JOIN o LEFT JOIN con una cláusula USING que nombra todas las columnas que existen en ambas tablas.

- INNER JOIN y , (coma) son semánticamente equivalentes en la ausencia de una condicicón de join: ambos producen un producto Cartesiano entre las tablas especificadas (esto es, cada registro en la primera tabla se junta con cada registro en la segunda tabla).
- RIGHT JOIN funciona análogamente a LEFT JOIN. Para mantener el código portable entre bases de datos, se recomienda que use LEFT JOIN en lugar de RIGHT JOIN.
- STRAIGHT_JOIN es idéntico a JOIN, excepto que la tabla de la izquierda se lee siempre antes que la de la derecha. Esto puede usarse para aquéllos casos (escasos) en que el optimizador de join pone las tablas en orden incorrecto.

Puede proporcionar pistas de qué índice debe usar MySQL cuando recibe información de una tabla. Especificando USE INDEX (key_list), puede decirle a MySQL que use sólo uno de los posibles índices para encontrar registros en la tabla. La sintaxis alternativa IGNORE INDEX (key_list) puede usarse para decir a MySQL que no use algún índice particular. Estos trucos son útiles si EXPLAIN muestra que MySQL está usando el índice incorrecto de la lista de posibles índices.

También puede usar FORCE INDEX, que actúa como USE INDEX (*key_list*) pero con la adición que un escaneo de tabla se asume como operación *muy* cara. En otras palabras, un escaneo de tabla se usa sólo si no hay forma de usar uno de los índices dados para encontrar registros en la tabla.

USE KEY, IGNORE KEY, y FORCE KEY son sinónimos de USE INDEX, IGNORE INDEX, y FORCE INDEX.

Nota: USE INDEX, IGNORE INDEX, y FORCE INDEX sólo afecta los índices usados cuando MySQL decide cómo encontrar registros en la tabla y cómo hacer el join. No afecta si un índice está en uso cuando se resuelve unORDER BY o GROUP BY.

Algunos ejemplos de join:

```
mysql> SELECT * FROM table1,table2 WHERE table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 USING (id);
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
->          LEFT JOIN table3 ON table2.id=table3.id;
mysql> SELECT * FROM table1 USE INDEX (key1,key2)
->          WHERE key1=1 AND key2=2 AND key3=3;
mysql> SELECT * FROM table1 IGNORE INDEX (key3)
->          WHERE key1=1 AND key2=2 AND key3=3;
```

Consulte Sección 7.2.9, “Cómo optimiza MySQL los LEFT JOIN y RIGHT JOIN”.

13.2.7.2. Sintaxis de UNION

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
[UNION [ALL | DISTINCT]
SELECT ...]
```

UNION se usa para combinar el resultado de un número de comandos SELECT en un conjunto de resultados.

Las columnas seleccionadas lisatadas en posiciones correspondientes de cada comando SELECT deben tener el mismo tipo. (Por ejemplo, la primera columna seleccionada por el primer comando debe tener el

mismo tipo que la primer columna seleccionada por otros comandos.) Los nombres de columna usados por el primer comando SELECT se usan como nombres de columna para los resultados retornados.

Los comandos SELECT son comandos select normales, pero con las siguientes restricciones:

- Sólo el último comando SELECT puede usar INTO OUTFILE.
- HIGH_PRIORITY no puede usarse con comandos SELECT que sean parte de una UNION. Si lo especifica para el primer SELECT, no tiene efecto. Si lo especifica para cualquier SELECT posterior, aparece un error de sintaxis.

Si no usa la palabra clave ALL para UNION, todos los registros retornados son únicos, como si hubiera hecho un DISTINCT para el conjunto de resultados total. Si especifica ALL, obtiene todos los registros coincidentes de todos los comandos SELECT usados.

La palabra clave DISTINCT es una palabra opcional que no tiene efecto, pero se permite en la sintaxis como requiere el estándar SQL . (En MySQL, DISTINCT representa el comportamiento por defecto de una union.)

En MySQL 5.0, puede mezclar UNION ALL y UNION DISTINCT en la misma consulta. Tipos de UNION mezclados se tratan de forma que una unión DISTINCT sobreescribe cualquier unión ALL a su izquierda. Una unión DISTINCT puede producirse explícitamente usando UNION DISTINCT o implícitamente usando UNION sin palabra clave DISTINCT o ALL a continuación.

Si quiere usar una cláusula ORDER BY o LIMIT para ordenar o limitar el resultado UNION entero, ponga entre paréntesis los comandos SELECT individuales y ponga el ORDER BY o LIMIT tras el último. El siguiente ejemplo usa ambas cláusulas:

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1)
UNION
(SELECT a FROM tbl_name WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

Este tipo de ORDER BY no puede usar referencias de columnas que incluyan un nombre de columna (esto es, nombres en formato *tbl_name.col_name*). En su lugar, proporcione un alias de columna al primer comando SELECT y refiérase al alias en el ORDER BY, o a la columna en el ORDER BY usando su posición de columna. (Un alias es preferible porque el uso de la posición de la columna está obsoleto.)

Para aplicar ORDER BY o LIMIT a un SELECT individual, ponga la cláusula dentro de los paréntesis alrededor del SELECT:

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM tbl_name WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

Los ORDER BY para comandos SELECT individuales entre paréntesis tienen efecto sólo al combinarlos con LIMIT. De otro modo, el ORDER BY se optimiza a parte.

En MySQL 5.0, los tipos y longitudes de las columnas en el conjunto de resultados de una UNION tienen en cuenta los valores recibidos por todos los comandos SELECT. Por ejemplo, considere lo siguiente:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
```

```
| bbbbbbbbbb    |
+---------------+
```

(En alguna versión anterior de MySQL, el segundo registro se habría truncado a una longitud de 1.)

13.2.8. Sintaxis de subconsultas

Una subconsulta es un comando `SELECT` dentro de otro comando.

MySQL 5.0 soporta todas las formas de subconsultas y operaciones que requiere el estándar SQL, así como algunas características específicas de MySQL.

Aquí hay un ejemplo de subconsulta:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

En este ejemplo, `SELECT * FROM t1 ...` es la *consulta externa* (o *comando externo*), y `(SELECT column1 FROM t2)` es la *subconsulta*. Decimos que la subconsulta está *anidada* dentro de la consulta exterior, y de hecho, es posible anidar subconsultas dentro de otras subconsultas hasta una profundidad considerable. Una subconsulta debe siempre aparecer entre paréntesis.

Las principales ventajas de subconsultas son:

- Permiten consultas *estructuradas* de forma que es posible aislar cada parte de un comando.
- Proporcionan un modo alternativo de realizar operaciones que de otro modo necesitarían joins y uniones complejos.
- Son, en la opinión de mucha gente, leíbles. De hecho, fue la innovación de las subconsultas lo que dio a la gente la idea original de llamar a SQL “Structured Query Language.”

Aquí hay un comando de ejemplo que muestra los puntos principales de la sintaxis de subconsultas como especifica el estándar SQL y soporta MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
(SELECT COUNT(*) /* no hint */ FROM t2
WHERE NOT EXISTS
(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)));
```

Una subconsulta puede retornar un escalar (un valor único), un registro, una columna o una tabla (uno o más registros de una o más columnas). Éstas se llaman consultas de escalar, columna, registro y tabla. Las subconsultas que retornan una clase particular de resultado a menudo pueden usarse sólo en ciertos contextos, como se describe en las siguientes secciones.

Hay pocas restricciones sobre los tipos de comandos en que pueden usarse las subconsultas. Una subconsulta puede contener cualquiera de las palabras claves o cláusulas que puede contener un `SELECT` ordinario: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, trucos de índices, constructores `UNION` , comentarios, funciones, y así.

Una restricción es que el comando exterior de una subconsulta debe ser: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, o `DO`. Otra restricción es que actualmente no puede modificar una tabla y seleccionar de la misma tabla en la subconsulta. Esto se aplica a comandos tales como `DELETE`, `INSERT`, `REPLACE`, y

`UPDATE`. Una discusión más comprensible de las restricciones en las subconsultas se da en Apéndice H, *Restricciones en características de MySQL*.

13.2.8.1. La subconsulta, como un operador sobre valores escalares

En su forma más sencilla, una subconsulta es una subconsulta escalar que retorna un único valor. Una subconsulta escalar es un operando simple, y puede usarlo prácticamente en cualquier sitio en que un valor de columna o literal sea legal, y puede esperar que tenga las características que tienen todos los operandos: un tipo de datos, una longitud, una indicación de si puede ser `NULL`, etcétera. Por ejemplo:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

La subconsulta en este `SELECT` retorna un valor único (`'abcde'`) que tiene un tipo de datos `CHAR`, una longitud de 5, un conjunto de caracteres y una colación iguales a la que había por defecto cuando se realizó el `CREATE TABLE` , y una indicación que el valor en la columna puede ser `NULL`. De hecho, casi todas las consultas pueden ser `NULL`. Si la tabla usada en este ejemplo estuviese vacía, la tabla de la subconsulta sería `NULL`.

Hay algunos contextos en que una subconsulta escalar no se puede usar. Si un comando permite sólo un valor literal, no puede usar una subconsulta. Por ejemplo, `LIMIT` necesita argumentos enteros, y `LOAD DATA` necesita una cadena con un nombre de fichero. No puede usar subconsultas para proporcionar estos valores.

Cuando vea los ejemplos en las siguientes secciones que contengan el constructor `(SELECT column1 FROM t1)`, imagine que su própio código contiene construcciones mucho más diversas y complejas.

Por ejemplo, suponga que hacemos dos tablas:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Luego realice `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

El resultado es `2` ya que hay un registro en `t2` que contiene una columna `s1` con un valor de `2`.

Una subconsulta escalar puede ser parte de una expresión. No olvide los paréntesis, incluso si la subconsulta es un operando que proporciona un argumento para una función. Por ejemplo:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.8.2. Uso de subconsultas en subconsultas

El uso más común de una subconsulta es de la forma:

```
non_subquery_operand comparison_operator (subquery)
```

Donde *`comparison_operator`* es uno de estos operadores:

```
```

```
=  >  <  >=  <=  <>
```

Por ejemplo:

```
  ... 'a' = (SELECT column1 FROM t1)
```

Tiempo atrás el único sitio legal para una subconsulta era la parte derecha de la comparación, y puede encontrar algunos SGBDs que insistan en ello.

He aquí un ejemplo de una comparación común de subconsultas que no puede hacerse mediante un join. Encuentra todos los valores en la tabla `t1` que son iguales a un valor máximo en la tabla `t2`:

```
SELECT column1 FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Aquí hay otro ejemplo, que de nuevo es imposible de hacer con un join ya que involucra agregación para una de las tablas. Encuentra todos los registros en la tabla `t1` que contengan un valor que ocurre dos veces en una columna dada:

```
SELECT * FROM t1 AS t
WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

Para una comparación realizada con uno de estos operadores, la subconsulta debe retornar un escalar, con la excepción que `=` puede usarse con subconsultas de registro. Consulte Sección 13.2.8.5, “Subconsultas de registro”.

13.2.8.3. Subconsultas con `ANY`, `IN` y `SOME`

Sintaxis:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

La palabra clave `ANY` , que debe seguir a un operador de comparación, significa “return `TRUE` si la comparación es `TRUE` para `ANY` (cualquiera) de los valores en la columna que retorna la subconsulta.” Por ejemplo:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suponga que hay un registro en una tabla `t1` que contiene `(10)`. La expresión es `TRUE` si la tabla `t2` contiene `(21,14,7)` ya que hay un valor `7` en `t2` que es menor que `10`. La expresión es `FALSE` si la tabla `t2` contiene `(20,10)`, o si la tabla `t2` está vacía. La expresión es `UNKNOWN` si la tabla `t2` contiene `(NULL,NULL,NULL)`.

La palabra `IN` es un alias para `= ANY`. Por lo tanto, estos dos comandos son lo mismo:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

Sin embargo, `NOT IN` no es un alias para `<> ANY`, sino para `<> ALL`. Consulte Sección 13.2.8.4, “Subconsultas con `ALL`”.

La palabra `SOME` es un alias para `ANY`. Por lo tanto, estos dos comandos son el mismo:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

El uso de la palabra `SOME` es raro, pero este ejemplo muestra cómo puede ser útil. Para la mayoría de gente, la frase en inglés “a is not equal to any b” significa “there is no b which is equal to a,” pero eso no es lo que quiere decir la sintaxis SQL. La sintaxis significa “there is some b to which a is not equal.” Usando `<> SOME` en su lugar ayuda a asegurar que todo el mundo entiende el significado de la consulta.

13.2.8.4. Subconsultas con `ALL`

Sintaxis:

```
operand comparison_operator ALL (subquery)
```

La palabra `ALL`, que debe seguir a un operador de comparación, significa “return `TRUE` si la comparación es `TRUE` para `ALL` todos los valores en la columna que retorna la subconsulta.” Por ejemplo:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suponga que hay un registro en la tabla `t1` que contiene `(10)`. La expresión es `TRUE` si la tabla `t2` contiene `(-5,0,+5)` ya que `10` es mayor que los otros tres valores en `t2`. La expresión es `FALSE` si la tabla `t2` contiene `(12,6,NULL,-100)` ya que hay un único valor `12` en la tabla `t2` mayor que `10`. La expresión es `UNKNOWN` si la tabla `t2` contiene `(0,NULL,1)`.

Finalmente, si la tabla `t2` está vacía, el resultado es `TRUE`. Puede pensar que el resultado debería ser `UNKNOWN`, pero lo sentimos, es `TRUE`. Así, aunque extraño, el siguiente comando es `TRUE` cuando la tabla `t2` está vacía:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

Pero este comando es `UNKNOWN` cuando la tabla `t2` está vacía:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

Además, el siguiente comando es `UNKNOWN` cuando la tabla `t2` está vacía:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

En general, *las tablas con valores `NULL`* y *las tablas vacías* son *casos extremos*. Al escribir código para subconsultas, siempre considere si ha tenido en cuenta estas dos posibilidades.

`NOT IN` es un alias para `<> ALL`. Por lo tanto, estos dos comandos son equivalentes:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.8.5. Subconsultas de registro

La discusión en este punto ha sido entre subconsultas escalares o de columnas, esto es, subcolumnas que retornan un único valor o una columna de valores. Una *subconsulta de registro* es una variante de subconsulta que retorna un único registro y por lo tanto retorna más de un valor de columna. Aquí hay dos ejemplos:

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

Las consultas aquí son ambas `TRUE` si la tabla `t2` tiene un registro en que `column1 = 1` y `column2 = 2`.

Las expresiones `(1,2)` y `ROW(1,2)` a veces se llaman *constructores de registros*. Ambos son equivalentes. También son legales en otros contextos. Por ejemplo, los siguientes dos comandos son semánticamente equivalentes (aunque actualmente sólo puede optimizarse el segundo):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

El uso normal de constructores de registros, sin embargo, es para comparaciones con subconsultas que retornan dos o más columnas. Por ejemplo, la siguiente consulta responde a la petición, “encuentra todos los registros en la tabla `t1` que también existen en la tabla `t2`”:

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
(SELECT column1,column2,column3 FROM t2);
```

13.2.8.6. `EXISTS` y `NOT EXISTS`

Si una subconsulta retorna algún registro, entonces `EXISTS subquery` es `TRUE`, y `NOT EXISTS subquery` es `FALSE`. Por ejemplo:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Tradicionalmente, una subconsulta `EXISTS` comienza con `SELECT *`, pero puede comenzar con `SELECT 5` o `SELECT col1` o nada. MySQL ignora la lista `SELECT` en tales subconsultas, así que no hace distinción.

Para el ejemplo precedente, si `t2` contiene algún registro, incluso registros sólo con valores `NULL` entonces la condición `EXISTS` es `TRUE`. Este es un ejemplo poco probable, ya que prácticamente siempre una subconsulta `[NOT] EXISTS` contiene correlaciones. Aquí hay algunos ejemplos más realistas:

- ¿Qué clase de tienda hay en una o más ciudades?

  ```
  SELECT DISTINCT store_type FROM Stores
  WHERE EXISTS (SELECT * FROM Cities_Stores
  WHERE Cities_Stores.store_type = Stores.store_type);
  ```

- ¿Qué clase de tienda no hay en ninguna ciudad?

  ```
  SELECT DISTINCT store_type FROM Stores
  WHERE NOT EXISTS (SELECT * FROM Cities_Stores
  WHERE Cities_Stores.store_type = Stores.store_type);
  ```

- ¿Qué clase de tienda hay en todas las ciudades?

  ```
  SELECT DISTINCT store_type FROM Stores S1
  WHERE NOT EXISTS (
  SELECT * FROM Cities WHERE NOT EXISTS (
  ```

```
SELECT * FROM Cities_Stores
WHERE Cities_Stores.city = Cities.city
AND Cities_Stores.store_type = Stores.store_type));
```

El último ejemplo es un doblemente anidado `NOT EXISTS` . Esto es, tiene una cláusula `NOT EXISTS` dentro de otra `NOT EXISTS`. Formalmente, responde a la pregunta "¿existe una ciudad con una tienda que no esté en `Stores`?" Sin embargo, es más fácil decir que un `NOT EXISTS` responde a la pregunta "¿es x `TRUE` para todo y?"

13.2.8.7. Subconsultas correlacionadas

Una *subconsulta correlacionada* es una subconsulta que contiene una referencia a una tabla que también aparece en la consulta exterior. Por ejemplo:

```
SELECT * FROM t1 WHERE column1 = ANY
(SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

Tenga en cuenta que la subconsulta contiene una referencia a una columna de `t1`, incluso aunque la cláusula `FROM` de la subconsulta no menciona una tabla `t1`. Por lo tanto, MySQL busca fuera de la subconsulta y encuentra `t1` en la consulta externa.

Suponga que la tabla `t1` contiene un registro en que `column1 = 5` y `column2 = 6`; mientras, la tabla `t2` contiene un registro en que `column1 = 5` y `column2 = 7`. La expresión `... WHERE column1 = ANY (SELECT column1 FROM t2)` sería `TRUE`, pero en este ejemplo, la cláusula `WHERE` dentro de la subconsulta es `FALSE` (ya que `(5,6)` no es igual a `(5,7)`), así que la subconsulta como un todo es `FALSE`.

Regla de visibilidad: MySQL evalúa desde dentro hacia fuera. Por ejemplo:

```
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));
```

En este comando, `x.column2` debe ser una columna en la tabla `t2` ya que `SELECT column1 FROM t2 AS x ...` renombra `t2`. No hay una columna en la tabla `t1` porque `SELECT column1 FROM t1 ...` es una consulta externa que está *demasiado afuera*.

Para subconsultas en cláusulas `HAVING` u `ORDER BY` , MySQL busca nombres de columna en la lista de selección exterior.

Para ciertos casos, una subconsulta correlacionada es óptima. Por ejemplo:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

De otro modo, son ineficientes y lentas. Reescribir la consulta como un join puede mejorar el rendimiento.

Las subconsultas correlacionadas no pueden referirse a los resultados de funciones agregadas de la consulta exterior.

13.2.8.8. Subconsultas en la cláusula `FROM`

Las subconsultas son legales en la cláusula `FROM` de un comando `SELECT`. La sintaxis que vería es:

```
SELECT ... FROM (subquery) [AS] name ...
```

La cláusula `[AS] name` es obligatoria, ya que cada tabla en la cláusula `FROM` debe tener un nombre. Cualquier columna en la lista selecta de la `subquery` debe tener nombre único. Puede encontrar esta sintaxis descrita en este manual, dónde se usa el término “tablas derivadas.”

Asuma que tiene la tabla:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Aquí se muestra cómo usar una subconsulta en la cláusula `FROM` usando la tabla de ejemplo:

```
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;
```

Resultado: `2, '2', 4.0`.

Aquí hay otro ejemplo: suponga que quiere conocer la media de un conjunto de sumas para una tabla agrupada. Esto no funcionaría:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

Sin embargo, esta consulta proporciona la información deseada:

```
SELECT AVG(sum_column1)
FROM (SELECT SUM(column1) AS sum_column1
FROM t1 GROUP BY column1) AS t1;
```

Tenga en cuenta que el nombre de columna usado dentro de la subconsultas `(sum_column1)` se reconoce en la consulta exterior.

Las subconsultas en la cláusula `FROM` pueden retornar un escalar, columna, registro o tabla. De momento, las subconsultas en la cláusula `FROM` no pueden ser subconsultas correladas.

Las subconsultas en la cláusula `FROM` se ejecutan incluso para el comando `EXPLAIN` (esto es, se construyen las tablas temporales derivadas). Esto ocurre porque las consultas de niveles superiores necesitan información acerca de todas las tablas durante la fase de optimización.

13.2.8.9. Errores en subconsultas

Hay algunos retornos de error nuevos que se aplican sólo a subconsultas. Esta sección los agrupa ya que revisarlos ayuda a recordar algunos puntos importantes.

- Número incorrecto de columnas de la subconsulta:

  ```
  ERROR 1241 (ER_OPERAND_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  Este error ocurre en casos como este:

  ```
  SELECT (SELECT column1, column2 FROM t2) FROM t1;
  ```

Se permite usar una subconsulta que retorne múltiples columnas, si el propósito es la comparación. Consulte Sección 13.2.8.5, “Subconsultas de registro”. Sin embargo, en otros contextos, la subconsulta debe ser un operando escalar.

- Número incorrecto de registros de la subconsulta:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

Este error ocurre de comandos en que la subconsulta retorna más de un registro. Considere el siguiente ejemplo:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

Si `SELECT column1 FROM t2` retorna sólo un registro la consulta anterior funcionará. Si la subconsulta retorna más de un registro, ocurre el error 1242 . En ese caso, la consulta debe reescribirse como:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Tabla usada incorrectamente en la subconsulta:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

Este error ocurre en casos como el siguiente:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

Puede usar una subconsulta para asignaciones dentro del comando `UPDATE`, ya que las subconsultas son legales en los comandos `UPDATE` y `DELETE` así como en los `SELECT`. Sin embargo, no puede usar la misma tabla (en este caso la tabla `t1`) para la cláusula `FROM` de la subconsulta y el objetivo a actualizar.

Para motores transaccionales, el fallo de una subconsulta provoca que falle el comando entero. Para motores no transaccionales, las modificaciones de datos hechas antes de encontrar el error se preservan.

13.2.8.10. Optimizar subconsultas

El desarrollo está en marcha, por lo que no hay trucos de optimización fiables a largo plazo. Algunos trucos interesantes que puede usar son:

- Use cláusulas de subconsulta que afecten al número u orden de los registros en la subconsulta. Por ejemplo:

```
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);
```

- Reemplace un join con una subconsulta. Por ejemplo, pruebe:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
SELECT column1 FROM t2);
```

  En lugar de:

```
SELECT DISTINCT t1.column1 FROM t1, t2
WHERE t1.column1 = t2.column1;
```

- Algunas subconsultas pueden transformarse en joins por compatibilidad con versiones anteriores de MySQL que no soportan subconsultas. Sin embargo, en algunos casos, incluso en MySQL 5.0, convertir una subconsulta en un join puede mejorar el rendimiento. Consulte Sección 13.2.8.11, “Re-escribir subconsultas como joins en versiones de MySQL anteriores”.

- Mueva las cláusulas desde fuera hacia dentro en la subconsulta. Por ejemplo , use esta consulta:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

  En lugar de:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

  Otro ejemplo. Use esta consulta:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

  En lugar de:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use una subconsulta de registro en lugar de una subconsulta correlacionada . Por ejemplo, use:

```
SELECT * FROM t1
WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

  En lugar de:

```
SELECT * FROM t1
WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` en lugar de `a <> ALL (...)`.

- Use `x = ANY (table containing (1,2))` en lugar de `x=1 OR x=2`.

- Use `= ANY` en lugar de `EXISTS`.

- Para subconsultas no correlacionadas que siempre retornan un registro, `IN` siempre es más lento que `=`. Por ejemplo, use esta consulta:

```
SELECT * FROM t1 WHERE t1.col_name
```

```
= (SELECT a FROM t2 WHERE b = some_const);
```

En lugar de:

```
SELECT * FROM t1 WHERE t1.col_name
IN (SELECT a FROM t2 WHERE b = some_const);
```

Estos trucos pueden hacer que los programas vayan más rápidos o lentos. Usar recursos MySQL como la función `BENCHMARK()` es una buena idea para ver cuáles funcionan.

Algunas optimizaciones que realiza MySQL son:

- MySQL ejecuta subconsultas no correlacionadas sólo una vez. Use `EXPLAIN` para asegurar que una subconsulta dada realmente no está correlacionada.
- MySQL reescribe subconsultas `IN`, `ALL`, `ANY`, y `SOME` para aprovechar que las columnas de la lista de select de la subconsulta está indexada.
- MySQL reemplaza subconsultas de la siguiente forma con una función de búsqueda de índice, que `EXPLAIN` describe como tipo especial de join (`unique_subquery` o `index_subquery`):

  ```
  ... IN (SELECT indexed_column FROM single_table ...)
  ```

- MySQL mejora expresiones de la siguiente forma con una expresión que involucre `MIN()` o `MAX()`, a no ser que hayan involucrados valores `NULL` o conjuntos vacíos:

  ```
  value {ALL|ANY|SOME} {> | < | >= | <=} (non-correlated subquery)
  ```

  Por ejemplo, esta cláusula `WHERE`:

  ```
  WHERE 5 > ALL (SELECT x FROM t)
  ```

  puede tratarse por el optimizador como:

  ```
  WHERE 5 > (SELECT MAX(x) FROM t)
  ```

Hay un capítulo titulado “Cómo transforma las subconsultas MySQL” en el manual MySQL Internals Manual. Puede obtener este documento descargando el paquete fuente MySQL y buscando un fichero llamado `internals.texi` en el directorio `Docs` .

13.2.8.11. Re-escribir subconsultas como joins en versiones de MySQL anteriores

En versiones prévias de MySQL (anteriores a la MySQL 4.1), sólo se soportaban consultas anidadas de la forma `INSERT ... SELECT ...` y `REPLACE ... SELECT ...`. Este no es el caso en MySQL 5.0, pero es cierto que hay a veces otras formas de testear la pertenencia a un grupo de valores. También es cierto que en algunas ocasiones, no es sólo posible reescribir una consulta sin una subconsulta, sino que puede ser más eficiente hacerlo que usar subconsultas. Una de las técnicas disponibles es usar el constructor `IN()` :

Por ejemplo, esta consulta:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Puede reescribirse como:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

Las consultas:

```
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);
```

Pueden reescribirse usando `IN()`:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id
WHERE table2.id IS NULL;
```

Un `LEFT [OUTER] JOIN` puede ser más rápido que la subconsulta equivalente ya que el servidor puede ser capaz de optimizarlo mejor — este es un hecho no específico de MySQL Server . Antes de SQL-92, los outer joins no existían, así que las subconsultas eran el único modo de hacer ciertas cosas. Hoy, MySQL Server y otros sistemas de bases de datos ofrecen un ámplio rango de tipos de outer join.

MySQL Server soporta comandos `DELETE` para múltiples tablas que pueden usarse para borrar registros basándose en la información de una tabla o de varias al mismo tiempo. Los comandos `UPDATE` para múltiples tablas también se soportan en MySQL 5.0.

13.2.9. Sintaxis de TRUNCATE

```
TRUNCATE TABLE tbl_name
```

`TRUNCATE TABLE` vacía una tabla completamente. Lógicamente, esto es equivalente a un comando `DELETE` que borre todos los registros, pero hay diferencias prácticas bajo ciertas circunstáncias.

Para `InnoDB` antes de la versión 5.0.3, `TRUNCATE TABLE` se mapea a `DELETE`, así que no hay diferencia. A partir de MySQL/InnoDB-5.0.3, está disponible `TRUNCATE TABLE` muy rápido. La operación se mapea a `DELETE` si hay restricciones de clave foránea que referencien la tabla.

Para otros motores, `TRUNCATE TABLE` difiere de `DELETE FROM` en los siguientes puntos en MySQL 5.0:

- Las operaciones de truncado destruyen y recrean la tabla, que es mucho más rápido que borrar registros uno a uno.
- Las operaciones de truncado no son transaccionales; ocurre un error al intentar un truncado durante una transacción o un bloqueo de tabla.
- No se retorna el número de registros borrados.
- Mientras el fichero de definición de la tabla *`tbl_name`*`.frm` sea válido, la tabla puede recrearse como una vacía con `TRUNCATE TABLE`, incluso si los ficheros de datos o de índice se han corrompido.
- El tratador de tablas no recuerda el último valor `AUTO_INCREMENT` usado, pero empieza a contar desde el principio. Esto es cierto incluso para `MyISAM` y `InnoDB`, que normalmente no reúsan valores de secuencia.

`TRUNCATE TABLE` es una extensión de Oracle SQL adoptada en MySQL.

13.2.10. Sintaxis de UPDATE

Sintaxis para una tabla:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2 ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

Sintaxis para múltiples tablas:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
SET col_name1=expr1 [, col_name2=expr2 ...]
[WHERE where_definition]
```

El comando UPDATE actualiza columnas en registros de tabla existentes con nuevos valores. La cláusula SET indica qué columna modificar y los valores que puede recibir. La cláusula WHERE , si se da, especifica qué registros deben actualizarse. De otro modo, se actualizan todos los registros. Si la cláusula ORDER BY se especifica, los registros se actualizan en el orden que se especifica. La cláusula LIMIT es el límite de registros a actualizar.

El comando UPDATE soporta los siguientes modificadores:

- Si usa la palabra clave LOW_PRIORITY , la ejecución de UPDATE se retrasa hasta que no haya otros clientes leyendo de la tabla.
- Si usa la palabra clave IGNORE, el comando de actualización no aborta incluso si ocurren errores durante la actualización. Los registros que presenten conflictos de clave duplicada no se actualizan. Los registros cuyas columnas se actualizan a valores que provocarían errores de conversión de datos se actualizan al valor válido más próximo.

Si accede a una columna de *tbl_name* en una expresión, UPDATE usa el valora ctual de la columna. Por ejemplo, el siguiente comando pone la columna age a uno más que su valor actual:

```
mysql> UPDATE persondata SET age=age+1;
```

Las asignaciones UPDATE se avalúna de izquierda a derecha. Por ejemplo, el siguiente comando dobla la columna age y luego la incrementa:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

Si pone en una columna el valor que tiene actualmente, MySQL se da cuenta y no la actualiza.

Si actualiza una columna declarada como NOT NULL con un valor NULL, la columna recibe el valor por defecto apropiado para el tipo de la columna y se incrementa el contador de advertencias. El valor por defecto es 0 para tipos numéricos, la cadena vacía ('') para tipos de cadena, y el valor “cero” para valores de fecha y hora.

UPDATE retorna el número de registros que se cambian. En MySQL 5.0, la función mysql_info() de la API de C retorna el número de registros coincidentes actualizados y el número de advertencias que ocurren durante el UPDATE.

Puede usar LIMIT *row_count* para restringir el alcance del UPDATE. Una cláusula LIMIT es una restricción de registros coincidentes. El comando para en cuanto encuentra *row_count* registos que satisfagan la cláusula WHERE , tanto si han sido cambiados como si no.

Si un comando UPDATE incluye una cláusula ORDER BY, los registros se actualizan en el orden especificado por la cláusula.

Puede realizar operaciones `UPDATE` que cubran varias tablas. La parte *`table_references`* lista las tablas involucradas en el join. Su sintaxis se describe ámpliamente en Sección 13.2.7.1, “Sintaxis de `JOIN`”. Aquí hay un ejemplo:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

Este ejemplo muestra un inner join usando el operador coma, pero los comandos `UPDATE` de múltiples tablas pueden usar cualquier tipo de join permitido en comandos `SELECT` tales como `LEFT JOIN`.

Nota: *No* puede usar `ORDER BY` o `LIMIT` con un `UPDATE` de múltiples tablas.

En MySQL 5.0, necesita el permiso `UPDATE` sólo para columnas referenciadas en un `UPDATE` de múltiples tablas que se actualizan realmente. Necesita sólo el permiso `SELECT` para algunas columnas que se leen pero no se modifican.

Si usa un comando `UPDATE` de múltiples tablas que involucren tablas `InnoDB` con restricciones de claves foráneas, el optimizador de MySQL puede procesar tablas en un orden distinto al de la relación padre/hijo. En este caso, el comando fall y hace un roll back. En su lugar, actualice una única tabla y confíen en las capacidades de `ON UPDATE` que proporciona `InnoDB` para que el resto de tablas se modifiquen acórdemente. Consulte Sección 15.6.4, “Restricciones (constraints) `FOREIGN KEY`”.

Actualmente, no puede actualizar una tabla y seleccionar de la misma en una subconsulta.

13.3. Sentencias útiles de MySQL

13.3.1. Sintaxis de `DESCRIBE` (Información acerca de las columnas)

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

`DESCRIBE` proporciona información acerca de columnas en una tabla. Es una abreviación de `SHOW COLUMNS FROM`. Desde MySQL 5.0.1, estos comandos también muestran información para vistas.

Consulte Sección 13.5.4.3, “Sintaxis de `SHOW COLUMNS`”.

`col_name` puede ser un nombre de columna, o una cadena con los caracteres de SQL '`%`' y '`_`' para obtener salida sólo para las columnas con nombres que coincidan con la cadena. No hay necesidad de delimitar la cadena con comillas a no ser que contenga espacios u otros caracteres especiales.

```
mysql> DESCRIBE city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  |      | PRI | NULL    | auto_increment |
| Name       | char(35) |      |     |         |                |
| Country    | char(3)  |      | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  |      |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

La columna `Null` indica si pueden almacenarse los valores `NULL` , mostrando `YES` cuando se permiten valores `NULL`.

La columna `Key` indica si el campo está indexado. Un valor de `PRI` indica que el campo es parte de una clave primaria de tabla. `UNI` indica que el campo es parte de un índice `UNIQUE`. El valor `MUL` indica que se permiten múltiples ocurrencias de un valor dado dentro del campo.

Un campo puede designarse como MUL incluso si se usa un índice UNIQUE si se permiten valores NULL , ya que múltiples registros en un índice UNIQUE pueden tener un valor NULL si la columna no se declara NOT NULL. Otra causa para MUL en un índice UNIQUE es cuando dos columnas de un índice UNIQUE compuesto; mientras la combinación de las columnas sea única, cada columna puede tener múltiples ocurrencias de un valor dado. Tenga en cuenta que en un índice compuesto sólo el campo de más a la izquierda del índice tiene una entrada en la columna Key .

La columna Default indica el valor por defecto asignado al campo.

La columna Extra contiene cualquier información adicional disponible acerca de un campo dado. En nuestro ejemplo la columna Extra indica que la columna Id se creó con la palabra clave AUTO_INCREMENT .

Si los tipos de columna son distintos a los esperados según el comando CREATE TABLE , tenga en cuenta que a veces MySQL cambia los tipos de columa. Consulte Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”.

El comando DESCRIBE se proporciona por compatibilidad con Oracle.

Los comandos SHOW CREATE TABLE y SHOW TABLE STATUS proporcionan información acerca de tablas . Consulte Sección 13.5.4, “Sintaxis de SHOW”.

13.3.2. Sintaxis de USE

```
USE db_name
```

El comando USE *db_name* le dice a MySQL que use la base de datos *db_name* como la base de datos por defecto para los comandos siguientes. Sigue siendo la base de datos por defecto hasta el final de la sesión o hasta que se realiza otro comando USE :

```
mysql> USE db1;
mysql> SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
mysql> USE db2;
mysql> SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

Hacer una base de datos particular la actual signfica que el comando USE no le imposibilita a acceder a tablas en otras bases de datos. El siguiente ejemplo accede a la tabla author desde la base de datos db1 y a la tabla editor desde la base de datos db2:

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
    ->        WHERE author.editor_id = db2.editor.editor_id;
```

El comando USE se proporciona por compatibilidad con Sybase.

13.4. Comandos transaccionales y de bloqueo de MySQL

13.4.1. Sintaxis de START TRANSACTION, COMMIT y ROLLBACK

Por defecto, MySQL se ejecuta con el modo autocommit activado. Esto significa que en cuanto ejecute un comando que actualice (modifique) una tabla, MySQL almacena la actualización en disco.

Si usa tablas transaccionales (como InnoDB o BDB), puede desactivar el modo autocommit con el siguiente comando:

```
SET AUTOCOMMIT=0;
```

Tras deshabilitar el modo autocommit poniendo la variable AUTOCOMMIT a cero, debe usar COMMIT para almacenar los cambios en disco o ROLLBACK si quiere ignorar los cambios hechos desde el comienzo de la transacción.

Si quiere deshabilitar el modo autocommit para una serie única de comandos, puede usar el comando START TRANSACTION:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

Con START TRANSACTION, autocommit permanece deshabilitado hasta el final de la transacción con COMMIT o ROLLBACK. El modo autocommit vuelve a su estado prévio.

BEGIN y BEGIN WORK se soportan como alias para START TRANSACTION para iniciar una transacción. START TRANSACTION es sintaxis SQL estándar y es la forma recomendada para iniciar una transacción ad-hoc . El comando BEGIN difiere del uso de la palabra clave BEGIN que comienza un comando compuesto BEGIN ... END. El último no comienza una transacción. Consulte Sección 19.2.7, “Sentencia compuesta BEGIN ... END”.

Puede comenzar una transacción así:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

La cláusula WITH CONSISTENT SNAPSHOT comienza una lectura consistente para motores de almacenamiento capaces de ello. Actualmente, esto se aplica sólo a InnoDB. El efecto es el mismo que realizar un START TRANSACTION seguido por un SELECT desde cualquier tabla InnoDB . Consulte Sección 15.10.4, “Lecturas consistentes que no bloquean”.

Comenzar una transacción provoca que se realice un UNLOCK TABLES implícito.

Tenga en cuenta que si no usa tablas transaccionales, cualquier cambio se almacena de golpe, a pesar del estado del modo autocommit .

Si realiza un comando ROLLBACK tras actualizar una tabla no transaccional dentro de una transacción, ocurre una advertencia ER_WARNING_NOT_COMPLETE_ROLLBACK. Los cambios en tablas transaccionales se deshacen, pero no los cambios en tablas no transaccionales.

Cada transacción se almacena en el log binario en un trozo, hasta COMMIT. Las transacciones que se deshacen no se loguean. (*Excepción*: Las modificaciones a tablas no transaccionales no pueden deshacerse. Si una transacción que se deshace incluye modificaciones a tablas no transaccionales, la transacción entera se loguea con un comando ROLLBACK al final para asegurar que las modificaciones a estas tablas se replican.) Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

Puede cambiar el nivel de aislamiento para transacciones con SET TRANSACTION ISOLATION LEVEL. Consulte Sección 13.4.6, “Sintaxis de SET TRANSACTION”.

Deshacer puede ser una operación lenta que puede ocurrir sin que el usuario lo haya pedido explícitamente (por ejemplo, cuando ocurre un error). Debido a ello, SHOW PROCESSLIST en MySQL 5.0 muestra Rolling back en la columna Statepara la conexión durante rollbacks implícitos y explícitos (comando SQL ROLLBACK).

13.4.2. Sentencias que no se pueden deshacer

Algunos comandos no pueden deshacerse. En general, esto incluye comandos del lenguaje de definición de datos (DDL), tales como los que crean y borran bases de datos, los que crean, borran o alteran tablas o rutinas almacenadas.

Debe designar que sus transacciones no incluyan tales comandos. Si realiza un comando pronto en una transacción que no puede deshacerse, y luego un comando posterior falla, el efecto global de la transacción no puede deshacerse mediante un comando `ROLLBACK` .

13.4.3. Sentencias que causan una ejecución (commit) implícita

Cada uno de los comandos siguientes (y cualquier sinónimo de los mismos) terminan una transacción implícitamente, como si hubiera realizado un `COMMIT` antes de ejecutar el comando:

`ALTER TABLE`	`BEGIN`	`CREATE INDEX`
`CREATE TABLE`		`CREATE DATABASE`
`DROP DATABASE`	`DROP INDEX`	`DROP TABLE`
`LOAD MASTER DATA`	`LOCK TABLES`	`RENAME TABLE`
`SET AUTOCOMMIT=1`	`START TRANSACTION`	`TRUNCATE TABLE`

`UNLOCK TABLES` también realiza un commit de una transacción si hay cualquier tabla bloqueada.

Las transacciones no pueden anidarse. Esto es una consecuencia del `COMMIT` implícito realizado por cualquier transacción actual cuando realiza un comando `START TRANSACTION` o uno de sus sinónimos.

13.4.4. Sintaxis de `SAVEPOINT` y `ROLLBACK TO SAVEPOINT`

```
SAVEPOINT identifier
ROLLBACK TO SAVEPOINT identifier
```

En MySQL 5.0, `InnoDB` soporta los comandos SQL `SAVEPOINT` y `ROLLBACK TO SAVEPOINT`.

El comando `SAVEPOINT` crea un punto dentro de una transacción con un nombre `identifier`. Si la transacción actual tiene un punto con el mismo nombre, el antiguo se borra y se crea el nuevo.

El comando `ROLLBACK TO SAVEPOINT` deshace una transacción hasta el punto nombrado. Las modificaciones que la transacción actual hace al registro tras el punto se deshacen en el rollback, pero `InnoDB` *no* libera los bloqueos de registro que se almacenaron en memoria tras el punto . (Tenga en cuenta que para un nuevo registro insertado, la información de bloqueo se realiza a partir del ID de transacción almacenado en el registro; el bloqueo no se almacena separadamente en memoria. En este caso, el bloqueo de registro se libera al deshacerse todo.) Los puntos creados tras el punto nombrado se borran.

Si un comando retorna el siguiente error, significa que no existe ningún punto con el nombre especificado:

```
ERROR 1181: Got error 153 during ROLLBACK
```

Todos los puntos de la transacción actual se borran si ejecuta un `COMMIT`, o un `ROLLBACK` que no nombre ningún punto.

13.4.5. Sintaxis de `LOCK TABLES` y `UNLOCK TABLES`

```
LOCK TABLES
    tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
    [, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
UNLOCK TABLES
```

LOCK TABLES bloquea tablas para el flujo actual. Si alguna de las tablas la bloquea otro flujo, bloquea hasta que pueden adquirirse todos los bloqueos. UNLOCK TABLES libera cualquier bloqueo realizado por el flujo actual. Todas las tablas bloqueadas por el flujo actual se liberan implícitamente cuando el flujo reliza otro LOCK TABLES, o cuando la conexión con el servidor se cierra.

Un bloqueo de tabla protege sólo contra lecturas inapropiadas o escrituras de otros clientes. El cliente que tenga el bloqueo, incluso un bloqueo de lectura, puede realizar operaciones a nivel de tabla tales como DROP TABLE.

Tenga en cuenta lo siguiente a pesar del uso de LOCK TABLES con tablas transaccionales:

- LOCK TABLES no es una operación transaccional y hace un commit implícito de cualquier transacción activa antes de tratar de bloquear las tablas. También, comenzar una transacción (por ejemplo, con START TRANSACTION) realiza un UNLOCK TABLES implícito. (Consulte Sección 13.4.3, “Sentencias que causan una ejecución (commit) implícita”.)

- La forma correcta de usar LOCK TABLES con tablas transaccionales,como InnoDB, es poner AUTOCOMMIT = 0 y no llamar a UNLOCK TABLES hasta que hace un commit de la transacción explícitamente. Cuando llama a LOCK TABLES, InnoDB internamente realiza su propio bloqueo de tabla, y MySQL realiza su propio bloqueo de tabla. InnoDB libera su bloqueo de tabla en el siguiente commit, pero para que MySQL libere su bloqueo de tabla, debe llamar a UNLOCK TABLES. No debe tener AUTOCOMMIT = 1, porque entonces InnoDB libera su bloqueo de tabla inmediatamente tras la llamada de LOCK TABLES, y los deadlocks pueden ocurrir fácilmente. (Tenga en cuenta que en MySQL 5.0, no adquirimos el bloqueo de tabla InnoDB en absoluto si AUTOCOMMIT=1, para ayudar a aplicaciones antiguas a envitar deadlocks.)

- ROLLBACK no libera bloqueos de tablas no transaccionales de MySQL.

Para usar LOCK TABLES en MySQL 5.0, debe tener el permiso LOCK TABLES y el permiso SELECT para las tablas involucradas.

La razón principal para usar LOCK TABLES es para emular transacciones o para obtener más velocidad al actualizar tablas. Esto se explica con más detalle posteriormente.

Si un flujo obtiene un bloqueo READ en una tabla, ese flujo (y todos los otros) sólo pueden leer de la tabla. Si un flujo obtiene un bloqueo WRITE en una tabla, sólo el flujo con el bloqueo puede escribir a la tabla. El resto de flujos se bloquean hasta que se libera el bloqueo.

La diferencia entre READ LOCAL y READ es que READ LOCAL permite comandos INSERT no conflictivos (inserciones concurrentes) se ejecuten mientras se mantiene el bloqueo. Sin embargo, esto no puede usarse si va a manipular los ficheros de base de datos fuera de MySQL mientras mantiene el bloqueo. Para tablas InnoDB , READ LOCAL esencialmente no hace nada: No bloquea la tabla. Para tablas InnoDB , el uso de READ LOCAL está obsoleto ya que una SELECT consistente hace lo mismo, y no se necesitan bloqueos.

Cuando usa LOCK TABLES, debe bloquear todas las tablas que va a usar en sus consultas. Mientras los bloqueos obtenidos con un comando LOCK TABLES están en efecto, no puede acceder a ninguna tabla que no estuviera bloqueada por el comando. Además, no puede usar una tabla bloqueada varias veces en una consulta --- use alias para ello. Tenga en cuenta que en este caso, debe tener un bloqueo separado para cada alias.

```
mysql> LOCK TABLE t WRITE, t AS t1 WRITE;
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;
```

Si sus consultas se refieren a una tabla que use un alias, debe bloquear la tabla que usa el mismo alias. No funciona bloquear la tabla sin especificar el alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Si bloquea una tabla usando un alias, debe referirse a ella en sus consultas usando este 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` bloquea normalmente teniendo una prioridad superior que `READ` al bloquear para asegurar que las actualizaciones se procesan en cuanto se puede. Esto significa que si un flujo obtiene un bloqueo `READ` y luego otro flujo pide un bloqueo `WRITE` , las peticiones de bloqueo `READ` posteriores esperan hasta que el flujo `WRITE` quita el bloqueo. Puede usar bloqueos `LOW_PRIORITY WRITE` para permitir a otros flujos que obtengan bloqueos `READ` mientras el flujo está en espera para el bloqueo `WRITE`. Debe usar bloqueos `LOW_PRIORITY WRITE` sólo si está seguro que habrá un momento sin flujos con bloqueos `READ` .

`LOCK TABLES` funciona como sigue:

1. Ordena todas las tablas a ser bloqueadas en un orden definido internamente. Desde el punto de vista del usuario, este orden es indefinido.
2. Si una tabla se bloquea con bloqueo de lectura y escritura, pone el bloqueo de escritura antes del de lectura.
3. Bloquea una tabla a la vez hasta que la sesión obtiene todos los bloqueos.

Esta política asegura un bloqueo de tablas libre de deadlocks. Sin embargo hay otros puntos que debe tener en cuenta respecto a esta política:

Si está usando un bloqueo `LOW_PRIORITY WRITE` para una tabla, sólo significa que MySQL espera para este bloqueo hasta que no haya flujos que quieren un bloqueo `READ` . Cuando el flujo ha obtenido el bloqueo `WRITE` y está esperando para obtener un bloqueo para la siguiente tabla en la lista, todos los otros flujos esperan hasta que el bloqueo `WRITE` se libera. Si esto es un problema con su aplicación, debe considerar convertir algunas de sus tablas a transaccionales.

Puede usar `KILL` para terminar un flujo que está esperando para un bloqueo de tabla. Consulte Sección 13.5.5.3, “Sintaxis de KILL”.

Tenga en cuenta que *no* debe bloquear ninguna tabla que esté usando con `INSERT DELAYED` ya que en tal caso el `INSERT` lo realiza un flujo separado.

Normalmente, no tiene que bloquear tablas, ya que todos los comandos `UPDATE` son atómicos, ningún otro flujo puede interferir con ningún otro que está ejecutando comandos SQL . Hay algunos casos en que no debe bloquear tablas de ningún modo:

- Si va a ejecutar varias operaciones en un conjunto de tablas `MyISAM` , es mucho más rápido bloquear las tablas que va a usar. Bloquear tablas `MyISAM` acelera la inserción, las actualizaciones, y los

borrados. Por contra, ningún flujo puede actualizar una tabla con un bloqueo READ (incluyendo el que tiene el bloqueo) y ningún flujo puede acceder a una tabla con un bloqueo WRITE distinto al que tiene el bloqueo.

La razón que algunas operaciones MyISAM sean más rápidas bajo LOCK TABLES es que MySQL no vuelca la caché de claves para la tabla bloqueada hasta que se llama a UNLOCK TABLES. Normalmente, la caché de claves se vuelca tras cada comando SQL.

- Si usa un motor de almacenamiento en MySQL que no soporta transacciones, debe usar LOCK TABLES si quiere asegurarse que ningún otro flujo se ejecute entre un SELECT y un UPDATE. El ejemplo mostrado necesita LOCK TABLES para ejecutarse sin problemas:

```
mysql> LOCK TABLES trans READ, customer WRITE;
mysql> SELECT SUM(value) FROM trans WHERE customer_id=some_id;
mysql> UPDATE customer
    ->     SET total_value=sum_from_previous_statement
    ->     WHERE customer_id=some_id;
mysql> UNLOCK TABLES;
```

Sin LOCK TABLES, es posible que otro flujo pueda insertar un nuevo registro en la tabla trans entre la ejecución del comando SELECT y UPDATE.

Puede evitar usar LOCK TABLES en varios casos usando actualizaciones relativas (UPDATE customer SET *value=value+new_value*) o la función LAST_INSERT_ID() , Consulte Sección 1.7.5.3, “Transacciones y operaciones atómicas”.

Puede evitar bloquear tablas en algunos casos usando las funciones de bloqueo de nivel de usuario GET_LOCK() y RELEASE_LOCK(). Estos bloqueos se guardan en una tabla hash en el servidor e implementa pthread_mutex_lock() y pthread_mutex_unlock() para alta velocidad. Consulte Sección 12.9.4, “Funciones varias”.

Consulte Sección 7.3.1, “Métodos de bloqueo”, para más información acerca de la política de bloqueo.

Puede bloquear todas las tablas en todas las bases de datos con bloqueos de lectura con el comando FLUSH TABLES WITH READ LOCK . Consulte Sección 13.5.5.2, “Sintaxis de FLUSH”. Esta es una forma muy conveniente para obtener copias de seguridad si tiene un sistema de ficheros como Veritas que puede obtener el estado en un punto temporal.

Nota: Si usa ALTER TABLE en una tabla bloqueada, puede desbloquearse. Consulte Sección A.7.1, “Problemas con ALTER TABLE”.

13.4.6. Sintaxis de SET TRANSACTION

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

Este comando prepara el nivel de aislamiento de transacción para la siguiente transacción, globalmente, o para la sesión actual.

El comportamiento por defecto de SET TRANSACTION es poner el nivel de aislammiento para la siguiente transacción (que no ha empezado todavía). Si usa lka palabra clave GLOBAL el comando pone el nivel de aislamiento de transacción por defecto globalmente para todas las transacciones creadas desde ese momento. Las conexiones existentes no se ven afectadas. Necesita el permiso SUPER para hacerlo. Usar la palabra clave SESSION determina el nivel de transacción para todas las transacciones futuras realizadas en la conexión actual.

Para descripciones del nivel de aislamiento de cada transacción `InnoDB`, consulte Sección 15.10.3, “`InnoDB` y `TRANSACTION ISOLATION LEVEL`”. `InnoDB` soporta cada uno de estos niveles en MySQL 5.0. El nivel por defecto es `REPEATABLE READ`.

Puede inicializar el nivel de aislamiento global por defecto para `mysqld` con la opción `--transaction-isolation` . Consulte Sección 5.3.1, “Opciones del comando `mysqld`”.

13.5. Sentencias de administración de base de datos

13.5.1. Sentencias para la gestión de cuentas

13.5.1.1. Sintaxis de `CREATE USER`

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

El comando `CREATE USER` se añadió en MySQL 5.0.2. Este comando crea nuevas cuentas MySQL. Para usarlas, debe tener el permiso global `CREATE USER` o el permiso `INSERT` para la base de datos `mysql` . Para cada cuenta, `CREATE USER` crea un nuevo registro en la tabla `mysql.user` que no tiene permisos. Un error ocurre si la cuenta ya existe.

La cuenta puede tener una contraseña con la cláusula opcional `IDENTIFIED BY`. El valor *`user`* y la contraseña se dan del mismo modo que para el comando `GRANT`. En particular, para especificar la contraseña en texto plano, omita la palabra clave `PASSWORD`. Para especificar la contraseña como el valor hasheado retornado por la función `PASSWORD()` , incluya la palabra clave `PASSWORD`. Consulte Sección 13.5.1.3, “Sintaxis de `GRANT` y `REVOKE`”.

13.5.1.2. Sintaxis de `DROP USER`

```
DROP USER user [, user] ...
```

El comando `DROP USER` borra una o más cuentas MySQL . Para usarlo, debe tener el permiso global `CREATE USER` o el permiso `DELETE` para la base de datos `mysql` . Cada cuenta se nombra usando el mismo formato que para `GRANT` o `REVOKE`; por ejemplo, `'jeffrey'@'localhost'`. Las partes de usuario y equipo del nombre de cuenta se corresponden a las columnas `User` y `Host` del registro de la tabla `user` para la cuenta.

`DROP USER` como está en MySQL 5.0.0 borra sólo cuentas que no tienen permisos. En MySQL 5.0.2, se modificó para eliminar permisos de cuenta también. Esto significa que el procedimiento para borrar una cuenta depende en su versión de MySQL.

Desde MySQL 5.0.2, puede borrar una cuenta y sus permisos como sigue:

```
DROP USER user;
```

El comando borra registros de permisos para la cuenta de todas las tablas de permisos.

En MySQL 5.0.0 y 5.0.1, `DROP USER` borra sólo cuentas MySQL que no tienen permisos. En estas versiones MySQL sólo sirve para borrar cada registro de cuenta de la tabla `user` . Para borrar una cuenta MySQL completamente (incluyendo todos sus permisos), debe usar el siguiente procedimiento, realizando estos pasos en el orden mostrado:

1. Use `SHOW GRANTS` para determinar los permisos que tiene la cuenta. Consulte Sección 13.5.4.10, “Sintaxis de `SHOW GRANTS`”.

2. Use `REVOKE` para revocar los permisos mostrados por `SHOW GRANTS`. Esto borra registros para la cuenta de todas las tablas de permisos excepto la tabla `user` , y revoca cualquier permiso global listado en la tabla `user` . Consulte Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”.

3. Borre la cuenta usando `DROP USER` para borrar el registro de la tabla `user` .

`DROP USER` no cierra automáticamente ninguna sesión de usuario. En lugar de ello, en el evento que un usuario con una sesión abierta se elimina, el comando no tiene efecto hasta que se cierra la sesión de usuario. Una vez se ha cerrado, el usuario se borra, y el próximo usuario de logueo del usuario fallará. Esto es por diseño.

13.5.1.3. Sintaxis de `GRANT` y `REVOKE`

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON [object_type] {tbl_name | * | *.* | db_name.*}
    TO user [IDENTIFIED BY [PASSWORD] 'password']
        [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
    [REQUIRE
        NONE |
        [{SSL| X509}]
        [CIPHER 'cipher' [AND]]
        [ISSUER 'issuer' [AND]]
        [SUBJECT 'subject']]
    [WITH with_option [with_option] ...]

object_type =
    TABLE
  | FUNCTION
  | PROCEDURE

with_option =
    GRANT OPTION
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
```

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON [object_type] {tbl_name | * | *.* | db_name.*}
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Los comandos `GRANT` y `REVOKE` permiten a los adminitradores de sistemas crear cuentas de usuario MySQL y darles permisos y quitarlos de las cuentas.

La información de cuenta de MySQL se almacena en las tablas de la base de datos `mysql` . Esta base de datos y el sistema de control de acceso se discuten extensivamente en Capítulo 5, *Administración de bases de datos*, que puede consultar para más detalles.

Si las tablas de permisos tienen registros de permisos que contienen nombres de tablas o bases de datos con mayúsculas y minúsculas y la variable de sistema `lower_case_table_names` está activa, `REVOKE` no puede usarse para quitar los permisos. Es necesario manipular las tablas de permisos directamente. (`GRANT` no creará estos registros cuando está activo `lower_case_table_names` , pero tales registros pueden haberse creado préviamente a activar la variable.)

Los permisos pueden darse en varios niveles:

- **Nivel global**

Los permisos globales se aplican a todas las bases de datos de un servidor dado. Estos permisos se almacenan en la tabla `mysql.user`. `GRANT ALL ON *.*` y `REVOKE ALL ON *.*` otorgan y quitan sólo permisos globales.

- **Nivel de base de datos**

  Los permisos de base de datos se aplican a todos los objetos en una base de datos dada. Estos permisos se almacenan en las tablas `mysql.db` y `mysql.host` . `GRANT ALL ON db_name.*` y `REVOKE ALL ON db_name.*` otorgan y quitan sólo permisos de bases de datos.

- **Nivel de tabla**

  Los permisos de tabla se aplican a todas las columnas en una tabla dada. Estos permisos se almacenan en la tabla `mysql.tables_priv` . `GRANT ALL ON db_name.tbl_name` y `REVOKE ALL ON db_name.tbl_name` otorgan y quian permisos sólo de tabla.

- **Nivel de columna**

  Los permisos de columna se aplican a columnas en una tabla dada. Estos permisos se almacenan en la tabla `mysql.columns_priv` . Usando `REVOKE`, debe especificar las mismas columnas que se otorgaron los permisos.

- **Nivel de rutina**

  Los permisos `CREATE ROUTINE`, `ALTER ROUTINE`, `EXECUTE`, y `GRANT` se aplican a rutinas almacenadas. Pueden darse a niveles global y de base de datos. Además, excepto para `CREATE ROUTINE`, estos permisos pueden darse en nivel de rutinas para rutinas individuales y se almacenan en la tabla `mysql.procs_priv` .

La cláusula *`object_type`* se añadió en MySQL 5.0.6. Debe especificarse como `TABLE`, `FUNCTION`, o `PROCEDURE` cuando el siguiente objeto es una tabla, una función almacenada, o un procedimiento almacenado. Para usar esta cláusula cuando actualice de una versión anterior de MySQL a la 5.0.6, debe actualizar las tablas de permisos. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Para usar `GRANT` o `REVOKE`, debe tener el permiso `GRANT OPTION` , y debe tener los permisos que está dando o quitando.

Para hacer fácil de quitar todos los permisos, MySQL 5.0 tiene la siguiente sintaxis, que borra todos los permisos globales, de nivel de base de datos y de nivel de tabla para los usuarios nombrados:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Para usar esta sintaxis `REVOKE` , debe tener el permiso `CREATE USER` global o el permiso `UPDATE` para la base de datos `mysql` .

Para los comandos `GRANT` y `REVOKE` , `priv_type` pueden especificarse como cualquiera de los siguientes:

Permiso	Significado
`ALL [PRIVILEGES]`	Da todos los permisos simples excepto `GRANT OPTION`
`ALTER`	Permite el uso de `ALTER TABLE`
`ALTER ROUTINE`	Modifica o borra rutinas almacenadas

`CREATE`	Permite el uso de `CREATE TABLE`
`CREATE ROUTINE`	Crea rutinas almacenadas
`CREATE TEMPORARY TABLES`	Permite el uso de `CREATE TEMPORARY TABLE`
`CREATE USER`	Permite el uso de `CREATE USER`, `DROP USER`, `RENAME USER`, y `REVOKE ALL PRIVILEGES`.
`CREATE VIEW`	Permite el uso de `CREATE VIEW`
`DELETE`	Permite el uso de `DELETE`
`DROP`	Permite el uso de `DROP TABLE`
`EXECUTE`	Permite al usuario ejecutar rutinas almacenadas
`FILE`	Permite el uso de `SELECT ... INTO OUTFILE` y `LOAD DATA INFILE`
`INDEX`	Permite el uso de `CREATE INDEX` y `DROP INDEX`
`INSERT`	Permite el uso de `INSERT`
`LOCK TABLES`	Permite el uso de `LOCK TABLES` en tablas para las que tenga el permiso `SELECT`
`PROCESS`	Permite el uso de `SHOW FULL PROCESSLIST`
`REFERENCES`	No implementado
`RELOAD`	Permite el uso de `FLUSH`
`REPLICATION CLIENT`	Permite al usuario preguntar dónde están los servidores maestro o esclavo
`REPLICATION SLAVE`	Necesario para los esclavos de replicación (para leer eventos del log binario desde el maestro)
`SELECT`	Permite el uso de `SELECT`
`SHOW DATABASES`	`SHOW DATABASES` muestra todas las bases de datos
`SHOW VIEW`	Permite el uso de `SHOW CREATE VIEW`
`SHUTDOWN`	Permite el uso de `mysqladmin shutdown`
`SUPER`	Permite el uso de comandos `CHANGE MASTER`, `KILL`, `PURGE MASTER LOGS`, and `SET GLOBAL` , el comando `mysqladmin debug` le permite conectar (una vez) incluso si se llega a `max_connections`
`UPDATE`	Permite el uso de `UPDATE`
`USAGE`	Sinónimo de “no privileges”
`GRANT OPTION`	Permite dar permisos

El permiso `EXECUTE` no es operacional hasta MySQL 5.0.3. `CREATE VIEW` y `SHOW VIEW` se añadieron en MySQL 5.0.1. `CREATE USER`, `CREATE ROUTINE`, y `ALTER ROUTINE` se añadieron en MySQL 5.0.3. Para usar estos permisos al actualizar desde una versión anterior de MySQL que no los tenga, debe actualizar primero las tablas de permisos, como se describe en Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

El permiso `REFERENCES` actualmente no se usa.

`USAGE` puede especificarse cuando quiere crear un usuario sin permisos.

Use `SHOW GRANTS` para determinar qué permisos tiene la cuenta. Consulte Sección 13.5.4.10, “Sintaxis de `SHOW GRANTS`”.

Puede asignar permisos globales usando sintaxis `ON *.*` o permisos a nivel de base de datos usando la sintaxis `ON db_name.*`. Si especifica `ON *` y tiene seleccionada una base de datos por defecto, los permisos se dan en esa base de datos. (**Atención:** Si especifica `ON *` y *no* ha seleccionado una base de datos por defecto, los permisos dados son globales.)

Los permisos `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN`, y `SUPER` son permisos administrativos que sólo pueden darse globalmente (usando sintaxis `ON *.*`).

Otros permisos pueden darse globalmente o a niveles más específicos.

Los únicos valores `priv_type` que puede especificar para una tabla son `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT OPTION`, `INDEX`, y `ALTER`.

Los únicos valores `priv_type` que puede especificar para una columna (cuando usa la cláusula `column_list`) son `SELECT`, `INSERT`, y `UPDATE`.

Los únicos valores `priv_type` que puede especificar a nivel de rutina son `ALTER ROUTINE`, `EXECUTE`, y `GRANT OPTION`. `CREATE ROUTINE` no es un permiso de nivel de rutina porque debe tener este permiso para ser capaz de crear una rutina en primer lugar.

Para los niveles global, base de datos, tabla y rutina, `GRANT ALL` asigna sólo los permisos que existen en el nivel que está otorgándolos. Por ejemplo, si usa `GRANT ALL ON db_name.*`, este es un comando de nivel de base de datos, así que ninguno de los permisos únicamente globales tales como `FILE` se otorgan.

MySQL le permite dar permisos incluso en objetos de bases de datos que no existen. En tales casos, los permisos a dar deben incluir el permiso `CREATE` . *Este es el comportamiento diseñado*, y se pretende permitir al administrador de la base de datos perparar cuentas de usuario y permisos para objetos de base de datos que se crearán posteriormente.

MySQL no elimina automáticamente nigún permiso si borra una tabla o base de datos . Si borra un rutina, se quita cualquier permiso dado a nivel de rutina para la misma.

Nota: los carácters comodín '_' y '%' se permiten al especificar nombres de base de datos en comandos `GRANT` que otorgan permisos a nivel global o de base de datos. Esto significa, por ejemplo, que si quiere usar un carácter '_' como parte de un nombre de base de datos, debe especificarlo como '_' en el comando `GRANT` , para evitar que el usuario sea capaz de acceder a bases de datos adicionales que coincidan con el patrón de los comodines, por ejemplo `GRANT ... ON `foo_bar`.* TO ...`.

Para acomodar los permisos a los usuarios de equipos arbitrários, MySQL soporta especificar el valor *`user`* con la forma *`user_name@host_name`*. Si un valor *`user_name`* o *`host_name`* es legal como identificador sin poner entre comillas, no necesita hacerlo. Sin embargo, las comillas son necesarias para especificar una cadena *`user_name`* conteniendo caracteres especiales (tales como '-'), o una cadena *`host_name`* conteniendo caracteres especiales o comodín (tales como '%'); por ejemplo, `'test-user'@'test-hostname'`. Entrecomille el nombre de usuario y de equipo separadamente.

Puede especificar caracteres comodín en el nombre de equipo. Por ejemplo, *`user_name`*`@'%.loc.gov'` se aplica a *`user_name`* para cualquier equipo en el dominio `loc.gov` , y *`user_name`*`@'144.155.166.%'` se aplica a *`user_name`* para cualquier equipo en la clase subred clase C `144.155.166` .

La forma simple *`user_name`* es sinónimo de *`user_name`*`@'%'`.

MySQL no soporta comodines en el nombre de usuario. Los usuarios anónimos se definien insertando entradas con `User=''` en la tabla `mysql.user` o creando un usuario con un nombre vacío con el comando `GRANT` :

```
mysql> GRANT ALL ON test.* TO ''@'localhost' ...
```

Al especificar valores delimitados, use comillas simples para delimitar los nombres de bases de datos, tabla, columna y de rutina (`` ` ``). Para los nombres de equipo, nombres de usuario, y contraseñas como cadenas, use apóstrofes (`''`).

Advertencia: Si permite conectar con el servidor a usuarios anónimos, debe dar permisos a todos los usuarios locales como *`user_name`*`@localhost`. De otro modo, la cuenta de usuario anónimo para `localhost` en la tabla `mysql.user` (creada durante la instalación de MySQL) se usa cuando los usuarioa con nombre intentan loguear con el servidor MySQL desde la máquina local.

Puede determinar si esto se aplica a su sistema ejecutando la siguiente consulta, que lista cualquier usuario anónimo:

```
mysql> SELECT Host, User FROM mysql.user WHERE User='';
```

Si quiere borrar la cuenta anónima local para evitar el problema descrito, use estos comandos:

```
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

`GRANT` soporta nombres de equipo de hasta 60 caracteres. Los nombres de bases de datos, tablas, columnas y rutinas pueden tener hasta 64 caracteres. Los nombres de usuario pueden tener hasta 16 caracteres. Los nombres de usuario pueden tener hasta 16 caracteres. *Estos límites están harcodeados en el software MySQL y no pueden cambiarse alterando las tablas de permisos* .

Los permisos para una tabla o columna se forman de forma aditiva como una `OR` lógica de los permisos en cada uno de los cuatro niveles de permisos. Por ejemplo, si la tabla `mysql.user` especifica que un usuario tiene un permiso `SELECT` global, el permiso no puede denegarse mediante una entrada en el nivel de base de datos, tabla o columna.

Los permisos de una columna pueden calcularse como sigue:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

En la mayoría de casos, puede dar derechoa a un usuario en sólo uno de los niveles de permisos, así que la vida normalmente no es tan complicada. Los detalles de este procedimiento de chequeo de permisos se presentan en Sección 5.6, “El sistema de privilegios de acceso de MySQL”.

Si otorga permisos para una combinación usuario/equipo que no existe en la tabla `mysql.user` se añade una entrada que permite allí hasta que se borra con un comando `DELETE`. En otras palabras, `GRANT` puede crear entradas `user` pero `REVOKE` no los borra; debe hacerlo explícitamente usando `DROP USER` o `DELETE`.

Si se crea un nuevo usuario o si tiene permisos globales para otorgar permisos, la contraseña de usuario se cambia con la contraseña especificada por la cláusula `IDENTIFIED BY` , si se da una. Si el usuario ya tiene una contraseña, esta se reemplaza por la nueva.

Atención: Si crea un nuevo usuario pero no especifica una cláusula `IDENTIFIED BY`, el usuario no tiene contraseña. Esto es muy poco seguro. Desde MySQL 5.0.2, puede activar el modo SQL `NO_AUTO_CREATE_USER` para evitar que `GRANT` cree un nuevo usuario si lo hiciese de otro modo, a no ser que `IDENTIFIED BY` se de para proporcionar la nueva contraseña de usuario.

Las contraseñas pueden ponerse con el comando `SET PASSWORD` . Consulte Sección 13.5.1.5, “Sintaxis de `SET PASSWORD`”.

En la cláusula `IDENTIFIED BY` , la contraseña debe darse como el valor de contraseña literal. No es necesario usar la función `PASSWORD()` como lo es para el comando `SET PASSWORD` . Por ejemplo:

```
GRANT ... IDENTIFIED BY 'mypass';
```

Si no quiere enviar la contraseña en texto plano y conoce el valor haseado que `PASSWORD()` retornaría para la contraseña, puede especificar el valor hasheado precedido por la palabra clave `PASSWORD`:

```
GRANT ...
IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

En un programa C , puede obtener el valor haseado usando la función `make_scrambled_password()` de la API de C.

Si da permisos para una base de datos, se crea una entrada en la tabla `mysql.db` si es necesario. Si todos los permisos para la base de datos se eliminan con `REVOKE`, esta entrada se borra.

Si un usuario no tiene permisos para una tabla, el nombre de tabla no se muestra cuando el usuario pide una lista de talbas (por ejemplo, con el comando `SHOW TABLES`).

El permiso `SHOW DATABASES` le permite a la cuenta ver nombres de bases de datos realizando el comando `SHOW DATABASE` . Las cuentas que no tienen este permiso sólo ven las bases de datos para las que tienen algún permiso, y no pueden usar el comando para nada si el servidor se arranca con la opción `--skip-show-database`.

La cláusula `WITH GRANT OPTION` le da al usuario la habilidad para dar a otros usuarios cualquier permiso que tenga el usuario en el nivel de permiso especificado. Debe tener cuidado de a quién da el permiso `GRANT OPTION`, ya que dos usuarios con permisos distintos pueden ser capaces de juntar permisos!

No puede dar a otro usuario un permiso que no tenga usted mismo; el permiso `GRANT OPTION` le permite asignar sólo los permisos que tenga usted.

Tenga en cuenta que cuando le da a un usuario el permiso `GRANT OPTION` a un nivel de permisos particular, cualquier permiso que tenga el usuario (o que se de en el futuro!) a este nivel también son otorgables por este usuario. Suponga que le da a un usuario el permisos `INSERT` en una base de datos. Si otorga el permiso `SELECT` en la base de datos y especifica `WITH GRANT OPTION`, el usuario puede quitar no sólo el permiso `SELECT` sino también `INSERT`. Si luego otorga el permiso `UPDATE` al usuario en la base de datos, el usuario puede quitar `INSERT`, `SELECT`, y `UPDATE`.

No debe otorgar permisos `ALTER` a un usuario normal. Si lo hace, el usuario puede intentar engañar al sistema de permisos renombrando tablas!

Las opciones `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, y `MAX_CONNECTIONS_PER_HOUR` *`count`* limitan el número de consultas, actualizaciones, y logueos que puede realizar un usuario durante cualquier perído de una hora. Si *`count`* es 0 (por defecto), esto significa que no hay limitación para ese usuario.

La `MAX_USER_CONNECTIONS` *`count`* opción, implementada en MySQL 5.0.3, limita el máximo número de conexiones simultáneas que la cuenta puede hacer. Si *`count`* es 0 (por defecto), la `max_user_connections` variable de sistema determina el número de conexiones simultáneas para la cuenta.

Nota: para especificar cualquiera de estas opciones de limitación de recursos para un usuario existente sin afectar a los permisos existentes, use `GRANT USAGE ON *.* ... WITH MAX_...`.

Consulte Sección 5.7.4, “Limitar recursos de cuentas”.

MySQL puede chequear atributos certificados X509 además que la autenticación usual que se basa en el nombre de usuario y contraseña. Para especificar opciones relacionadas con SSL para la cuenta MySQL, use la cláusula `REQUIRE` del comando `GRANT` . (Para información de transfondo sobre el uso de SSL con MySQL, consulte Sección 5.7.7, “Usar conexiones seguras”.)

Hay distintas posibilidades para limitar tipos de conexión para una cuenta:

- Si una cuenta no tiene requerimientos de SSL o X509, se permiten conexiones sin cifrar si la contraseña y nombre de usuario son válidos. Sin embargo, las conexiones no cifradas pueden usarse en las opciones de cliente, si el cliente tiene los ficheros clave y de certificado apropiados.

- La opción `REQUIRE SSL` le dice al servidor que permita sólo conexiones SSL cifradas para la cuenta. Tenga en cuenta que esta opción puede omitirse si hay algunos registros de control de acceso que permitan conexiones no SSL.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

- `REQUIRE X509` significa que el cliente debe tener un certificado válido pero que el certificador exacto y el asunto no importan. El único requerimiento que debe ser posible de verificar es la firma con uno de las AC certificadas.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

- `REQUIRE ISSUER 'issuer'` crea una restricción de intentos de conexión en que el cliente debe presentar un certificado X509 válido presentado por la AC *`issuer`*. Si el cliente presenta un certificado válido pero de otra AC, el servidor rehúsa la conexión. El uso de certificados X509 siempre implica encripción, por lo que la opción `SSL` no es necesaria.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
```

  Tenga en cuenta que el valor `ISSUER` debe entrarse como una cadena única.

- `REQUIRE SUBJECT 'subject'` crea la restricción en los intentos de conexión de que el cliente debe presentar un certificado X509 válido con el asunto *`subject`*. Si el cliente presenta un certificado válido pero con un asunto distinto, el servidor rehúsa la conexión.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com';
```

  Tenga en cuenta que el valor `SUBJECT` debe entrarse como una única cadena.

- `REQUIRE CIPHER 'cipher'` se necesita para asegurar que se usan cifradores suficientemente fuertes y longitudes de claves acordes. SSL por sí mismo puede ser débil si se usan algoritmos antiguos

con claves de cifrado cortas. Con esta opción, puede especificar el método de cifrado exacto para permitir una conexión.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Las opciones `SUBJECT`, `ISSUER`, y `CIPHER` pueden combinarse en la cláusula `REQUIRE` así:

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com'
    -> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Tenga en cuenta que los valores `SUBJECT` y `ISSUER` deben entrarse como una única cadena.

En MySQL 5.0, la palabra clave `AND` es opcional entre las opciones `REQUIRE` .

El orden de las opciones no importa, pero no puede especificarse ninguna opción dos veces.

Cuando `mysqld` arranca, todos los permisos se leen en memoria. Para más detalles, consulte Sección 5.6.7, “Cuándo tienen efecto los camios de privilegios”.

Tenga en cuenta que si usa permisos de tablas o de columnas para un usuario, el servidor examina los permisos de tablas y usuarios para todos los usuarios y esto ralentiza MySQL ligeramente. De forma similar, si limita el número de consultas, actualizaciones o conexiones para cualquier usuario, el servidor debe monitorizar estos valores.

Las mayores diferencias entre las versiones de `GRANT` de MySQL y SQL estándar son:

- En MySQL, los permisos se asocian con una combinación de nombre de usuario/equipo y no sólo con el usuario.
- SQL estándar no tienen permisos globales o a nivel de base de datos, ni soporta todos los tipos de permisos que soporta MySQL .
- MySQL no soporta los permisos de SQL estándar `TRIGGER` o `UNDER`.
- Los permisos de SQL estándar se estructuran de forma jerárquica. Si borra un usuario, todos los permisos que tuviera el usuario se eliminan. Esto es cierto a partir de MySQL 5.0.2 y si usa `DROP USER`. Antes de 5.0.2, los permisos otorgados no se eliminaban automáticamente; debía hacerlo a mano. Consulte Sección 13.5.1.2, “Sintaxis de `DROP USER`”.
- En SQL estándar, cuando borra una tabla, todos los permisos para la tabla se eliminan. Con SQL estándar, cuando quita un permiso, todos los permisos otorgados basados en ese permiso también se eliminaban. En MySQL, los permisos sólo pueden borrarse con comandos `REVOKE` explícitos o manipulando las tablas de permisos de MySQL.
- En MySQL, es posible tener el permiso `INSERT` sólo para algunas de las columnas en la tabla. En este caso, todavía puede ejecutar comandos `INSERT` en la tabla mientras omita esas columnas para las que no tiene el permiso `INSERT` . Las columnas omitidas obtienen su valor por defecto implícito si no está activado el modo SQL estricto. En modo estricto, el comando se rehúsa si algunas de las columnas omitidas no tienen valor por defecto. Sección 5.3.2, “El modo SQL del servidor” discute acerca del

modo estricto. Sección 13.1.5, “Sintaxis de CREATE TABLE” disctue acerca de los valores por defecto implícitos.

Las columnas para las que no tiene el permiso INSERT se ponen a su valor por defecto. SQL estándar requiere que tenga el permiso INSERT en todas las columnas.

En MySQL, si tiene el permiso INSERT sólo en alguna de las columnas de la tabla, puede ejecutar comandos INSERT — mientras omita las columnas para las que no tiene el permiso de su comando INSERT; tales columnas obtendrán su valor por defecto. En modo estricto (cuando sql_mode="traditional"), si alguna de las columnas omitidas no tiene valor por defecto, el comando INSERT se rehúsa.

13.5.1.4. Sintaxis de RENAME USER

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

El comando RENAME USER renombra cuentas de usuario MySQL existentes. Para usarlo, debe tener el permiso CREATE USER global o el permiso UPDATE para la base de datos mysql . Ocurre un error si cualquier de las antiguas cuentas no existe o cualquiera de las nuevas ya existe. Los valores *old_user* y *new_user* se dan igual que para el comando GRANT .

El comando RENAME USER se añadió en MySQL 5.0.2.

13.5.1.5. Sintaxis de SET PASSWORD

```
SET PASSWORD = PASSWORD('some password')
SET PASSWORD FOR user = PASSWORD('some password')
```

El comando SET PASSWORD asigna una contraseña a una cuenta de usuario MySQL existente.

La primera sintaxis asigna la contraseña para el usuario actual. Cualquier cliente que se conecte al servidor usando una cuenta no anónima puede cambiar la contraseña para la misma.

La segunda sintaxis asigna una contraseña para una cuenta específica en el servidor actual. Sólo los clientes con el permiso UPDATE para la base de datos mysql pueden hacerlo. El valor *user* debe darse en formato *user_name@host_name* donde *user_name* y *host_name* son exactamente los mismos que cuando se listan en las columnas User y Host de la tabla mysql.user . Por ejemplo, si tiene una entrada en las columnas User y Host con los valores 'bob' y '%.loc.gov', escribiría el comando así:

```
mysql> SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

Esto es equivalente al siguiente comando:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpass')
    -> WHERE User='bob' AND Host='%.loc.gov';
mysql> FLUSH PRIVILEGES;
```

Nota: Si se está conectando a un servidor MySQL 4.1 o posterior usando programas clientes anteriores a la 4.1, no use los comandos SET PASSWORD o UPDATE precedentes sin leer Sección 5.6.9, “Hashing de contraseñas en MySQL 4.1” primero. El formato de contraseña cambió en MySQL 4.1, y bajo ciertas circunstáncias, puede que no sea capaz de conectar al servidor.

En MySQL 5.0, puede ver su entrada de autenticación user@host ejecutando SELECT CURRENT_USER().

13.5.2. Sentencias para el mantenimiento de tablas

13.5.2.1. Sintaxis de `ANALYZE TABLE`

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

Este comando analiza y almacena la distribución de clave para una tabla. Durante el análisis, la tabla se bloquea con un bloqueo de lectura. En MySQL 5.0, funciona en tablas `MyISAM`, `BDB`, y `InnoDB` . Para tablas `MyISAM` , este comando es equivalente a usar `myisamchk -a`.

MySQL usa la distribución de claves almacenada para decidir el orden en que las tablas deben hacer los joins cuando realiza uno en algo que no sea una constante.

El comando retorna una tabla con las siguientes columnas:

Columna	Valor
`Tabla`	Nombre de tabla
`Op`	Siempre `analyze`
`Msg_type`	Es `status`, `error`, `info`, o `warning`
`Msg_text`	Mensaje

Puede chequear la distribución de claves almacenada con el comando `SHOW INDEX` . Consulte Sección 13.5.4.11, “Sintaxis de `SHOW INDEX`”.

Si la tabla no ha cambiado desde el último comando `ANALYZE TABLE` , la tabla no se vuelve a analizar.

En MySQL 5.0, los comandos `ANALYZE TABLE` se escriben en el log binario a no ser que la palabra clave `NO_WRITE_TO_BINLOG` opcional (o su alias `LOCAL`) se use.

13.5.2.2. Sintaxis de `BACKUP TABLE`

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory'
```

Nota: Este comando está obsoleto. Estamos trabajano en un mejor sustituto para este que proporcionará capacidades de copia de seguridad en línea. De momento, el script `mysqlhotcopy` puede usarse.

`BACKUP TABLE` copia al directorio de base de datos el mínimo número de ficheros de tablas necesarias para restaurar la tabla, tras volcar cualquier cambios almacenados en el buffer a disco. El comando funciona sólo para tablas `MyISAM` . Copia los ficheros de definición `.frm` y de datos `.MYD` . El fichero índice `.MYI` puede reconstruirse desde estos otros. El directorio debe especificarse con la ruta entera.

Antes de usar este comando consulte Sección 5.8.1, “Copias de seguridad de bases de datos”.

Durante la copia de seguridad, se realiza un bloqueo de lectura para cada tabla, uno cada vez, mientras se hace la copia. Si quier hacer una copia de seguridad de varias tablas como una muestra (evitando que ninguna de ellas se cambie durante la operación de la copia de seguridad), debe realizar un comando `LOCK TABLES` para obtener un bloqueo de lectura para cada tabla en el grupo.

El comando retorna una tabla con las siguientes columnas:

Columna	Valor
`Tabla`	Nombre de tabla
`Op`	Siempre `backup`

`Msg_type`	Es `status`, `error`, `info`, o `warning`
`Msg_text`	Mensaje

13.5.2.3. Sintaxis de `CHECK TABLE`

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

Chequea una tabla o tablas para errores. `CHECK TABLE` funciona para tablas `MyISAM` y `InnoDB` . Para tablas `MyISAM` , la estadística de clave se actualiza.

Desde MySQL 5.0.2, `CHECK TABLE` puede comprobar las vistas en busca de problemas tales como tablas que se referencian en la definición de la vista que ya no existe.

El comando `CHECK TABLE` retorna una tabla con las siguientes columnas:

Columna	Valor
`Tabla`	Nombre de tabla
`Op`	Siempre `check`
`Msg_type`	Es `status`, `error`, `info`, o `warning`
`Msg_text`	Mensaje

Tenga en cuenta que el comando puede producir varios registros para información de cada tabla chequeada. El último registro tiene un valor `Msg_type` de `status` y `Msg_text` normalmente debe ser `OK`. Si no obtiene `OK`, o `Table is already up to date` debe realizar una reparación de la tabla. Consulte Sección 5.8.3, “Mantenimiento de tablas y recuperación de un fallo catastrófico (crash)”. `Table is already up to date` significa que el motor de almacenamiento para la tabla indicada indica que no hay necesidad de chequear la tabla.

Las distintas opciones de chequeo que pueden darse se muestran en la siguiente tabla. Estas opciones se aplican sólo para tablas `MyISAM` y se ignoran en tablas `InnoDB` y vistas.

Tipo	Significado
`QUICK`	No escanea los registros para chequear enlaces incorrectos.
`FAST`	Sólo chequea tablas que no se han cerrado correctamente.
`CHANGED`	Sólo las tablas chequeadas que se han cambiado desde el úlitmo chequeo o no se han cerrado correctamente.
`MEDIUM`	Escanea registros para verificar que los enlaces borrados están bien. También calcula el checksum de la clave para los registros y lo verifica con el checksum calculado para las claves.
`EXTENDED`	Realiza una búsqueda completa para todas las claves para cada registro. Se asegura que la tabla es consistente 100%, pero tarda mucho tiempo!

Si ninguna de las opciones `QUICK`, `MEDIUM`, o `EXTENDED` se especifica, el tipo de chequeo por defecto para tablas de formato dinámico `MyISAM` es `MEDIUM`. Esto es lo mismo que ejecutar `myisamchk --medium-check tbl_name` en la tabla. El tipo de chequeo por defecto también es `MEDIUM` para tablas `MyISAM` de formato estático, a no ser que se especifique `CHANGED` o `FAST`. En tal caso, por defecto es `QUICK`. El escaneo de registro se evita para `CHANGED` y `FAST` porque los registros están corruptos muy raramente.

Puede combinar opciones de chequeo, como en el siguiente ejemplo, que realiza un chequeo rápido de la tabla para ver si se cerró correctamente:

```
CHECK TABLE test_table FAST QUICK;
```

Nota: En algunos casos, `CHECK TABLE` cambia la tabla. Esto ocurre si la tabla se marca como “corrupted” o “not closed properly” pero `CHECK TABLE` no encuentra ningún problema en la tabla. En este caso, `CHECK TABLE` marca la tabla como correcta.

Si una tabla está corrupta, es más probable que el problema esté en el índice y no en la parte de datos. Todos los tipos de chequeo chequean los índices profundamente y deberían encontrar la mayoría de errores.

Si quiere chequear una tabla que asume como correcta, no debe usar opciones de chequeo o la opción `QUICK`. Ésta debe usarse cuando tiene prisa y puede permitirse el pequeño riesgo que `QUICK` no encuentre un error en el fichero de datos. (En la mayoría de casos, MySQL debería encontrar, bajo uso normal, cualquier error en el fichero de datos. Si esto ocurre, la tabla se marca como “corrupted” y no puede usarse hasta que se repare.)

`FAST` y `CHANGED` están pensados para usar desde un script (por ejemplo, para ejecutarse desde `cron`) si quiere chequear sus tablas de vez en cuando. En la mayoría de casos, `FAST` se prefiere en lugar de `CHANGED`. (El único caso en que no es el método preferido es cuando sospecha que ha encontrado un bug en el código `MyISAM` .)

`EXTENDED` debe usarse sólo después de ejecutar un chequeo normal pero todavía obtiene errores extraños de la tabla cuando MySQL intenta actualizar un registro o encuentra un registro mediante la clave. (Esto es muy improbable si un chequeo normal ha tenido éxito.)

Algunos problemas reportados por `CHECK TABLE` no pueden corregirse automáticamente:

- `Found row where the auto_increment column has the value 0`.

  Esto significa que tiene un registro en la tabla donde la columna `AUTO_INCREMENT` contiene un valor de índice de 0. (Es posible crear un registro donde la columna `AUTO_INCREMENT` es 0 poneindo la columna explícitamente a 0 con un comando `UPDATE` .)

  Esto no es un error por si mismo, pero puede causar problemas si decide volcar la tabla y restaurarla o realizar un `ALTER TABLE` en la tabla. En este caso, la columna `AUTO_INCREMENT` cambia los valores según las reglas de las columnas `AUTO_INCREMENT` , que pueden causar problemas tales como errores de clave duplicada.

  Para evitar las advertencias, símplemente ejecute un comando `UPDATE` para poner en la columna un valor distinto a 0.

13.5.2.4. Sintaxis de `CHECKSUM TABLE`

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

Reporta un checksum de tabla.

Si `QUICK` se especifica, el checksum de la tabla se reporta si está disponible, o `NULL` en otro caso. Esto es muy rápido. Un checksum en vivo está permitido especificando la opción de tabla `CHECKSUM=1` , actualmente sólo soportado por tablas `MyISAM` . Consulte Sección 13.1.5, “Sintaxis de `CREATE TABLE`”.

En modo `EXTENDED` la tabla completa se lee registro a registro y se calcula el checksum. Esto puede ser muy lento para tablas grandes.

Por defecto, si no se especifica ni `QUICK` ni `EXTENDED` , MySQL retorna un checksum en vivo si el motor de tabla lo soporta y escanea la tabla de otro modo.

`CHECKSUM TABLE` retorna `NULL` para tablas no existentes. Desde MySQL 5.0.3, se genera una advertencia para esta condición.

13.5.2.5. Sintaxis de `OPTIMIZE TABLE`

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

`OPTIMIZE TABLE` debe usarse si ha borrado una gran parte de la tabla o si ha hecho varios cambios en una tabla con registros de longitud variable (tablas que tienen columnas `VARCHAR`, `BLOB`, o `TEXT`). Los registros borrados se mantienen en una lista enlazada y operaciones `INSERT` posteriores reúsan posiciones de antiguos registros. Puede usar `OPTIMIZE TABLE` para reclamar el usuario no usado y para defragmentar el fichero de datos.

En la mayoría de inicializaciones, no necesita ejecutar `OPTIMIZE TABLE` para nada. Incluso si hace muchas actualizaciones a registros de longitud variables, no es probable que necesite hacerlo más de una vez a la semana o mes y sólo en ciertas tablas.

Actualmente, `OPTIMIZE TABLE` funciona sólo en tablas `MyISAM`, `BDB` y `InnoDB` .

Para tablas `MyISAM` , `OPTIMIZE TABLE` funciona como sigue:

1. Si la tabla ha borrado o dividido registros, repare la tabla.
2. Si las páginas índice no están ordenadas, ordénelas.
3. Si las estadísticas no están actualizadas (y la reparación no puede hacerse ordenando el índice), actualícelas.

Para tablas `BDB` , `OPTIMIZE TABLE` es mapea como `ANALYZE TABLE`. Para tablas `InnoDB` , se mapea con `ALTER TABLE`, que reconstruye la tabla. Reconstruye las estadísticas actualizadas de índice y libera espacio no usado en el índice clusterizado. Consulte Sección 13.5.2.1, “Sintaxis de `ANALYZE TABLE`”.

Puede hacer que `OPTIMIZE TABLE` funcione con otros tipos de tabla arrancando `mysqld` con la opción `--skip-new` o `--safe-mode` ; en este caso `OPTIMIZE TABLE` se mapea con `ALTER TABLE`.

Tenga en ceunta que MySQL bloquea la tabla mientras se ejecuta `OPTIMIZE TABLE` .

En MySQL 5.0, los comandos `OPTIMIZE TABLE` se escriben en el log binario a no ser que la palabra `NO_WRITE_TO_BINLOG` opcional(o su alias `LOCAL`) se use. Esto se hace para que los comandos `OPTIMIZE TABLE` se usen en MySQL server actuando como maestro de replicación se replique por defecto en el esclavo de replicación.

13.5.2.6. Sintaxis de `REPAIR TABLE`

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
    tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` repara una tabla posiblemente corrupta. Por defecto, tiene el mismo efecto que `myisamchk --recover tbl_name`. `REPAIR TABLE` funciona sólo en tablas `MyISAM`.

Normalmente nunca debe ejecutar este comando. Sin embargo, si hay un desastre, `REPAIR TABLE` puede recuperar todos los datos de una tabla `MyISAM` . Si sus tablas se corrompen a menudo, debe intentar encontrar la razón de lo que lo causa, para eliminar la necesidad de usar `REPAIR TABLE`.

Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”. Consulte Sección 14.1.4, “Problemas en tablas MyISAM”.

El comando retorna una tabla con las siguientes columnas:

Columna	Valor
Tabla	Nombre de tabla
Op	Siempre es repair
Msg_type	Es status, error, info, o warning
Msg_text	Mensaje

El comando REPAIR TABLE puede producir muchos registros de información para cada tabla reparada. El último registro tiene un valor Msg_type de status y Msg_test normalmente debe ser OK. Si no obtiene OK, debe intentar reparar la tabla con myisamchk --safe-recover, ya que REPAIR TABLE no implementa todas las opciones de myisamchk. Plaenamos hacerlo más flexible en el futuro.

Si se da QUICK , REPAIR TABLE intenta reparar sólo el árbol índice. Este tipo de reparación es como lo que hace myisamchk --recover --quick.

Si usa EXTENDED, MySQL crea el índice registro a registro en lugar de crear un índice a la vez ordenando. Este tipo de reparación es como el hecho por myisamchk --safe-recover.

También hay un modo USE_FRM disponible en MySQL 5.0 para REPAIR TABLE. Use esto si el fichero índice .MYI no existe o su cabecera está corrupta. En este modo, MySQL recrea el fichero .MYI usando información desde el fichero .frm . Este tipo de reparación no puede hacerse con myisamchk. **Nota**: Use este modo **sólo** si no puede usar modos REPAIR normalmente. La cabecera .MYI contiene información importante de metadatos (en particular, los valores actuales AUTO_INCREMENT y Delete link) que se pierden en REPAIR ... USE_FRM. No use USE_FRM si la tabla está comprimida, ya que esta información se almacena en el fichero .MYI .

En MySQL 5.0, los comandos REPAIR TABLE se escriben en el log binario a no ser que la palabra opcional NO_WRITE_TO_BINLOG (o su alias LOCAL) se use.

Atención: Si el servidor muere durante una operación REPAIR TABLE, es esencial tras restaurarla que inmediatamente ejecute otro comando REPAIR TABLE para la tabla antes de realizar cualquier otra operación en ella. (Siempre es una buena idea empezar haciendo una copia de seguridad.) En el peor caso, puede tener un nuevo fichero índice limpio sin información acerca del fichero de datos, y luego la siguiente operación que realice puede sobreescribir el fichero de datos. Este es un escenario improbable pero posible.

13.5.2.7. Sintaxis de RESTORE TABLE

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

Restaura la tabla o tablas de una copia de seguridad que se hizo con BACKUP TABLE. Las tablas existentes no se sobreescriben; si trata restaurar una tabla existente, obtiene un error. Pero como BACKUP TABLE, RESTORE TABLE actualmente funciona sólo para tablas MyISAM . El directorio debe especificarse como una ruta completa.

La copia de seguridad para cada tabla consiste en su fichero de formato .frm y fichero de datos .MYD . La operación de restauración restaura aquellos ficheros, luego los usa para reconstruir el fichero índice .MYI. La restauración tarda más tiempo que la copia de seguridad debido a la necesidad de reescribir los índices. Mientras más índices tenga la tabla, más tarda.

El comando retorna una tabla con las siguientes columnas:

Columna	Valor
Tabla	Nombre de tabla
Op	Siempre restore
Msg_type	Es status, error, info, o warning
Msg_text	Mensaje

13.5.3. Sintaxis de SET

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
      user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | @@[global. | session.]system_var_name = expr
```

SET inicializa distintos tipos de variables que afectan la operación del servidor o de su cliente. Puede usarse para asignar valores a las variables de usuario o de sistema.

El comando SET PASSWORD para asignar contraseñas de cuenta se describen en Sección 13.5.1.5, “Sintaxis de SET PASSWORD”.

La mayoría del sistema puede cambiarse en tiempo de ejecución. Las variables de sistema que pueden describirse dinámicamente se describen en Sección 5.3.3.1, “Variables de sistema dinámicas”.

Nota: Las versiones antiguas de MySQL empleaban SET OPTION para este comando, pero su uso está obsoleto en favor de SET.

El siguiente ejemplo muestra las distintas sintaxis que puede usar para cambiar las variables.

Una variable de usuario se escribe como *@var_name* y puede cambiarse como sigue:

```
SET @var_name = expr;
```

Más información sobre variables de usuario se da en Sección 9.3, “Variables de usuario”.

Se puede referir a las variables de sistema en comandos SET como *var_name*. El nombre puede ir precedido opcionalmente por GLOBAL o @@global. para indicar explícitamente que la variable es global, o por SESSION, @@session., o @@ para indicar que es una variable de sesión. LOCAL y @@local. son sinónimos para SESSION y @@session.. Si no hay modificador presente, SET asigna un valor a la variable de sesión.

La sintaxis *@@var_name* para variables de sistema se soporta para hacer la sintaxis de MySQL compatible con otros sistemas de base de datos.

Si cambia varias variables de sistema en el mismo comando, la última opción GLOBAL o SESSION usada se usa para variables que no tienen modo especificado.

```
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;
```

Si cambia una variable de sistema usando SESSION (por defecto), el valor queda en efecto hasta que la sesión actual finaliza o hasta que cambia la variable con un valor distinto. Si cambia la variable de sistema usando GLOBAL, que requiere el permiso SUPER , el valor se recuerda y se usa para nuevas conexiones hasta que el servidor se reinicia. Si quiere hacer un cambio de variable permanente, debe ponerlo en un fichero de opciones. Consulte Sección 4.3.2, “Usar ficheros de opciones”.

Para evitar uso incorrecto, MySQL produce un error si usa SET GLOBAL con una variable que sólo puede ser usada con SET SESSION o si no especifica GLOBAL (o @@) cuando cambie una variable global.

Si quiere cambiar una variable SESSION al valor GLOBAL o un valor GLOBAL al valor de compilación de MySQL por defecto, puede hacerlo con DEFAULT. Por ejemplo, los siguientes dos comandos son idénticos en cambiar los valores de sesión o max_join_size al valor global:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Puede obtener una lista de la mayoría de variables de sistema con SHOW VARIABLES. (Consulte Sección 13.5.4.21, “Sintaxis de SHOW VARIABLES”.) Para obtener un nombre de variable específico, use una cláusula LIKE commo se muestra:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW GLOBAL VARIABLES LIKE 'max_join_size';
```

Para obtener una lista de variables cuyos nombres coinciden con un patrón, use el comodín '%':

```
SHOW VARIABLES LIKE 'have%';
SHOW GLOBAL VARIABLES LIKE 'have%';
```

El comodín puede usarse en cualquier posición dentro del patrón para coincidir.

Puede obtener el valor de un valor específico usando la sintaxis @@[global.|local.]var_name con SELECT:

```
SELECT @@max_join_size, @@global.max_join_size;
```

Cuando recibe una variable con SELECT @@var_name (esto es,no especifica global., session., o local.), MySQL retorna el valor SESSION si existe y el valor GLOBAL en otro caso.

La siguiente lista describe variables que tienen sintaxis no estándar o que no se describe en la lista de variables de sistema que se encuentra en Sección 5.3.3, “Variables de sistema del servidor”. Aunque estas variables no se muestran con SHOW VARIABLES, puede obtener sus valores con SELECT (con la excepción de CHARACTER SET y SET NAMES). Por ejemplo:

```
mysql> SELECT @@AUTOCOMMIT;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
```

- AUTOCOMMIT = {0 | 1}

  Pone el modo autocommit . Con valor 1, todos los cambios de una tabla toman efecto inmediatamente. Si se pone a 0, debe usar COMMIT para aceptar una transacción o ROLLBACK para cancelarla. Si cambia el modo AUTOCOMMIT de 0 a 1, MySQL realiza un COMMIT automático de cualquier transacción

abierta . Otra forma de comenzar una transacción es usar un comando START TRANSACTION o BEGIN. Consulte Sección 13.4.1, “Sintaxis de START TRANSACTION, COMMIT y ROLLBACK”.

- BIG_TABLES = {0 | 1}

  Si se pone a 1, todas las tablas temporales se almacenan en disco en lugar que en memoria. Esto es un poco lento, pero el error The table *tbl_name* is full no ocurre para operaciones SELECT que requieran una tabla temporal grande. El valor por defecto para una nueva conexión es 0 (use tablas temporales en memoria). Normalmente, nunca debería necesitar usar esta variable, ya que MySQL 5.0 convierte automáticamente tablas en memoria a tablas en disco como se requiere. (*Nota*: Esta variable se llamaba préviamente SQL_BIG_TABLES.)

- CHARACTER SET {*charset_name* | DEFAULT}

  Esto mapea todas las cadenas desde y hacia el cliente con el mapeo dado. Puede añadir nuevos mapeos editando sql/convert.cc en la distribución fuente MySQL. En MySQL 5.0, SET CHARACTER SET cambia tres variables de sistema character_set_client y character_set_results se actualizan con el conjunto de caracteres dado, y character_set_connection al valor de character_set_database.

  El mapeo por defecto puede restaurarse usando el valor DEFAULT.

  Tenga en cuenta que la sintaxis para SET CHARACTER SET difiere de la de la mayoría de otras opciones.

- FOREIGN_KEY_CHECKS = {0 | 1}

  Con valor de 1 (por defecto), las claves foráneas para tablas InnoDB se chequean. Si se pone a 0, se ignoran. Deshabilitar el chequeo de clave foránea puede ser útil para recargar tablas InnoDB en un orden distinto que el requerido por sus relaciones padre/hijo, Consulte Sección 15.6.4, “Restricciones (constraints) FOREIGN KEY”.

- IDENTITY = *value*

  La variable es un sinónimo para la variable LAST_INSERT_ID . Existe por compatibilidad con otras bases de datos. Puede leer su valor con SELECT @@IDENTITY, y cambiarlo mediante SET IDENTITY.

- INSERT_ID = *value*

  Cambia el valor a ser usado por los comandos INSERT o ALTER TABLE al insertar un valor AUTO_INCREMENT . Esto se usa principalmente con el lob binario.

- LAST_INSERT_ID = *value*

  Cambia el valor a ser retornado de LAST_INSERT_ID(). Esto se almacena en el log binario cuando usa LAST_INSERT_ID() en un comando que actualice una tabla. Cambiar esta variable no actualiza el valor retornado por la función de la mysql_insert_id() API de C.

- NAMES {'*charset_name*' | DEFAULT}

  SET NAMES cambia tres variables de sesión de sistema character_set_client, character_set_connection, y character_set_results al conjunto de caracteres dado. Cambiar character_set_connection a charset_name también cambia collation_connection a la colación por defecto para charset_name.

  El mapeo por defecto puede restaurarse usando un valor de DEFAULT.

Tenga en cuenta que la sintaxis para SET NAMES difiere de la usada para la mayoría de otras opciones.

- SQL_NOTES = {0 | 1}

  Con el valor 1 (por defecto), advertencias del nivel Note se registran. Con valor 0, las advertencias Note se suprimen. mysqldump incluye la salida para cambiar esta variable a 0 así que recargar el fichero volcado no produce advertencias para eventos que no afectan a la integridad de la operación de recarga. SQL_NOTES se añadió en MySQL 5.0.3.

- SQL_AUTO_IS_NULL = {0 | 1}

  Con valor 1 (por defecto), puede encontrar el último registro insertado para una tabla que contiene una columna AUTO_INCREMENT usando el siguiente constructor:

```
WHERE auto_increment_column IS NULL
```

  Este comportamiento lo usan algunos programas ODBC, como Access.

- SQL_BIG_SELECTS = {0 | 1}

  Con valor 0, MySQL aborta los comandos SELECT que probablemente tardarán mucho tiempo (esto es, comandos para los que el optimizador estima que el número de registros examinados excede el valor de max_join_size). Esto es útil cuando un comando WHERE no aconsejable se ejecuta. El valor por defecto para una nueva conexión es 1, que permite todos los comandos SELECT .

  Si cambia la variable de sistema max_join_size a un valor distinto a DEFAULT, SQL_BIG_SELECTS se pone a 0.

- SQL_BUFFER_RESULT = {0 | 1}

  SQL_BUFFER_RESULT fuerza los resultados de los comandos SELECT a poner en tablas temporales. Esto ayuda a MySQL a liberar los bloqueos de tabla rápidamente y pueden ser beneficioso en caso que tarde un largo tiempo para enviar resultados al cliente.

- SQL_LOG_BIN = {0 | 1}

  Con valor 0, no se realiza logueo en el log binario para el cliente. El cliente debe tener el permiso SUPER para cambiar esta opción.

- SQL_LOG_OFF = {0 | 1}

  Con valor 1, no se realiza logueo en el log de consultas generales para el cliente. El cliente debe tener el permiso SUPER para cambiar esta opción.

- SQL_LOG_UPDATE = {0 | 1}

  Esta variable está obsoleta, y es mapea a SQL_LOG_BIN.

- SQL_QUOTE_SHOW_CREATE = {0 | 1}

  Con valor 1, SHOW CREATE TABLE entrecomilla los nombres de tabla y columnas. Si se pone a 0, se desactiva el entrecomillado. Esta opción está activada por defecto, así que la replicación funciona para tablas con nombres de tabla y columna que no lo requieren. Consulte Sección 13.5.4.5, “Sintaxis de SHOW CREATE TABLE”.

- SQL_SAFE_UPDATES = {0 | 1}

Con valor 1, MySQL aborta comandos UPDATE o DELETE que no usan una clave en la cláusula WHERE o LIMIT . Esto hace posible cazar los comandos UPDATE o DELETE donde las claves no se usan apropiadamente y que probablemente cambiarían o borrarían un gran número de registros.

- SQL_SELECT_LIMIT = {*value* | DEFAULT}

  El máximo número de registros a retornar desde comandos SELECT . El valor por defecto para una nueva conexión es “unlimited.” Si cambia este límite, el valor por defecto puede restaurarse usando un valor SQL_SELECT_LIMIT de DEFAULT.

  Si un SELECT tiene una cláusula LIMIT el LIMIT tiene preferencia sobre el valor de SQL_SELECT_LIMIT.

  SQL_SELECT_LIMIT no se aplica a comandos SELECT ejecutados en rutinas almacenadas. Tampoco se aplica a comandos SELECT que no producen un conjunto de resultados a ser retornado al cliente. Esto incluye comandos SELECT en subconsultas, CREATE TABLE ... SELECT, y INSERT INTO ... SELECT.

- SQL_WARNINGS = {0 | 1}

  Esta variable controla si comandos INSERT de un registro producen una cadena de información si hay una advertencia. Por defecto es 0. Cambie el valor a 1 para producir una cadena de información.

- TIMESTAMP = {*timestamp_value* | DEFAULT}

  Cambia la hora del cliente. Se usar para obtener la fecha y hora original si usa el log binario para restaurar registros. timestamp_value debe ser un Unix epoch timestamp, no un timestamp de MySQL.

- UNIQUE_CHECKS = {0 | 1}

  Con valor 1 (por defecto), se realizan chequeos en tablas InnoDB para índices secundarios. Con valor 0, no se hacen chequeos de valores únicos para entradas de índices insertados en el búffer de inserción de InnoDB. Si sabe con certeza que sus datos no contienen violaciones de valores únicos, puede ponerlo a 0 para acelerar importaciones de tablas grandes a InnoDB.

13.5.4. Sintaxis de SHOW

SHOW tiene varias formas que proporcionan información acerca de bases de datos, tablas, columnas o información de estado acerca del servidor. Esta sección describe estos puntos:

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW ENGINE engine_name {LOGS | STATUS }
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW [BDB] LOGS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW TRIGGERS
```

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]
```

El comando SHOW también tiene formas que proporcionan información acerca de servidores de replicación maestros y esclavos y se describen en Sección 13.6, “Sentencias de replicación”:

```
SHOW BINLOG EVENTS
SHOW MASTER LOGS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
```

En la sintaxis para un comando SHOW dado incluye una parte LIKE 'pattern' , 'pattern' es una cadena que puede contener los caracteres de SQL '%' y '_' . El patrón es útil para restringir la salida del comando para valores coincidentes.

13.5.4.1. Sintaxis de SHOW CHARACTER SET

```
SHOW CHARACTER SET [LIKE 'pattern']
```

El comando SHOW CHARACTER SET muestra todos los conjuntos de caracteres disponibles. Esto tiene una cláusula LIKE opcional que indica qué nombres de conjuntos de caracteres hay coincidentes. Por ejemplo:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | ISO 8859-1 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 |
+---------+-----------------------------+-------------------+--------+
```

La columna Maxlen muestra el máximo número de bytes usados para almacenar un carácter.

13.5.4.2. Sintaxis de SHOW COLLATION

```
SHOW COLLATION [LIKE 'pattern']
```

La salida de SHOW COLLATION incluye todos los conjuntos de caracteres disponibles. Tiene una cláusula LIKE opcional cuyo *pattern* indica qué nombres de colación coinciden. Por ejemplo:

```
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 |
+-------------------+---------+----+---------+----------+---------+
```

La columna Default indica si una colación está por defecto para su conjunto de caracteres. Compiled indica si el conjunto de caracteres está compilado en el servidor. Sortlen está relacionado con la cantidad de memoria requerida para ordenar cadenas expresadas en el conjunto de caracteres.

13.5.4.3. Sintaxis de SHOW COLUMNS

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

SHOW COLUMNS muestra información acerca de las columnas en una tabla dada. También funciona para vistas desde MySQL 5.0.1.

Si los tipos de columnas difieren de los que espera basados en su comando CREATE TABLE, tenga en cuenta que MySQL a veces cambia tipos de columnas cuando crea o altera una tabla. Las condiciones en que esto ocurre se describen en Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”.

La palabra clave FULL hace que la salida incluya los permisos que tiene así como cualquier comentario por columna para cada columna.

Puede usar *db_name.tbl_name* como alternativa a la sintaxis *tbl_name* FROM *db_name* . En otras palabras, estos dos comandos son equivalentes:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW FIELDS es un sinónimo para SHOW COLUMNS. Puede listar las columnas de una tabla con el comando mysqlshow *db_name tbl_name* .

El comando DESCRIBE proporciona información similar a SHOW COLUMNS. Consulte Sección 13.3.1, “Sintaxis de DESCRIBE (Información acerca de las columnas)”.

13.5.4.4. Sintaxis de SHOW CREATE DATABASE

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

Muestra un comando CREATE DATABASE que crea la base de datos dada. SHOW CREATE SCHEMA puede usarse desde MySQL 5.0.2.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

13.5.4.5. Sintaxis de SHOW CREATE TABLE

```
SHOW CREATE TABLE tbl_name
```

Muestra un comando CREATE TABLE que crea la tabla dada. Desde MySQL 5.0.1, este comando funciona con vistas.

```
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 entrecomilla los nombres de tabla y columna según el valor de la opción SQL_QUOTE_SHOW_CREATE . Sección 13.5.3, “Sintaxis de SET”.

13.5.4.6. Sintaxis de SHOW DATABASES

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

SHOW DATABASES lista la base de datos en el servidor MySQL . Puede obtener esta lista usando el comando mysqlshow . En MySQL 5.0, ve sólo las bases de datos para las que tiene alguna clase de permiso, si no tiene el permiso SHOW DATABASES .

Si el servidor se arrancó con la opción --skip-show-database , no puede usar este comando a no ser que tenga el permiso SHOW DATABASES .

SHOW SCHEMAS puede usarse desde MySQL 5.0.2

13.5.4.7. Sintaxis de SHOW ENGINE

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

SHOW ENGINE muestra información de log o estado acerca de motores de almacenamiento. Los siguientes comandos se soportan actualmente:

```
SHOW ENGINE BDB LOGS
SHOW ENGINE INNODB STATUS
```

SHOW ENGINE BDB LOGS muestra información de estado acerca de ficheros de log BDB existentes. Retorna los siguientes campos:

- File

  Ruta completa al fichero de log.

- Type

  Tipo del fichero de log (BDB para ficheros de log Berkeley DB).

- Status

  Estado del fichero de log (FREE si el fichero puede borrarse, o IN USE si el fichero se necesita en el subsistema de transacción)

SHOW ENGINE INNODB STATUS muestra información extendida acerca del estado del comando InnoDB .

Sinónimos antiguos (ahora obsoletos) para estos comandos son SHOW [BDB] LOGS y SHOW INNODB STATUS.

SHOW ENGINE puede usarse desde MySQL 4.1.2.

13.5.4.8. Sintaxis de SHOW ENGINES

```
SHOW [STORAGE] ENGINES
```

SHOW ENGINES muestra su información de estado acerca del motor de almacenamiento. Esto es particularmente útil para chequear si un motor de almacenamiento se soporta, o para ver si el motr es. SHOW TABLE TYPES es un sinónimo obsoleto.

```
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
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)
```

Un valor `Support` indica si el motor se soporta, y cuál está activo por defecto. Por ejemplo, si el servidor se arranca con la opción `--default-table-type=InnoDB`, el valor `Support` para el registro `InnoDB` tiene el valor `DEFAULT`.

13.5.4.9. Sintaxis de `SHOW ERRORS`

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

Este comando es similar a `SHOW WARNINGS`, excepto que en lugar de mostrar errores, advertencias, y notas sólo muestra errores.

La cláusula `LIMIT` tiene la misma sintaxis que para el comando `SELECT` . Consulte Sección 13.2.7, “Sintaxis de SELECT”.

El comando `SHOW COUNT(*) ERRORS` muestra el número de errores. Puede recibir este número de la variable `error_count` :

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

Para más información consulte Sección 13.5.4.22, “Sintaxis de SHOW WARNINGS”.

13.5.4.10. Sintaxis de `SHOW GRANTS`

```
SHOW GRANTS FOR user
```

Este comando lista el comando `GRANT` que debe realizarse para duplicar los permisos para una cuenta de usuario MySQL.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

Para listar los permisos de la sesión actual, puede usar cualquiera de los siguientes comandos:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

13.5.4.11. Sintaxis de SHOW INDEX

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

SHOW INDEX retorna información de índice de tabla en un formato que recuerda la llamada SQLStatistics en ODBC.

SHOW INDEX returna los siguientes campso:

- Table

  Nombre de tabla.

- Non_unique

  0 si el índice no puede contener duplicados, 1 si puede.

- Key_name

  Nombre del índice

- Seq_in_index

  Número de secuencia de columna en el índice, comenzando con 1.

- Column_name

  Nombre de columna.

- Collation

  Cómo se ordena la columna en el índice. En MySQL, puede tener valores 'A' (Ascendente) o NULL (No ordenado).

- Cardinality

  Número de valores únicos en el índice. Se actualiza ejecutando ANALYZE TABLE o myisamchk -a. Cardinality se cuenta basándose en las estadísticas almacenadas como enteros, así que no es necesariamente precisa para tablas pequeñas. Mientras más grande sea, más grande es la probabilidad que MySQL use el índice al hacer joins.

- Sub_part

  Número de caracteres indexados si la columna sólo está indexada parcialmente. NULL si la columna entera está indexada.

- Packed

  Indica cómo está empaquetada la clave. NULL si no lo está.

- Null

  Contiene YES si la columna puede contener NULL. Si no, la columna contiene NO desde MySQL 5.0.3, y '' antes.

- Index_type

  Método de índice usado (BTREE, FULLTEXT, HASH, RTREE).

- Comment

  Comentarios varios.

Puede usar *db_name.tbl_name* como alternativa para la sintaxis *tbl_name* FROM *db_name* . Estos dos comandos son equivalentes:

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

SHOW KEYS es sinónimo para SHOW INDEX. Puede listar los índices de una tabla con el comando mysqlshow -k *db_name tbl_name* .

13.5.4.12. Sintaxis de SHOW INNODB STATUS

```
SHOW INNODB STATUS
```

En MySQL 5.0, este es un sinónimo obsoleto para SHOW ENGINE INNODB STATUS. Consulte Sección 13.5.4.7, “Sintaxis de SHOW ENGINE”.

13.5.4.13. Sintaxis de SHOW LOGS

```
SHOW [BDB] LOGS
```

En MySQL 5.0, este es un sinónimo obsoleto para SHOW ENGINE BDB LOGS. Consulte Sección 13.5.4.7, “Sintaxis de SHOW ENGINE”.

13.5.4.14. Sintaxis de SHOW OPEN TABLES

```
SHOW OPEN TABLES
```

SHOW OPEN TABLES lista las tablas no TEMPORARY abiertas actualmente en la caché de tablas. Consulte Sección 7.4.8, “Cómo abre y cierra tablas MySQL”.

SHOW OPEN TABLES returna los siguientes campos:

- Database

  La base de datos que contiene la tabla.

- Table

  Nombre de tabla.

- In_use

  Número de veces que la tabla está en uso para consultas. Si el contador es cero, la tabla está abierta, pero no está siendo usada.

- Name_locked

Si un nombre de tabla está bloqueado. El bloqueo de nombres se usa para operaciones tales como borrar o renombrar tablas.

`SHOW OPEN TABLES` se añadió en MySQL 3.23.33.

13.5.4.15. Sintaxis de `SHOW PRIVILEGES`

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` muestra la lista de permisos de sistema que soporta MySQL server. La salida exacta depende de la versión de su servidor.

```
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: Functions,Procedures
Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
*************************** 6. row ***************************
Privilege: Create view
Context: Tables
Comment: To create new views
*************************** 7. row ***************************
Privilege: Create user
Context: Server Admin
Comment: To create new users
*************************** 8. row ***************************
Privilege: Delete
Context: Tables
Comment: To delete existing rows
*************************** 9. row ***************************
Privilege: Drop
Context: Databases,Tables
Comment: To drop databases, tables, and views
*************************** 10. row ***************************
Privilege: Execute
Context: Functions,Procedures
Comment: To execute stored routines
*************************** 11. row ***************************
Privilege: File
Context: File access on server
Comment: To read and write files on the server
*************************** 12. row ***************************
Privilege: Grant option
Context: Databases,Tables,Functions,Procedures
Comment: To give to other users those privileges you possess
*************************** 13. row ***************************
Privilege: Index
```

```
Context: Tables
Comment: To create or drop indexes
*************************** 14. row ***************************
Privilege: Insert
Context: Tables
Comment: To insert data into tables
*************************** 15. row ***************************
Privilege: Lock tables
Context: Databases
Comment: To use LOCK TABLES (together with SELECT privilege)
*************************** 16. row ***************************
Privilege: Process
Context: Server Admin
Comment: To view the plain text of currently executing queries
*************************** 17. row ***************************
Privilege: References
Context: Databases,Tables
Comment: To have references on tables
*************************** 18. row ***************************
Privilege: Reload
Context: Server Admin
Comment: To reload or refresh tables, logs and privileges
*************************** 19. row ***************************
Privilege: Replication client
Context: Server Admin
Comment: To ask where the slave or master servers are
*************************** 20. row ***************************
Privilege: Replication slave
Context: Server Admin
Comment: To read binary log events from the master
*************************** 21. row ***************************
Privilege: Select
Context: Tables
Comment: To retrieve rows from table
*************************** 22. row ***************************
Privilege: Show databases
Context: Server Admin
Comment: To see all databases with SHOW DATABASES
*************************** 23. row ***************************
Privilege: Show view
Context: Tables
Comment: To see views with SHOW CREATE VIEW
*************************** 24. row ***************************
Privilege: Shutdown
Context: Server Admin
Comment: To shut down the server
*************************** 25. row ***************************
Privilege: Super
Context: Server Admin
Comment: To use KILL thread, SET GLOBAL, CHANGE MASTER, etc.
*************************** 26. row ***************************
Privilege: Update
Context: Tables
Comment: To update existing rows
*************************** 27. row ***************************
Privilege: Usage
Context: Server Admin
Comment: No privileges - allow connect only
```

13.5.4.16. Sintaxis de SHOW PROCESSLIST

```
SHOW [FULL] PROCESSLIST
```

`SHOW PROCESSLIST` le muestra qué flujos están en ejecución. Puede obtener esta información usando el comando `mysqladmin processlist` . Si tiene el permiso `SUPER` , puede ver todos los flujos. De

otro modo, puede ver sólo los propios (esto es, flujos asociados con la cuenta MySQL que está usando). Consulte Sección 13.5.5.3, “Sintaxis de KILL”. Si no usa la palabra clave FULL , sólo los primeros 100 caracteres de cada consulta se muestran.

En MySQL 5.0, el comando reporta el nombre de equipo ara conexiones TCP/IP en formato *host_name*:*client_port* para hacer más fácil determinar qué hace cada cliente.

Este comando es útil si obtiene el mensaje de error "demasiadas conexiones" para encontrar qué ocurre. MySQL reserva una conexión extra para usar por cuentas con el permiso SUPER, para asegurar que el administrador siempre es capaz de conectar y comprobar el sistema (asumiendo que no da este permiso a todos los usuarios).

Algunos estados vistos comúnmente en la salida de SHOW PROCESSLIST:

- Checking table

  El flujo está realizando un chequeo (automático) de la tabla.

- Closing tables

  Significa que el flujo está volcando los datos que han cambiado de la tabla a disco y cerrando las tablas usadas. Esto debe ser una operación rápido. Si no lo es, debe verificar que no tiene el disco lleno y que el disco no tiene un uso muy pesado.

- Connect Out

  Esclavo conectando con el maestro.

- Copying to tmp table on disk

  El conjunto de resultados temporal era mayor que tmp_table_size y el flujo está cambiando la tabla temporal de memoria a disco para ahorrar memoria.

- Creating tmp table

  El flujo está creando una tabla temporal para guardar parte del resultado de una consulta.

- deleting from main table

  El servidor está ejecutando la primera parte de un borrado de tablas múltiple y borrando sólo la primera tabla.

- deleting from reference tables

  El servidor está ejecutando la segunda parte de un borrado de tablas múltiples y borrando los registros coincidentes de las otras tablas.

- Flushing tables

  El flujo está ejecutando FLUSH TABLES y espera a que todos los flujos cierren sus tablas.

- Killed

  Alguien ha enviado un comando KILL al flujo y debería abortar en cuanto chequee el flag kill. El flag se chequea en cada vuelta al bucle principal de MySQL, pero en algunos casos puede tardar algo de tiempo en que muera el flujo. Si el flujo está bloqueado por algún otro flujo, el kill tiene efecto en cuanto el otro flujo libera el bloqueo.

- Locked

La consulta está bloqueada por otra consulta.

- `Sending data`

  El flujo está procesando registros para un comando `SELECT` y también enviando datos al cliente.

- `Sorting for group`

  El flujo está ordenando para un `GROUP BY`.

- `Sorting for order`

  El flujo está ordenando para un `ORDER BY`.

- `Opening tables`

  El flujo está intentando abrir una tabla. Esto debería ser un proceso muy rápido, a no ser que algo importante evite la abertura. Por ejemplo, un comando `ALTER TABLE` o `LOCK TABLE` puede evitar abrir una tabla hasta que el comando acabe.

- `Removing duplicates`

  La consulta usaba `SELECT DISTINCT` de forma que MySQL no podía optimizar las distintas operaciones en una fase temprana. Debido a ello, MySQL necesita una fase extra para borrar todos los registros duplicados antes de enviar el resultado al cliente.

- `Reopen table`

  El flujo obtivo un bloqueo para la tabla, pero se dio cuenta tras obtenerlo que la estructura de la tabla cambió. Se libera el bloqueo, cierra la tabla y trata de reabrirla.

- `Repair by sorting`

  El código de reparación está usando una ordenación para crear índices.

- `Repair with keycache`

  El código de reparación está usando creación de claves una a una en la caché de claves. Esto es mucho más lento que `Repair by sorting`.

- `Searching rows for update`

  El flujo hace una primera fase para encontrar todos los registro coincidentes antes de actualizarlos. Esto debe hacerse si `UPDATE` está cambiando el índice que se usa para encontrar los registros implicados.

- `Sleeping`

  El flujo espera que el cliente envíe un nuevo comando .

- `System lock`

  El flujo espera obtener un bloqueo de sistema externo para la tabla. Si no está usando múltiples servidors `mysqld` accediendo a las mismas tablas, puede deshabilitar los bloqueos de sistema con la opción `--skip-external-locking` .

- `Upgrading lock`

  El handler `INSERT DELAYED` trata de obtener un bloqueo para la tabla para insertar registros.

- `Updating`

  El flujo está buscando registros para actualizar.

- `User Lock`

  El flujo espera un `GET_LOCK()`.

- `Waiting for tables`

  El flujo obtuvo una notificación que la estructura de la tabla cambió y necesita reabrirla. Sin embargo, para ello, debe esperar a que el resto de flujos cierren la tabla en cuestión.

  Esta notificación tiene lugar si otro flujo ha usado `FLUSH TABLES` o uno de los siguientes comandos en la tabla en cuestión: `FLUSH TABLES tbl_name`, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, o `OPTIMIZE TABLE`.

- `waiting for handler insert`

  El handler `INSERT DELAYED` ha procesado las inserciones pendientes y espera nuevas.

La mayoría de estados se corresponden a operaciones rápidas. Si un flujo está en alguno de ellos varios segundos, puede existir un problema que necesite investigar.

Hay algunos estdos que no se mencionan en la lista precedente, pero varios de ellos son útiles sólo para encontrar fallos en el servidor.

13.5.4.17. Sintaxis de `SHOW STATUS`

```
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
```

`SHOW STATUS` proporciona información de estado del servidor. Esta información puede obtenerse usando el comando `mysqladmin extended-status` .

Aquí se muestra una salida parcial. La lista de variables y sus vallires pueden ser distintos para su servidor. El significado de cada variable se da en Sección 5.3.4, “Variables de estado del servidor”.

```
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      |
+--------------------------+------------+
```

Con una cláusula LIKE , el comando muestra sólo las variables que coinciden con el patrón:

```
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  |
+--------------------+----------+
```

Las opciones GLOBAL y SESSION son nuevas en MySQL 5.0.2 Con GLOBAL, obtiene los valores de estado para todas las conexiones a MySQL. Con SESSION, obtiene los valores de estado para la conexión actual. Si no usa estas opciones, por defecto es SESSION. LOCAL es sinónimo de SESSION.

Tenga en cuenta que algunas variables de estado sólo tienen un valor global. Para ellas obtiene el mismo valor para GLOBAL y SESSION.

13.5.4.18. Sintaxis de SHOW TABLE STATUS

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

SHOW TABLE STATUS funciona como SHOW TABLE, pero proporciona mucha más información acerca de cada tabla. Puede obtener esta lista usando el comando mysqlshow --status *db_name* . Desde MySQL 5.0.1, este comando también muestra información sobre vistas.

SHOW TABLE STATUS returna los siguientes campos:

- Name

  Nombre de tabla.

- Engine

  Motor para la tabla. Antes de MySQL 4.1.2, este valor se llama Type. Consulte Capítulo 14, *Motores de almacenamiento de MySQL y tipos de tablas*.

- Version

  Número de versión del fichero .frm de la tabla.

- Row_format

  Formato de almacenamiento de registros (Fixed, Dynamic, Compressed, Redundant, Compact). Desde MySQL/InnoDB 5.0.3, el formato de tablas InnoDB se reporta como Redundant o Compact. Antes de 5.0.3, las tablas InnoDB siempre están en formato Redundant .

- Rows

  Número de registros. Algunos motores como MyISAM, guardan el número exacto.

Para otros motores, como InnoDB, este valor es una aproximación y puede variar del valor real hasta de un 40 a 50%. En tales casos, use SELECT COUNT(*) para obtener el número real de registros.

El valor Rows es NULL para tablas en la base de datos INFORMATION_SCHEMA .

- Avg_row_length

  Longitud de registro media.

- Data_length

  Tamaño del fichero de datos.

- Max_data_length

  Máxima longitud del fichero d edatos. Este es el número total de bytes de datos que pueden almacenarse en la tabla dado el tamaño de puntero de datos usado.

- Index_length

  Tamaño de fichero índice.

- Data_free

  Número de bytes reservados no usados.

- Auto_increment

  Siguiente valor AUTO_INCREMENT .

- Create_time

  Cuándo se creó la tabla .

- Update_time

  Cuándo se actualizó por última vez el fichero de datos.

- Check_time

  Cuándo se chequeó la tabla por última vez.

- Collation

  Conjunto de caracteres y colación de la tabla.

- Checksum

  Valor de checksum en vivo (si hay).

- Create_options

  Opciones extra usadas con CREATE TABLE.

- Comment

  Comentario usado al crear la tabla (o información de porqué MySQL no puede acceder a la información de tabla).

En el comentario de tabla, las tablas InnoDB reportan el espacio libre del espacio de tabla al que pertenece la tabla. Para una tabla localizada en el espacio compartido, este es el espacio libre del espacio de tabla compartido. Si usa múltiples espacios y la tabla tiene el suyo, el espacio libre es sólo para esa tabla.

Para tablas MEMORY (HEAP) los valores Data_length, Max_data_length, y Index_length aproximan la cantidad real de memoria reservada. El algoritmo de reserva reserva grandes cantidades de memoria para reducir el número de operaciones de reserva.

Para vistas, todos los campos mostrados por SHOW TABLE STATUS son NULL excepto que Name indicata el nombre de vista y Comment dice view.

13.5.4.19. Sintaxis de SHOW TABLES

```
SHOW [FULL] TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW TABLES lista las tablas no TEMPORARY en una base de datos dada. Puede obtener esta lista usando el comando mysqlshow *db_name* .

Antes de MySQL 5.0.1, la salida de SHOW TABLES contiene una única columna de nombres de tabla. Desde MySQL 5.0.1, este comando lista cualquier vista en la base de datos. Desde MySQL 5.0.2, se soporta el modificador FULL de forma que SHOW FULL TABLES muestra una segunda columna de salida. Los valores para la segunda columna son BASE TABLE para una tabla VIEW para una vista.

Nota: Si no tiene permisos para una tabla, la tabla no se muestra en la salida de SHOW TABLES o mysqlshow db_name.

13.5.4.20. Sintaxis de SHOW TRIGGERS

```
SHOW TRIGGERS [FROM db_name] [LIKE expr]
```

SHOW TRIGGERS lista los disparadores definidos en el MySQL server. Se implementó en MySQL 5.0.10.

Para el disparadores ins_sum como se define en Sección 20.3, “Utilización de disparadores”, la salida de este comando es la que se muestra:

```
mysql> SHOW TRIGGERS LIKE 'acc%';
+---------+--------+---------+-------------------------------+--------+---------+
| Trigger | Event  | Table   | Statement                     | Timing | Created |
+---------+--------+---------+-------------------------------+--------+---------+
| ins_sum | INSERT | account |  SET @sum = @sum + NEW.amount | BEFORE | NULL    |
+---------+--------+---------+-------------------------------+--------+---------+
```

Nota: Cuando use una cláusula LIKE con SHOW TRIGGERS, la expresión a cumplir (*expr*) se compara con el nombre de la tabla en que se declara el disparador, y no con el nombre del disparador:

```
mysql> SHOW TRIGGERS LIKE 'ins%';
Empty set (0.01 sec)
```

Aquí se muestra una breve explicación de las columnas de la salida de este comando:

- Trigger: Nombre del disparador.
- Event: Evento que invoca el disparador. Debe ser 'INSERT', 'UPDATE', o 'DELETE'.
- Table: La tabla para la que se define el disparador.

- Statement: Comando a ejecutar cuando se invoca el disparador. Es lo mismo que el texto mostrado en la columna ACTION_STATEMENT de INFORMATION_SCHEMA.TRIGGERS.
- Timing: Puede ser 'BEFORE' o 'AFTER'.
- Created: Actualmente el valor de esta columna siempre es NULL.

Debe tener el permiso SUPER para ejecutar SHOW TRIGGERS.

Consulte Sección 22.1.16, “La tabla INFORMATION_SCHEMA TRIGGERS”.

13.5.4.21. Sintaxis de SHOW VARIABLES

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

SHOW VARIABLES muestra los valores de algunas variables de sistema de MySQL. Esta información puede obtenerse usando el comando mysqladmin variables .

Con la opción GLOBAL, obtiene los valores que se usan para nuevas conexiones de MySQL. Con SESSION, obtiene los valores que hay en efecto para la conexión actual. Si no usa estas opciones, por defecto es SESSION.

LOCAL es sinónimo de SESSION.

Si los valores por defecto no son adecuados, puede cambiar la mayoría de variables usando opciones de línea de comandos cuando mysqld arranca o en tiempo de ejecución con el comando SET . Consulte Sección 5.3.1, “Opciones del comando mysqld” y Sección 13.5.3, “Sintaxis de SET”.

La salida parcial se muestra aquí. La lista de variables y sus valores pueden ser distintos para su servidor. El significado de cada variable se da en Sección 5.3.3, “Variables de sistema del servidor”. Información acerca de cómo adecuarlos se proporciona en Sección 7.5.2, “Afinar parámetros del servidor”.

```
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                         | 5.0.7-beta-Max                                          |
| version_bdb                     | Sleepycat Software: Berkeley DB 4.1.24: (June 11, 2005) |
| version_comment                 | MySQL Community Edition - Max (GPL)                     |
| version_compile_machine         | i686                                                    |
| version_compile_os              | pc-linux-gnu                                            |
| wait_timeout                    | 28800                                                   |
+---------------------------------+---------------------------------------------------------+
```

Con una cláusula `LIKE` , el comando muestra sólo las variables que coinciden con el patrón:

```
mysql> SHOW VARIABLES LIKE 'have%';
+----------------------+----------+
| Variable_name        | Value    |
+----------------------+----------+
| have_archive         | NO       |
| have_bdb             | YES      |
| have_blackhole_engine | YES      |
| have_compress        | YES      |
| have_crypt           | YES      |
| have_csv             | YES      |
| have_example_engine  | YES      |
| have_federated_engine | YES      |
| have_geometry        | YES      |
| have_innodb          | YES      |
| have_isam            | NO       |
| have_ndbcluster      | DISABLED |
| have_openssl         | NO       |
| have_query_cache     | YES      |
| have_raid            | NO       |
| have_rtree_keys      | YES      |
| have_symlink         | YES      |
+----------------------+----------+
```

13.5.4.22. Sintaxis de `SHOW WARNINGS`

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS` muestra los mensajes de error, advertencia y notas retornadas por el último comando que haya generado algún mensaje, o nada si el último mensaje que haya usado una tabla no haya generado mensajes. Un comando relacionado, `SHOW ERRORS`, sólo muestra los errores. Consulte Sección 13.5.4.9, “Sintaxis de SHOW ERRORS”.

La lista de mensajes se resetea para cada nuevo comando que use una tabla.

El comando `SHOW COUNT(*) WARNINGS` muestra el número total de errores, advertencias y notas. Puede recibir este número de la variable `warning_count` :

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

El valor de `warning_count` puede ser mayor que el número de mensajes mostrados por `SHOW WARNINGS` si la variable de sistema `max_error_count` tiene un valor tan pequeño que no se almacenen todos los mensajes. Se muestra posteriormente un ejemplo de cómo puede pasar esto.

La cláusula `LIMIT` tiene la misma sintaxis que para el comando `SELECT` . Consulte Sección 13.2.7, “Sintaxis de SELECT”.

El servidor MySQL devuelve el número total de errores, advertencias, y notas que hayan resultado desde el último comando. Si usa la API de C, este valor puede obtenerse llamando a `mysql_warning_count()`. Consulte Sección 24.2.3.63, “mysql_warning_count()”.

Las advertencias se generan para comandos tales como `LOAD DATA INFILE` y comandos DML tales como `INSERT`, `UPDATE`, `CREATE TABLE`, y `ALTER TABLE`.

El siguiente comando `DROP TABLE` provoca una nota:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
+-------+------+-------------------------------+
| Level | Code | Message                       |
+-------+------+-------------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-------------------------------+
```

Aquí hay un ejemplo simple que muesta una advertencia de sintaxis para `CREATE TABLE` y advertencias de conversiones para `INSERT`:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
         'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),(NULL,'test'),
    -> (300,'Open Source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 4

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
*************************** 4. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

El máximo número de errores, advertencias y notas a almacenar se controla mediante la variable de sistema `max_error_count`. Por defecto, su valor es 64. Para cambiar el número de mensajes que quiere almacenar, cambie el valor de `max_error_count`. En el siguiente ejemplo, el comando `ALTER TABLE` produce tres mensajes de advertencia, pero sólo se almacena uno porque `max_error_count` vale 1:

```
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;
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> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)
```

Para deshabilitar las advertencias, ponga `max_error_count` a 0. En este caso, `warning_count` indica cuántas advertencias ocurren, pero ninguno de los mensajes se almacenan.

Desde MySQL 5.0.3, puede poner la variable de sesión `SQL_NOTES` a 0 para que las advertencias del nivel `Note` no se guarden.

13.5.5. Otras sentencias para la administración

13.5.5.1. Sintaxis de `CACHE INDEX`

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

El comando `CACHE INDEX` asigna índices de tabla a una caché de clave específica. Se usa sólo para tablas `MyISAM` .

El siguiente comando asigna índices de las tablas `t1`, `t2`, y `t3` a la caché de claves llamada `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       |
+---------+--------------------+----------+----------+
```

La sintaxis de `CACHE INDEX` le permite especifcar que sólo deben asignarse índicess particulares de una tabla a la caché. Sin embargo, la implementación actual asigna todos los índices de la tabla a la caché, así que no hay razón para especificar nada más que el nombre de tabla.

La caché de clave referenciada en un comando `CACHE INDEX` puede crearse cambiando su tamaño con un comando que asigne un valor al parámetro o en la configuración del parámetro del servidor. Por ejemplo:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Puede accederse a los parámetros de la caché de claves como miembros de una variable de sistema estructurada. Consulte Sección 9.4.1, “Variables estructuradas de sistema”.

Una caché de claves debe existir antes de asignarle índices:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

Por defecto, los índices de tabla se asignan a la caché de claves principal (por defecto) creada en el arranque del servidor. Cuando se destruye una caché de índices, todos los índices asignados a la misma se asignan de nuevo a la caché por defecto.

Las asignaciones de índices afectan al servidor globalmente: Si un cliente asigna un índice a una caché dada, esta caché se usa para todas las consultas que tengan que ver con el índice, sin importar qué cliente realiza las consultas.

13.5.5.2. Sintaxis de `FLUSH`

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG] flush_option [, flush_option] ...
```

Debe usar el comando `FLUSH` si quiere limpiar algunas de las cachés internas que usa MySQL . Para ejecutar `FLUSH`, debe tener el permiso `RELOAD` .

`flush_option` puede ser cualquiera de los siguientes valores:

- `HOSTS`

  Vacía las tablas de la caché de equipos. Debe volcar las tablas de equipos si algunos de sus equipos cambia el número IP o si obtiene el mensaje de error `Host ... is blocked`. Cuando ocurren sucesivamente más de `max_connect_errors` errores para un equipo dado mientras conecta con el servidor MySQL , MySQL asume que hay algo incorrecto y bloquea el equipo de más peticiones de conexión. Volcar las tablas de equipos le permite al equipo intentar conectar de nuevo. Consulte Sección A.2.5, “`La máquina 'host_name' está bloqueada`”. Puede arrancar `mysqld` con `--max_connect_errors=999999999` para evitar este mensaje de error.

- `DES_KEY_FILE`

  Recarga las claves DES del fichero que se especifica con la opción `--des-key-file` en tiempo de arranque del servidor.

- `LOGS`

  Cierra y reabre todos los ficheros de log. Si ha especificado un fichero de log de actualizaciones o un fichero de log binario sin una extensión, el número de extensión del fichero de log se incrementa en uno respecto al fichero anterior. Si ha usado una extensión del nombre de fichero, MySQL cierra y reabre el fichero de log. En Unix, esto es lo mismo que enviar una señal `SIGHUP` al servidor `mysqld` (excepto en algunas versiones Mac OS X 10.3 donde `mysqld` ignora `SIGHUP` y `SIGQUIT`).

- `PRIVILEGES`

Recarga los permisos de las tablas de permisos en la base de datos `mysql` .

- `QUERY CACHE`

  Defragmenta caché de consulta para utilizar mejor su memoria. Este comando no borra ninguna consulta de la caché, no como `RESET QUERY CACHE`.

- `STATUS`

  Resetea la mayoría de variables de estado a cero. Esto es algo que debe usar sólo al debugar una consulta. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

- `{TABLE | TABLES} [tbl_name [, tbl_name] ...]`

  Cuando no se nombran tablas, cierra todas las tablas abiertas y fuerza a todas las tablas en uso a que se cierren. Esto también vuelca la caché de consultas. Con uno o más nombres de tabla, vuelca sólo las tablas dadas. `FLUSH TABLES` también borra todos los resultados de consultas de la caché de consultas, como el comando `RESET QUERY CACHE` .

- `TABLES WITH READ LOCK`

  Cierra todas las tablas abiertas y bloquea todas las tablas para todas las bases de datos con una bloqueo de lectura hasta que ejecute `UNLOCK TABLES`. Esto es una forma muy conveniente de obtener copias de seguridad si tiene un sistema de ficheros como Veritas que puede tomas muestras en puntos de tiempo concretos.

- `USER_RESOURCES`

  Resetea todos los recursos por hora de usuario a cero. Esto le permite a los clientes que hayan alcanzado el límite de su conexión de hora, de consulta o de actualización para reanudar las actividades inmediatamente. `FLUSH USER_RESOURCES` no se aplica al límite en conexiones máximas simultáneas. Consulte Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”.

En MySQL 5.0, los comandos `FLUSH` se escriben en el lob binario a no ser que la plabra `NO_WRITE_TO_BINLOG` (o su alias `LOCAL`) se use. **Nota**: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, y `FLUSH TABLES WITH READ LOCK` no se loguean en ningún caso porque causarían problemas si se replicasen en un esclavo.

Puede acceder a algunos de estos comandos con la utilidad `mysqladmin` usando los comandos `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, o `flush-tables` .

Consulte Sección 13.5.5.5, “Sintaxis de RESET” para información acerca cómo se usa el comando `RESET` para replicación.

13.5.5.3. Sintaxis de `KILL`

```
KILL [CONNECTION | QUERY] thread_id
```

Cada conexión a `mysqld` se ejecuta en un flujo separado. Puede ver los flujos en ejecución con el comando `SHOW PROCESSLIST` y matar un flujo con el comando `KILL thread_id` .

En MySQL 5.0.0, `KILL` permite los modificadores opcionales `CONNECTION` o `QUERY`:

- `KILL CONNECTION` es lo mismo que `KILL` sin modificadores: termina la conexión asociada con el *`thread_id`* dado.

- `KILL QUERY` termina el comando que la conexión está ejecutando actualmente, pero deja a la conexión intacta.

Si tiene el permiso `PROCESS` , puede ver todos los flujos, puede matar todos los flujos y comandos. De otro modo, puede ver y matar sólo sus propios flujos y comandos.

Puede usar los comandos `mysqladmin processlist` y `mysqladmin kill` para examinar y matar flujos.

Nota: No puede usar `KILL` con la biblioteca Embedded MySQL Server porque el servidor empotrado se ejecuta dentro del flujo de la aplicación que lo aloja. No crea ningún flujo de conexión por sí solo.

Cuando hace un `KILL`, se activa un flag específico para el flujo. En la mayoría de casos, puede que el flujo tarde algo de tiempo en morir, porque el flag kill se chequea sólo cada ciertos intervalos:

- En `SELECT`, `ORDER BY` y `GROUP BY` , el flag se chequea tras leer un bloque de registros. Si el flag kill está activado, el comando se aborta.
- Durante `ALTER TABLE`, el flag kill se chequea antes de que se lea cada bloque de registros de la tabla original. Si el flag kill está activado, el comando se aborta y la tabla temporal se borra.
- Durante operaciones `UPDATE` o `DELETE` , el flag kill se chequea tras cada lectura de bloque y tras cada registro borrado o actualizado. Si el flag kill está activado, el comando se aborta. Tenga en cuenta que si no está usando transacciones, los cambios no se deshacen.
- `GET_LOCK()` aborta y retorna `NULL`.
- Un flujo `INSERT DELAYED` rápidamente vuelca (inserta) todos los registros que tiene en memoria y luego termina.
- Si el flujo está en el handler de bloqueo (estado: `Locked`), el bloqueo de tabla se aborta rápidamente.
- Si el flujo está esperando a espacio libre en disco en una llamada de lectura, la escritura se aborta con un mensaje de error "disco lleno".
- **Advertencia**: Matar una operación `REPAIR TABLE` o `OPTIMIZE TABLE` en una tabla `MyISAM` resulta en una tabla que corrupta y no usable. Cualquier lectura o escritura en una tabla así falla hasta que la optimiza o repara de nuevo (sin interrupción).

13.5.5.4. Sintaxis de `LOAD INDEX INTO CACHE`

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

El comando `LOAD INDEX INTO CACHE` en MySQL 5.0 precarga un índice de tabla en la caché de clave para la que se ha asignado por un comando `CACHE INDEX` , o en la caché de claves por defecto en otro caso. `LOAD INDEX INTO CACHE` se usa sólo para tablas `MyISAM` .

El modificador `IGNORE LEAVES` hace que se carguen sólo los bloques para los nodos que no sean hojas del índice.

El siguiente comando precarga nodos (bloques índice) de índices para las tablas `t1` y `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       |
+---------+--------------+----------+----------+
```

Este comando precarga todos los bloques índice de `t1`. Sólo precarga bloques para los nodos que no sean hojas de `t2`.

La sintaxis de `LOAD INDEX INTO CACHE` le permite especificar que sólo índices particulares de una tabla deben precargarse. Sin embargo, la implementación actual precarga todos los índices de tabla en la caché, así que no hay razón para especificar nada distinto al nombre de tabla.

13.5.5.5. Sintaxis de RESET

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

El comando `RESET` se usa para limpiar el estado de varias operaciones de servidor. Actúa como una versión más fuerte del comando `FLUSH` . Consulte Sección 13.5.5.2, “Sintaxis de FLUSH”.

Debe tener el permiso `RELOAD` para ejecutar `RESET`,

`reset_option` puede ser cualquiera de las siguientes:

- `MASTER`

  Borra todos los logs binarios listados en el fichero índice, resetea el fichero de índice de log binario para vaciarlo, y crea un nuevo fichero de log binario. (Conocido como `FLUSH MASTER` en versiones prévias de MySQL.) Consulte Sección 13.6.1, “Sentencias SQL para el control de servidores maestros”.

- `QUERY CACHE`

  Borra todos los resultados de consulta de la caché de consulta.

- `SLAVE`

  Hace que el esclavo olvide su posición de replicación en los logs binarios del maestro. También resetea el log retardado borrando cualquier fichero de log retardado y comenzando uno nuevo. Conocido como `FLUSH SLAVE` en versiones prévias MySQL.) Consulte Sección 13.6.2, “Sentencias SQL para el control de servidores esclavos”.

13.6. Sentencias de replicación

Esta sección describe comandos SQL relacionados con replicación. Un grupo de comandos se usa para controlar los servidores maestros. El otro se usa para controlar servidores esclavos.

13.6.1. Sentencias SQL para el control de servidores maestros

La replicación puede controlarse mediante la interfaz SQL. Esta sección discute los comandos para administrar los maestros de replicación. Sección 13.6.2, “Sentencias SQL para el control de servidores esclavos” discute comandos para administrar servidores esclavos.

13.6.1.1. Sintaxis de PURGE MASTER LOGS

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'
```

```
PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

Borra todos los logs binarios listados en el índice de log prévio al log especificado o fecha. Los logs se borran de la lista guardada en el fichero de índice de log, así que el log dado pasa a ser el primero.

Ejemplo:

```
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

El argumento de `BEFORE date` puede estar en formato `'YYYY-MM-DD hh:mm:ss'` . `MASTER` y `BINARY` son sinónimos en MySQL 5.0.

Si tiene un esclavo activo que actualmente esté leyendo uno de los logs que está intentando borrar, este comando no hace nada y falla con un error. Sin embargo, si un esclavo está dormido y purga los logs que quiere leer, el esclavo no es capaz de replicar cuando se despierta. El comando puede ejecutarse mientras los esclavos se replican. No necesita pararlos.

Para purgar logs, siga este procedimiento:

1. En cada servidor esclavo, use `SHOW SLAVE STATUS` para chequear qué log está leyendo.
2. Obtinga una lista de los logs en el servidor maestro con `SHOW MASTER LOGS`.
3. Determine el primer log entre todos los esclavos. Este es el log objetivo. Si todos los esclavos están actualizados, este es el último log de la lista.
4. Haga una copia de seguridad de todos los logs que vaya a borrar. (Este paso es opcional, pero siempre recomendable.)
5. Purgue todos los logs hasta el log objetivo, pero no lo incluya.

13.6.1.2. Sintaxis de `RESET MASTER`

```
RESET MASTER
```

Borra todos los logs binarios listados en el fichero índice, resetea el fichero índice de lob binario a vaciar, y recrea un nuevo fichero de log binario.

13.6.1.3. Sintaxis de `SET SQL_LOG_BIN`

```
SET SQL_LOG_BIN = {0|1}
```

Activa o desactiva el logueo binario para la conexión actual (`SQL_LOG_BIN` es una variable de sesión) si el cliente conecta usando una cuenta que tenga el permiso `SUPER` . En MySQL 5.0, el comando se rechaza con un error si el cliente no tiene este permiso.

13.6.1.4. Sintaxis de `SHOW BINLOG EVENTS`

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Muestra los eventos en el log binario. Si no especifica `'log_name'`, se muestra el primer log binario.

La cláusula `LIMIT` tiene la misma sintaxis que para el comando `SELECT` . Consulte Sección 13.2.7, “Sintaxis de SELECT”.

Nota: Realizar `SHOW BINLOG EVENTS` sin cláusula `LIMIT` puede iniciar un proceso muy largo y que consume muchos recursos mientras el servidor vuelca los contenidos completos del log binario (lo que incluye la mayoría de las consultas ejecutadas por MySQL) a stdout. Para guardar el log binario en un fichero de texto para analizar posteriormente, use la utilidad `mysqlbinlog` . Consulte Sección 8.5, “La utilidad `mysqlbinlog` para registros binarios”.

13.6.1.5. Sintaxis de `SHOW MASTER LOGS`

```
SHOW MASTER LOGS
SHOW BINARY LOGS
```

Lista los ficheros del log binario en el servidor. Este comando se usa como parte del procedimiento descrito en Sección 13.6.1.1, “Sintaxis de `PURGE MASTER LOGS`” para determinar qué logs pueden purgarse.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

En MySQL 5.0, `SHOW BINARY LOGS`, es equivalente a `SHOW MASTER LOGS`. La columna `File_size` se muestra desde MySQL 5.0.7.

13.6.1.6. Sintaxis de `SHOW MASTER STATUS`

```
SHOW MASTER STATUS
```

Proporciona información de estado en los ficheros del log binario del maestro de replicación.

13.6.1.7. Sintaxis de `SHOW SLAVE HOSTS`

```
SHOW SLAVE HOSTS
```

Muestra una lista de esclavos de replicación registrados actualmente en el maestro. Cualquier esclavo no arrancado con la opción `--report-host=`*`slave_name`* no es visible en esta lista.

13.6.2. Sentencias SQL para el control de servidores esclavos

La replicación puede controlarse con la interfaz SQL. Esta sección discute comandos para administrar servidores de replicación esclavos. Sección 13.6.1, “Sentencias SQL para el control de servidores maestros” discute comandos para administrar servidores maestros.

13.6.2.1. Sintaxis de `CHANGE MASTER TO`

```
  CHANGE MASTER TO master_def [, master_def] ...

master_def:
      MASTER_HOST = 'host_name'
    | MASTER_USER = 'user_name'
    | MASTER_PASSWORD = 'password'
    | MASTER_PORT = port_num
    | MASTER_CONNECT_RETRY = count
    | 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'
```

Cambia los parámetros que usa el servidor esclavo para conectar y comunicar con el servidor maestro.

`MASTER_USER`, `MASTER_PASSWORD`, `MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, y `MASTER_SSL_CIPHER` proporciona información para el esclavo acerca de cómo conectar con su maestro.

Las opciones SSL (`MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, y `MASTER_SSL_CIPHER`) pueden cambiarse incluso en esclavos que se compilan sin soporte SSL. Se guardan en el fichero `master.info`, pero se ignorarn hasta que use un servidor que tenga soporte SSL activado.

Si no especifica un parámetro dado, mantiene su valor antiguo, excepto cuando se indica en la siguiente discusión. Por ejemplo, si la contraseña para conectar a su maestro MySQL ha cambiado, necesita ejecutar este comando para decir al esclavo la nueva contraseña:

```
mysql> STOP SLAVE; -- if replication was running
mysql> CHANGE MASTER TO MASTER_PASSWORD='new3cret';
mysql> START SLAVE; -- if you want to restart replication
```

Aquí no hay necesidad de especificar los parámetros que no cambian (equipo, puerto, usuario, y así).

`MASTER_HOST` y `MASTER_PORT` son el nombre de equipo (o dirección IP) del equipo maestro y su puerto TCP/IP. Tenga en cuenta que si `MASTER_HOST` es igual a `localhost`, entonces, como en otras partes de MySQL, el puerto puede ignorarse (si los ficheros socket Unix pueden usarse, por ejemplo).

Si especifica `MASTER_HOST` o `MASTER_PORT`, el esclavo asume que el servidor maestro es distinto que antes (incluso si especifica un valor de equipo o de puerto igual que el anterior.) En este caso, los antiguos valores para el log binario del servidor maestro y su posición no se consideran aplicables por más tiempo, así que si no especifica `MASTER_LOG_FILE` y `MASTER_LOG_POS` en el comando `MASTER_LOG_FILE=''` y `MASTER_LOG_POS=4` se añaden al final.

`MASTER_LOG_FILE` y `MASTER_LOG_POS` son las coordinadas en que flujo esclavo de entradea/salida debe empezar a leer del maestro la siguiente vez que el flujo arranque. Si especifica alguna de ellas, no puede especificar `RELAY_LOG_FILE` o `RELAY_LOG_POS`. Si no se especifica `MASTER_LOG_FILE` ni `MASTER_LOG_POS` , el esclavo usa las últimas coordenadas del *flujo SQL del esclavo* antes de realizar `CHANGE MASTER`. Esto asegura que no hay discontinuidad en la replicación, incluso si el flujo SQL esclavo se comparó posteriormente con el flujo esclavo de entrada/salida, cuando símplemente quiere cambiar, digamos, la contraseña a usar.

`CHANGE MASTER` *borra todos los ficheros de log retardados* y arranca uno nuevo, a no ser que especifique `RELAY_LOG_FILE` o `RELAY_LOG_POS`. En tal caso, los logs retardados se guardan; en MySQL 5.0, la variable global `relay_log_purge` se pone a 0.

`CHANGE MASTER TO` actualiza los contenidos de los ficheros `master.info` y `relay-log.info` .

`CHANGE MASTER` es útil para inicializar un esclavo cuando tiene una imagen del maestro y ha guardado el log y el desplazamiento correspondientes a la misma. Tras cargar la imagen en el esclavo, puede ejecutar `CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master` en el esclavo.

Ejemplos:

```
mysql> 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;

mysql> CHANGE MASTER TO
    ->     RELAY_LOG_FILE='slave-relay-bin.006',
    ->     RELAY_LOG_POS=4025;
```

El primer ejemplo cambia el maestro y las coordenadas del log binario del maestro. Esto se usa cuando quiere preparar el esclavo para replicar el maestro.

El segundo ejemplo muestra una operación que se emplea menos frecuentemente. Si se usa cuando el esclavo tiene logs retardados que quiere ejecutar de nuevo por alguna razón. Para ello, el maestro no necesita ser accesible. Sólo necesita usar `CHANGE MASTER TO` y arrancar el flujo SQL (`START SLAVE SQL_THREAD`).

Incluso puede usar la segunda operación en una configuración de no replicación con un servidor aislado, no esclavo, para recuperación de fallos posteriores. Suponga que su servidor ha fallado y ha restaurado una copia de seguridad. Quiere volver a ejecutar los logs binarios del servidor (no los logs retardados, sino los logs binarios regulares), llamados (por ejemplo) `myhost-bin.*`. En primer lugar, haga una copia de seguridad de los logs binarios en un sitio seguro, en caso que no siga exactamente el procedimiento posterior y accidentalmente haga que el servidor purgue los logs binarios. En MySQL 5.0, use `SET GLOBAL relay_log_purge=0` para seguridad adicional. Cuando arranca el servidor sin la opción `--log-bin` , En su lugar, use las opciones `--replicate-same-server-id`, `--relay-log=myhost-bin` (para que el servidor crea que los logs binarios regulares son los logs retardados), `--skip-slave-start` . Cuando el servidor arranca, ejecute estos comandos:

```
mysql> CHANGE MASTER TO
    ->     RELAY_LOG_FILE='myhost-bin.153',
    ->     RELAY_LOG_POS=410,
    ->     MASTER_HOST='some_dummy_string';
mysql> START SLAVE SQL_THREAD;
```

El servidor lee y ejecuta sus propios logs binarios, permitiendo recuperación de fallos. Una vez que finaliza la recuperación, ejecute `STOP SLAVE`, pare el servidor, borre `master.info` y `relay-log.info`, y reinicie el servidor con sus opciones originales.

Actualmente, especificar `MASTER_HOST` (incluso con un valor de prueba) se necesita para hacer que el servidor piense que es un esclavo. En el futuro, planeamos añadir opciones para evitar estas restricciones menores.

13.6.2.2. Sintaxis de `LOAD DATA FROM MASTER`

```
LOAD DATA FROM MASTER
```

Este comando toma una muesta del maestro y la copia en el esclavo. Actualiza los valores de `MASTER_LOG_FILE` y `MASTER_LOG_POS` para que el esclavo comience la replicación desde la posición correcta. Cualquier regla de exclusión de tabla y base de datos especificada con las opciones `--replicate-*-do-*` y `--replicate-*-ignore-*` se tienen en cuenta. `--replicate-rewrite-db` *no* se tienen en cuenta. Esto es porque un usuario puede, con esta opción, configurar un mapeo no

único tal como `--replicate-rewrite-db=db1->db3` y `--replicate-rewrite-db=db2->db3`, que puede confundir al esclavo al cargar tablas del maestro.

El uso de este comando está sujeto a las siguientes condiciones:

- Esto sólo funciona con tablas `MyISAM` . Intentos de cargar de una tabla no `MyISAM` provoca el siguiente error:

```
ERROR 1189 (08S01): Net error reading from master
```

- Adquiere un bloqueo de lectura global del maestro al tomar la muestra, que evita actualizaciones en el maestro durante la operación de carga.

En el futuro, planeamos hacer este comando compatible con tablas `InnoDB` y eliminar la necesidad de bloqueo de lectura global usando una copia de seguridad no bloqueante en línea.

Si está cargando tablas grandes, puede tener que incrementar los valores de `net_read_timeout` y `net_write_timeout` en los servidores esclavos y maestros. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

Tenga en cuenta que `LOAD DATA FROM MASTER` *no* copia ninguna tabla de la base de datos `mysql`. Esto hace fácil tener distintos usuarios y permisos en el maestro y el esclavo.

El comando `LOAD DATA FROM MASTER` necesita la cuenta de replicación que se usa para conectar con el maesto para tener los permisos `RELOAD` y `SUPER` en el maestro y el permiso `SELECT` para todas las tablas maestras que quiera cargar. Todas las tablas del maestro para las que el usuario no tiene el permiso `SELECT` se ignoran por `LOAD DATA FROM MASTER`. Esto es porque el maestro las oculta del usuario: `LOAD DATA FROM MASTER` llama `SHOW DATABASES` para conocer las bases de datos a cargar por parte del maestro, pero `SHOW DATABASES` retorna sólo bases de datos para las que el usuario tenga algunos permisos. Consulte Sección 13.5.4.6, “Sintaxis de `SHOW DATABASES`”. En la parte del esclavo, el usuario que ejecuta `LOAD DATA FROM MASTER` debe tener permisos para crear y borrar bases de datos y tablas que se copien.

13.6.2.3. Sintaxis de `LOAD TABLE nombre_de_tabla FROM MASTER`

```
LOAD TABLE tbl_name FROM MASTER
```

Transfiere una copia de la tabla desde el maestro al esclavo. Este comando se implementa principalmente para depurar `LOAD DATA FROM MASTER`. Requiere que la cuenta usada para conectar con el servidor maestro tenga los permisos `RELOAD` y `SUPER` en el maestro y el permiso `SELECT` en la tabla maestra a cargar. En la parte del esclavo, el usuario que ejecuta `LOAD TABLE FROM MASTER` debe tener permisos para borrar y crear la tabla.

Las condiciones para `LOAD DATA FROM MASTER` se aplican aquí. Por ejemplo, `LOAD TABLE FROM MASTER` funciona sólo en tablas `MyISAM`. También se aplican las notas sobre timeouts para `LOAD DATA FROM MASTER` .

13.6.2.4. Sintaxis de `MASTER_POS_WAIT()`

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

Esto es una función, no un comando. Se usa para asegurar que el esclavo ha leído y ejecutado eventos hasta la posición dada en el log binario del maestro. Consulte Sección 12.9.4, “Funciones varias” para una descripción completa.

13.6.2.5. Sintaxis de `RESET SLAVE`

```
RESET SLAVE
```

Hace que el esclavo olvide su posición de replicación en el log binario del maestro. Este comando se debe usar para un inicio limpio: borra los ficheros `master.info` y `relay-log.info` , todos los logs retardados, y arranca un nuevo log retardado.

Nota: Todos los logs retardados se borran, incluso si no se han ejecutado completamente por parte del flujo SQL esclavo . (Esta es una condición que es probable que exista en un esclavo de replicación si ha ejecutado un comando `STOP SLAVE` o si el esclavo está muy cargado.)

La información de conexión almacenada en el fichero `master.info` se resetea inmediatamente usando cualquier valor especificado en las opciones de arranque correspondientes. Esta información incluye valores tales como el equipo maestro, puerto, usuario y contraseña del maestro. Si el flujo SQL esclavo está en medio de una operación de replicar tablas temporales cuando se paró, y se ejecuta `RESET SLAVE` , estas tablas temporales replicadas se borran en el esclavo.

13.6.2.6. Sintaxis de `SET GLOBAL SQL_SLAVE_SKIP_COUNTER`

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n
```

Ignora los siguientes *n* eventos del maestro. Esto es útil para recuperarse de paradas de replicación provocadas por un comando.

Este comando es válido sólo cuando el flujo esclavo no está en ejecución. De otro modo, produce un error.

13.6.2.7. Sintaxis de `SHOW SLAVE STATUS`

```
SHOW SLAVE STATUS
```

Proporciona información de estado de parámetros esenciales de los flujos esclavos. Si ejecuta este comando usando el cliente `mysql` puede usar un terminador de comando `\G` en lugar de punto y coma para obtener una salida vertical más legible:

```
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
Relay_Master_Log_File: gbichot-bin.005
     Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
      Replicate_Do_DB:
  Replicate_Ignore_DB:
           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
```

`SHOW SLAVE STATUS` retorna los siguientes campos:

- `Slave_IO_State`

  Una copia del campo `State` se la salida de `SHOW PROCESSLIST` para el flujo esclavo de entrada/salida. Le dice si el flujo está tratando de conectar con el maestro, esperando eventos del maestro, reconectando con el maestro, etc. Los estados posibles se listan en Sección 6.3, “Detalles de la implementación de la replicación”. Consultar este campo es necesario porque, por ejemplo, el flujo puede estar en ejecución pero intentando conectar con el maestro sin éxito; sólo este campo le muestra el problema de conexión. El estado del flujo SQL no se copia porque es más simple. Si está en ejecución, no hay problema; si no es así, puede encontrar el error en el campo `Last_Error` (descrito posteriormente).

- `Master_Host`

  Equipo maestro actual

- `Master_User`

  Usuario actual usado para conectar con el maestro.

- `Master_Port`

  Puerto maestro actual.

- `Connect_Retry`

  Valor actual de la opción `--master-connect-retry` .

- `Master_Log_File`

  Nombre del fichero de log binario desde el que está leyendo actualmente el flujo de entrada/salida.

- `Read_Master_Log_Pos`

  La posición hasta la que el flujo de entrada/salida ha leído en el log binario del maestro.

- `Relay_Log_File`

  Nombre del fichero de log retardado desde el que el flujo SQL está leyendo y ejecutando actualmente.

- `Relay_Log_Pos`

  La posición hasta la que el flujo SQL ha leído y ejecutado en el flujo en el log retardado actual.

- `Relay_Master_Log_File`

  Nombre del log binario maestro que contiene la mayoría de los eventos recientes ejecutados por el flujo SQL.

- `Slave_IO_Running`

Si el flujo de entrada/salida está activo.

- `Slave_SQL_Running`

  Si el flujo SQL está activo.

- `Replicate_Do_DB, Replicate_Ignore_DB`

  La lista de bases de datos especificadas con las opciones `--replicate-do-db` y `--replicate-ignore-db` , si se dió alguna.

- `Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table`

  Lista de tablas especificadas con las opciones `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, y `--replicate-wild-ignore_table` , si se dió alguna.

- `Last_Errno, Last_Error`

  Número y mensaje de error retornados por la última consulta ejecutada. Un número de error 0 y mensaje vacío significa “no error.” Si el valor `Last_Error` no está vacío, también aparece como mensaje en el log de errores del esclavo.

  Por ejemplo:

  ```
  Last_Errno: 1051
  Last_Error: error 'Unknown table 'z'' on query 'drop table z'
  ```

  El mensaje indica que la tabla `z` existió en el maestro y se borró allí, pero no existió en el esclavo, así que `DROP TABLE` falló en el esclavo. (Esto puede ocurrir, por ejemplo, si olvidó copiar la tabla en el esclavo al configurar la replicación.)

- `Skip_Counter`

  El último valor usado para `SQL_SLAVE_SKIP_COUNTER`.

- `Exec_Master_Log_Pos`

  La posición del último evento ejecutado por el flujo SQL del log binario del maestro (`Relay_Master_Log_File`). (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) en el log binario del maestro correspondiente a (`Relay_Log_File`, `Relay_Log_Pos`) en el log retardado.

- `Relay_Log_Space`

  Tamaño total combinado de todos los logs retardados existentes.

- `Until_Condition, Until_Log_File, Until_Log_Pos`

  Valores especificados en la cláusula `UNTIL` del comando `START SLAVE` .

  `Until_Condition` tiene estos valores:

  - `None` si no se especificó `UNTIL`
  - `Master` si el esclavo está leyendo hasta una posición dada en el log binario del maestro
  - `Relay` si el esclavo está leyendo hasta una posición dada en su log retardado

`Until_Log_File` y `Until_Log_Pos` indica los valors del nombre de fichero y posición que definen el punto en que el flujo SQL para su ejecución.

- `Master_SSL_Allowed, Master_SSL_CA_File, Master_SSL_CA_Path, Master_SSL_Cert, Master_SSL_Cipher, Master_SSL_Key`

  Estos campos muestran los parámetros SSL usados por el esclavo para conectar con el maestro, si hay algo.

  `Master_SSL_Allowed` tiene estos valores:

  - `Yes` si se permite conexión SSL con el maestro
  - `No` si no se permite una conexión SSL con el maestro
  - `Ignored` se se permite una conexión SSL pero el servidor esclavo no tiene soporte SSL activdo

  Los valores de los otros campos relacionados con SSL se corresponden con los valores de las opciones `--master-ca`, `--master-capath`, `--master-cert`, `--master-cipher`, y `--master-key` .

- `Seconds_Behind_Master`

  Este campo indica el “retardo” del esclavo. Cuando el flujo SQL esclavo está en ejecución (procesando actualizaciones), este campo es el número de segundos que han pasado desde el momento del evento más reciente del maestro ejecutado por este flujo. Cuando ese flujo lo atrapa el flujo de entrada/salida esclavo y pasa a espera de más eventos del flujo de entrada/salida este campo es cero. En resumen, este campo mide en segundos la diferencia temporal entre el flujo SQL esclavo y el flujo de entrada/salida esclavo.

  Si la conexión de red entre maestro y esclavo es rápida, el flujo de entrada/salida esclavo es muy cercano al maestro, así que este campo es una buena aproximación de cuanto tarda el flujo SQL esclavo en compararse con el maestro. Si la red es lenta, esta **no** es una buena aproximación, el flujo SQL esclavo puede verse atrapado a menudo por un flujo de entrada/salida esclavo lento , así que `Seconds_Behind_Master` a menudo muestra un valor de 0, incluso si el flujo de entrada/salida se compara posteriormente con el maestro. En otras palabras, **esta columna es útil sólo para redes rápidas**.

  Esta computación de diferencia temporal funciona incluso si el esclavo y maestro no tienen relojes idénticos (la diferencia de tiempo se computa cuando el flujo de entrada/ salida del esclavo arranca, y se asume como que permance constante desde ese momento). `Seconds_Behind_Master` es `NULL` (que significa “desconocido”) si el flujo SQL esclavo no está en ejecución, o si el flujo de entrada/salida esclavo no está en ejecución o no conectado con el maestro. Por ejemplo si el flujo de entrada/salida esclavo está durmiendo durante `master-connect-retry` segundos antes de reconectar, `NULL` se muestra, ya que el esclavo no puede saber lo que hace el maestro, y así no puede asegurar el retardo.

  Este campo tiene una limitación. El timestamp se preserva a través de la replicación, lo que significa que, si un maestr M1 es un esclavo de M0, cualquier evento desde el log binario de M1 que se origine al replicar un evento del log binario M0 tiene el timestamp del evento. Esto permite a MySQL replicar `TIMESTAMP` con éxito. Sin embargo, la desventaja para `Seconds_Behind_Master` es que si M1 también recibe actualizaciones directas de los clientes, el valor se desvía aleatoriamente, ya que a veces el último evento de M1 es de M0 y otras de actualización directa, y por lo tanto es el timestampo más reciente.

13.6.2.8. Sintaxis de `START SLAVE`

```
START SLAVE [thread_type [, thread_type] ... ]
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_type: IO_THREAD | SQL_THREAD
```

`START SLAVE` sin opciones arranca los flujos esclavos. El flujo de entrada/salida lee consultas del servidor maestro y las almacena en el log retardado. El flujo SQL lee el log retardado y ejecuta las consultas. `START SLAVE` require el permiso `SUPER`.

Si `START SLAVE` tiene éxito al arrancar los flujos esclavos, retorna sin ningún error. Sin emgargo, incluso en tal caso, puede ser que el flujo esclavo arranqu y luego pare (por ejemplo, porque no puede conectar con el maestro o leer sus logs binarios, o algún otro problema). `START SLAVE` no le advierte acerca de esto. Debe chequear el log de errores esclavo para mensajes de error generados por los flujos esclavos, o chequear que estén ejecutándos correctamente con `SHOW SLAVE STATUS`.

En MySQL 5.0, puede añadir las opciones `IO_THREAD` y `SQL_THREAD` al comando para decir qué flujo arrancar.

Una cláusula `UNTIL` puede añadirse para especificar que el esclavo debe arrancar y ejecutar hasta que el flujo SQL llegue a un punto dado en los logs binarios del maestro o en los logs retardados del esclavo. Cuando el flujo SQL llega a ese punto, para. Si la opción `SQL_THREAD` se especifica en el comando, arranca sólo el flujo SQL. De otro modo, arranca ambos flujos esclavos. Si el flujo SQL está en ejecución, la cláusula `UNTIL` se ignora y se muestra una advertencia.

Para una cláusula `UNTIL` , debe especificar nombre de fichero de log y posición. No mezcle opciones de maestro y log retardado.

Cualquier condición `UNTIL` se resetea mediante un comando `STOP SLAVE` , un comando `START SLAVE` que no incluya cláusula `UNTIL` , o reinicio de servidor.

La cláusula `UNTIL` puede ser útil para depurar replicación, o para que la replicación proceda justo antes del punto en que quiera evitar tener un esclavo replicando un comando. Por ejemplo, si se ejecuta un comando `DROP TABLE` no deseado, puede usar `UNTIL` para decir al esclavo que ejecute hasta ese punto pero no más. Para encontrar el evento, use `mysqlbinlog` con los logs maestros o los logs retardados del esclavo, o usando un comando `SHOW BINLOG EVENTS` .

Si está usando `UNTIL` para tener las consultas replicadas por el proceso esclavo en secciones, se recomienda que arranque el esclavo con la opción `--skip-slave-start` para evitar que el flujo SQL se ejecute cuando arranque el servidor. Es probablemente mejor usar esta opción con un fichero de opciones en lugar que en la línea de comandos, así que un reinicio inesperado del servidor no hace que se olvide.

El comando `SHOW SLAVE STATUS` incluye campos de salida que muestran los valores actuales de la condición `UNTIL` .

En versiones prévias de MySQL, este comando se llamó `SLAVE START` cuyo uso todavía se acepta en MySQL 5.0 por compatibilidad con versiones anteriores, pero ahora está obsoleto.

13.6.2.9. Sintaxis de `STOP SLAVE`

```
STOP SLAVE [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Para el flujo esclavo. `STOP SLAVE` necesita el permiso `SUPER` .

Como `START SLAVE`, este comando puede usarse con las opciones `IO_THREAD` y `SQL_THREAD` para nombrar el flujo o flujos a parar.

En versiones prévias de MySQL, este comando se llamó `SLAVE STOP`. Su uso se acepta en MySQL 5.0 por compatibilidad con versiones anteriores, pero ahora está obsoleto.

13.7. Sintaxis SQL de sentencias preparadas

MySQL 5.0 proporciona soporte para comandos preparados en la parte del servidor. Este soporte aprovecha del protocolo binario cliente-servidor implementado en MySQL 4.1, dado que use una interfaz de programación cliente apropiada. Las interfaces candidatas incluyen la biblioteca de la API de C de MySQL y MySQL Connector/NET. Por ejemplo, la API C proporciona un conjunto de llamadas de funciones que prepararn su API de comandos preparados. Consulte Sección 24.2.4, “Sentencias preparadas de la API C”. Otras interfaces de lenguajes pueden proporcionar soporte para comandos preparados que usen el protocolo binario enlazando la biblioteca cliente C, un ejemplo es la extensión mysqli extension in PHP 5.0.

Una interfaz SQL alternativa para comandos preparados está disponible Su intefaz no es tan eficiente como usar el protocolo binario mediante una API de comandos preparados, pero no necesita programación porque está disponible directamente a nivel SQL:

- Puede usarlo cuando no haya interfaz programable diponible
- Puede usarlo desde cualquier programa que le permita enviar comandos SQL al servidor, tales como el programa cliente `mysql` .
- Puede usarlo incluso si el cliente está usando una antigua versión de la biblioteca cliente. El único requerimiento es que sea capaz de conectar a un servidor lo suficientemente reciente para soporta sintaxis SQL para comandos preparados.

La sintaxis SQL para comandos preparados está pensada para usar en situaciones como la siguiente:

- Puede querer testear cómo fucionan los comandos preparados en su aplicación antes de codificar la aplicación. O quizás una aplicación tenga un problema ejecutando comandos preparados y quiera determinar interactivamente cuál es el problema.
- Quiere crear un test de uso que describa un problema que está teniendo con comandos preparados, así que puede perparar un reporte de error.
- Necesita usar comandos preparados pero no tiene acceso a la API de programación que los soporta.

La sintaxis SQL para comandos preparados se abasa en tres comandos SQL:

```
PREPARE stmt_name FROM preparable_stmt;

EXECUTE stmt_name [USING @var_name [, @var_name] ...];

{DEALLOCATE | DROP} PREPARE stmt_name;
```

El comando PREPARE prepara un comando y le asigna un nombre, *`stmt_name`*, con el que referirse al comando posteriormente. Los nombres de comando no son sensibles a mayúsculas. *`preparable_stmt`* es una cadena literal o una variable de usuario que contenga el texto del comando. El texto debe representar un comando SQL único, no varios. Dentro del comando, pueden usarse caracteres '?' como marcadores de parámetros para indicar dónde estarán los valores en la consulta posterior cuando la ejecute. Los caracteres '?' no deben delimitarse con comillas, incluso si pretende ligarlos con valores de cadenas.

Si un comando preparado con ese nombre existe, se elimina implícitamente antes que se prepare el nuevo comando. Esto significa que si el nuevo comando contiene un error y no puede preparase, se retorna un error y no existe un comando con el nombre dado.

El alcance de los comandos preparados es la sesión de cliente dentro de la que se crea. Otros clintes no pueden verlos.

Tras preparar un comando, lo ejecuta con un comando `EXECUTE` que se refiere al nombre de comando preparado. Si el comando preparado contiene cualquier marcador de parámetro, debe añadir una cláusula `USING` que liste las variables de usuario conteniendo los valores a ligar con los parámetros. Los valores de parámetros pueden proporcionarse sólo por variables de usuario, y la cláusula `USING` debe nombrar exactamente tantas variables como el número de marcadores de parámetros en el comando.

Puede ejecutar un comando preparado dado varias veces, pasando distintas variables al mismo o configurando las variables con distintos valores para cada ejecución.

Para eliminar un comando preparado, use el comando `DEALLOCATE PREPARE` . Tratar de ejecutar un comando preparado tras borrarlo provoca un error.

Si termina una sesión de cliente sin borrar un comando preparado préviamente, el servidor lo borra automáticamente.

Los siguientes comandos SQL pueden usarse en comandos preparados: `CREATE TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE`, y la mayoría de comandos `SHOW` . Otros comandos no se soportan todavía.

Los siguientes ejemplos muestran dos formas equivalentes de preparar un comando que calcula la hipotenusa de un triángulo dadas las longitudes de los dos catetos.

El primer ejemplo muestra cómo crear un comando preparado usando una cadena literal para proporcionar el texto del comando:

```
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;
```

El segundo ejemplo es similar, pero proporciona el texto del comando como variable de usuario:

```
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;
```

La sintaxis SQL para comandos preparados no puede usarse anidada. Esto es, un comando pasado a `PREPARE` no puede ser el mismo un comando `PREPARE`, `EXECUTE`, o `DEALLOCATE PREPARE` .

Además, la sintaxis SQL para comandos preparados es distinta de la usada en llamadas a la API de comandos preparados. Por ejemplo, use la función `mysql_stmt_prepare()` de la API C para preparar un comando `PREPARE`, `EXECUTE`, o `DEALLOCATE PREPARE` .

La sintaxis SQL para comandos preparados no puede usarse dentro de procedimientos almacenados y funciones.

Capítulo 14. Motores de almacenamiento de MySQL y tipos de tablas

Tabla de contenidos

MySQL soporta varios motores de almacenamiento que tratan con distintos tipos de tabla. Los motores de almacenamiento de MySQL incluyen algunos que tratan con tablas transaccionales y otros que no lo hacen:

- MyISAM trata tablas no transaccionales. Proporciona almacenamiento y recuperación de datos rápida, así como posibilidad de búsquedas fulltext. MyISAM se soporta en todas las configuraciones MySQL, y es el motor de almacenamiento por defecto a no ser que tenga una configuración distinta a la que viene por defecto con MySQL.

- El motor de almacenamiento MEMORY proporciona tablas en memoria. El motor de almacenamiento MERGE permite una colección de tablas MyISAM idénticas ser tratadas como una simple tabla. Como MyISAM, los motores de almacenamiento MEMORY y MERGE tratan tablas no transaccionales y ambos se incluyen en MySQL por defecto.

  Nota: El motor de almacenamiento MEMORY anteriormente se conocía como HEAP.

- Los motores de almacenamiento InnoDB y BDB proporcionan tablas transaccionales. BDB se incluye en la distribución binaria MySQL-Max en aquellos sistemas operativos que la soportan. InnoDB también se incluye por defecto en todas las distribuciones binarias de MySQL 5.0 . En distribuciones fuente, puede activar o desactivar estos motores de almacenamiento configurando MySQL a su gusto.

- El motor de almacenamiento `EXAMPLE` es un motor de almacenamiento "tonto" que no hace nada. Puede crear tablas con este motor, pero no puede almacenar datos ni recuperarlos. El objetivo es que sirva como ejemplo en el código MySQL para ilustrar cómo escribir un motor de almacenamiento. Como tal, su interés primario es para desarrolladores.
- `NDB Cluster` es el motor de almacenamiento usado por MySQL Cluster para implementar tablas que se particionan en varias máquinas. Está disponible en distribuciones binarias MySQL-Max 5.0. Este motor de almacenamiento está disponible para Linux, Solaris, y Mac OS X . Añadiremos soporte para este motor de almacenamiento en otras plataformas, incluyendo Windows en próximas versiones.
- El motor de almacenamiento `ARCHIVE` se usa para guardar grandes cantidades de datos sin índices con una huella muy pequeña.
- El motor de almacenamiento `CSV` guarda datos en ficheros de texto usando formato de valores separados por comas.
- El motor de almacenamiento `FEDERATED` se añadió en MySQL 5.0.3. Este motor guarda datos en una base de datos remota. En esta versión sólo funciona con MySQL a través de la API MySQL C Client. En futuras versiones, será capaz de conectar con otras fuentes de datos usando otros drivers o métodos de conexión clientes.

Este capítulo describe cada uno de los motores de almacenamiento MySQL excepto `InnoDB` y `NDB Cluster`, que se tratan en Capítulo 15, *El motor de almacenamiento `InnoDB`* y Capítulo 16, *MySQL Cluster*.

Cuando crea una nueva tabla, puede decirle a MySQL qué tipo de tabla crear añadiendo la opción de tabla `ENGINE` o `TYPE` al comando `CREATE TABLE` :

```
CREATE TABLE t (i INT) ENGINE = INNODB;
CREATE TABLE t (i INT) TYPE = MEMORY;
```

Aunque se soporta `TYPE` en MySQL 5.0, `ENGINE` es el término preferido.

Si omite la opción `ENGINE` o `TYPE`, se usa el motor de almacenamiento por defecto, que es `MyISAM`. Puede cambiarlo usando las opciones de arranque `--default-storage-engine` o `--default-table-type` , o cambiando la variable de sistema `storage_engine` o `table_type` .

Cuando se instala MySQL en Windows usando el MySQL Configuration Wizard, `InnoDB` es el motor de almacenamiento por defecto en lugar de `MyISAM`. Consulte Sección 2.3.5.1, “Introducción”.

Para convertir una tabla de un tipo a otro, use un comando `ALTER TABLE` que indique el nuevo tipo:

```
ALTER TABLE t ENGINE = MYISAM;
ALTER TABLE t TYPE = BDB;
```

Consulte Sección 13.1.5, “Sintaxis de `CREATE TABLE`” y Sección 13.1.2, “Sintaxis de `ALTER TABLE`”.

Si trata de usar un motor de almacenamiento que no está compilado o que está desactivado, MySQL crea una tabla de tipo `MyISAM`. Este comportamiento es conveniente cuando quiere copiar tablas entre servidores MySQL que soportan distintos motores. (Por ejemplo, en una inicialización de replicación, tal vez su maestro suporte un motor de almacenamiento transaccional para más seguridad, pero los esclavos usan un motor de almacenamiento no transaccional para mayor velocidad.)

La sustitución automática del tipo `MyISAM` cuando se especifica un tipo no especificado puede ser confuso para nuevos usuarios. En MySQL 5.0, se genera una advertencia cuando se cambia un tipo de tabla automáticamente.

MySQL siempre crea un fichero .frm para guardar la definición de tabla y columnas. El índice y datos de la tabla puede estar almacenado en uno o más ficheros, en función del tipo de tabla. El servidor crea el fichero .frm por encima del nivel de almacenamiento del motor. Los motores de almacenamiento individuales crean los ficheros adicionales necesarios para las tablas que administran.

Una base de datos puede contener tablas de distintos tipos.

Las tablas transaccionales (TSTs) tienen varias ventajas sobre las no transaccionales (NTSTs):

- Más seguras. Incluso si MySQL cae o tiene problemas de hardware, puede recuperar los datos, mediante recuperación automática o desde una copia de seguridad más el log de transacciones.
- Puede combinar varios comandos y aceptarlos todos al mismo tiempo con el comando COMMIT (si autocommit está desactivado).
- Puede ejecutar ROLLBACK para ignorar los cambios (si autocommit está desactivado).
- Si falla una actualización, todos los cambios se deshacen. (Con tablas no transaccionales, todos los cambios son permanentes.)
- Motores de almacenamiento transaccionales pueden proporcionar mejor concurrencia para tablas que tienen varias actualizaciones concurrentes con lecturas.

En MySQL 5.0, InnoDB usa valores de configuración por defecto si no los especifica. Consulte Sección 15.3, “Configuración de InnoDB”.

Tablas no transaccionales tienen varias ventajas al no tener una sobrecarga transaccional:

- Más rápidas
- Menor requerimiento de espacio.
- Menos memoria para actualizaciones

Puede combinar tablas transaccionales y no transaccionales en el mismo comando para obtener lo mejor de ambos mundos. Sin embargo, en una transaccion con autocommit desactivado, los cambios de tablas no transaccionales son permanentes inmediatamente y no pueden deshacerse.

14.1. El motor de almacenamiento MyISAM

MyISAM es el motor de almacenamiento por defecto. Se basa en el código ISAM pero tiene muchas extensiones útiles. (Tenga en cuenta que MySQL 5.0 no soporta ISAM.)

Cada tabla MyISAM se almacena en disco en tres ficheros. Los ficheros tienen nombres que comienzan con el nombre de tabla y tienen una extensión para indicar el tipo de fichero. Un fichero .frm almacena la definición de tabla. El fichero de datos tiene una extensión .MYD (MYData) . El fichero índice tiene una extensión .MYI (MYIndex) .

Para especificar explícitamente que quiere una tabla MyISAM, indíquelo con una opción ENGINE:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

(**Nota**: Antiguas versiones de MySQL usaban TYPE en lugar de ENGINE (por ejemplo: TYPE = MYISAM). MySQL 5.0 soporta esta sintaxis para compatibilidad con versiones anteriores pero TYPE está obsoleto y ahora se usa ENGINE .)

Normalmente, la opción ENGINE no es necesaria; MyISAM es el motor de almacenamiento por defecto a no ser que se cambie.

Puede chequear o reparar tablas MyISAM con la utilidad myisamchk . Consulte Sección 5.8.3.7, “Usar myisamchk para recuperación de desastres”. Puede comprimir tablas MyISAM con myisampack para que ocupen mucho menos espacio. Consulte Sección 8.2, “myisampack, el generador de tablas comprimidas de sólo lectura de MySQL”.

Las siguientes son algunas características del motor de almacenamiento MyISAM :

- Todos los datos se almacenan con el byte menor primero. Esto hace que sean independientes de la máquina y el sistema operativo. El único requerimiento para portabilidad binaria es que la máquina use enteros con signo en complemento a dos (como todas las máquinas en los últimos 20 años) y formato en coma flotante IEEE (también dominante en todas las máquinas). La única área de máquinas que pueden no soportar compatibilidad binaria son sistemas empotrados, que a veces tienen procesadores peculiares.

  No hay penalización de velocidad al almacenar el byte menor primero; los bytes en un registro de tabla normalmente no están alineados y no es un problema leer un byte no alineado en orden normal o inverso. Además, el código en el servidor que escoge los valroes de las columnas no es crítico respecto a otro código en cuanto a velocidad.

- Ficheros grandes (hasta longitud de 63 bits) se soportan en sistemas de ficheros y sistemas operativos que soportan ficheros grandes.

- Registros de tamaño dinámico se fragmentan mucho menos cuando se mezclan borrados con actualizaciones e inserciones. Esto se hace combinando automáticamente bloques borrados adyacentes y extendiendo bloques si el siguiente bloque se borra.

- El máximo número de índices por tabla MyISAM en MySQL 5.0 es 64. Esto puede cambiarse recompliando. El máximo número de columnas por índice es 16.

- La longitud máxima de clave es 1000 bytes. Esto puede cambiarse recompilando. En caso de clave mayor a 250 bytes, se usa un tamaño de bloque mayor, de 1024 bytes.

- Las columnas BLOB y TEXT pueden indexarse.

- Valores NULL se permiten en columnas indexadas. Esto ocupa 0-1 bytes por clave.

- Todos los valores de clave numérico se almacenan con el byte mayor primero para mejor compresión de índice.

- Cuando se insertan registros en orden (como al usar columnas AUTO_INCREMENT), el árbol índice se divide de forma que el nodo mayor sólo contenga una clave. Esto mejora la utilización de espacio en el árbol índice.

- El tratamiento interno de una columna AUTO_INCREMENT por tabla. MyISAM actualiza automáticamente esta colunmna para operaciones INSERT y UPDATE . Esto hace las columnas AUTO_INCREMENT más rápidas (al menos 10%). Los valores iniciales de la secuencia no se reúsan tras ser borrados. (Cuando una columna AUTO_INCREMENT se define como la última columna de un índice de varias columnas, se reúsan los valores borrados iniciales de la secuencia.) El valor AUTO_INCREMENT puede cambiarse con ALTER TABLE o myisamchk.

- Si una tabla no tiene bloques libres en medio del fichero de datos, puede INSERT nuevos registros a la vez que otros flujos leen de la tabla. (Esto se conoce como inserciones concurrentes.) Un bloque libre puede ser resultado de borrar o actualziar registros de longitud dinámica con más datos que su contenido. Cuando todos los bloques libres se usan (se rellenan), las inserciones futuras vuelven a ser concurrentes.

- Puede tener el fichero de datos e índice en directorios distintos para obtener más velocidad con las opciones DATA DIRECTORY y INDEX DIRECTORY para CREATE TABLE. Consulte Sección 13.1.5, “Sintaxis de CREATE TABLE”.
- Cada columna de carácteres puede tener distintos conjuntos de carácteres. Consulte Capítulo 10, *Soporte de conjuntos de caracteres*.
- Hay un flag en el fichero índice MyISAM que indica si la tabla se ha cerrado correctamente. Si mysqld se arranca con la opción --myisam-recover , Las tablas MyISAM se chequean automáticamente al abrirse, y se reparan si la tabla no se cierra correctamente.
- myisamchk marca las tablas como chequeadas si se ejecuta con la opción --update-state . myisamchk --fast cheque sólo las tablas que no tienen esta marca.
- myisamchk --analyze almacena estadísticas para partes de las claves, así como para las claves enteras.
- myisampack puede comprimir columnas BLOB y VARCHAR .

MyISAM soporta las siguientes características:

- Soporte de un tipo VARCHAR auténtico; una columna VARCHAR comienza con la longitud almacenada en dos bytes.
- Tablas con VARCHAR pueden tener longitud de registro fija o dinámica.
- VARCHAR y CHAR pueden ser de hasta 64KB.
- Un índice hash puede usarse para UNIQUE. Esto le permite tener UNIQUE o cualquier combinación de columnas en una tabla . (Sin embargo, no puede buscar en un índice UNIQUE .)

14.1.1. Opciones de arranque de MyISAM

Las siguientes opciones de mysqld pueden usarse para cambiar el comportamiento de tablas MyISAM :

- --myisam-recover=*mode*

  Cambia el modo para recuperación automática para tablas MyISAM .

- --delay-key-write=ALL

  No vuelca buffers de clave entre escrituras para cualquier tabla MyISAM .

  Nota: Si hace esto, no debe usar tablas MyISAM desde otro programa (como otro servidor MySQL server o con myisamchk) cuando la tabla está en uso. Hacerlo provoca corrupción de índice.

  Usar --external-locking no ayuda para tablas que usan --delay-key-write.

Consulte Sección 5.3.1, “Opciones del comando mysqld”.

Las siguientes variables de sistema afectan al comportamiento de tablas MyISAM :

- bulk_insert_buffer_size

  Tamaño del árbol de caché usado en optimización de inserciones. **Nota**: Este es el límite *por flujo*!

- myisam_max_extra_sort_file_size

Usado para ayudar a MySQL a decidir cuándo usar el método de creación de índice de clave caché lento pero seguro **Nota**: Este parámetro se daba en bytes antes de MySQL 5.0.6, cuando se eliminó.

- `myisam_max_sort_file_size`

  No usa el método de ordenación de índice rápido para crear un índice si el fichero temporal será más grande a este tamaño. **Nota**: En MySQL 5.0, este parámetro se da en bytes.

- `myisam_sort_buffer_size`

  Cambia el tamaño del búffer usado al recuperar tablas.

Consulte Sección 5.3.3, “Variables de sistema del servidor”.

La recuperación automática se activa si arranca `mysqld` con la opción `--myisam-recover` . En ese caso, cuando el servidor abre una tabla `MyISAM` , chequea si la tabla está marcada como mal cerrada o si el contador de veces que se ha abierto la tabla no es 0 y está ejecuando el servidor con `--skip-external-locking`. Si alguna de estas condiciones es cierta, ocurre lo siguiente:

- Se chequea la tabla para errores.
- Si el servidor encuentra un error, trata de hacer una reparación de tabla rápida (ordenando y sin recrear el fichero de datos).
- Si falla la reparación debido a un error en el fichero de datos (por ejemplo, error de clave duplicada), el servidor lo intenta otra vez, esta vez recreando el fichero de datos.
- Si sigue fallando, el servidor trata una vez más con el método de reparación antiguo (escrito registro a registro sin ordenar). Este método debe ser capaz de reparar cualquier clase de error y tiene requerimientos de espacio bajos.

Si la recuperación no fuera capaz de recuperar todos los registros de un comando préviamente completado y no ha especificado `FORCE` en la opción `--myisam-recover` , la recuperación automática aborta con un mensaje de error en el log de errores:

```
Error: Couldn't repair table: test.g00pages
```

Si especifica `FORCE`, se escribe una advertencia como esta:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Tenga en cuenta qui el valor de recuperación automático incluye `BACKUP`, el proceso de recuperación crea ficheros con nombres de la forma *`tbl_name-datetime`*`.BAK`. Debe tener un script `cron` que mueva estos ficheros automáticamente del directorio de base de datos al dispositivo de copia de seguridad.

14.1.2. Cuánto espacio necesitan las claves

Tablas `MyISAM` usan índices B-tree. Puede calcular el tamaño del fichero índice mediante `(key_length+4)/0.67`, sumado sobre todas las claves. Este es el peor caso en que todas las claves se insertan en orden ordenado y la tabla no tienen ninguna clave comprimida.

Los índices de cadenas de carácteres están comprimidos en espacio. Si la primera parte del índice es una cadena de carácteres, también tiene el prefijo comprimido. La compresión de espacio hace que el fichero índice sea menor que el peor caso si la columna de la cadena de carácteres tiene muchos espacios

finales o es una columna VARCHAR que no se usa siempre con la longitud total. La compresión de prefijo se usa en claves que comienzan con una cadena de carácteres. La compresión de prefijo ayuda si hay muchas cadenas de carácteres con un prefijo idéntico.

En tablas MyISAM puede comprimir números prefijo especificando PACK_KEYS=1 cuando crea la tabla. Esto ayuda cuando tiene muchas claves enteras con un prefijo idéntico cuando los números se almacenan con el byte mayor primero.

14.1.3. Formatos de almacenamiento de tablas MyISAM

MyISAM soporta tres formatos de almacenamiento distintos. Dos de ellos (formato fijo y dinámico) se eligen automáticamente en función del tipo de columnas que esté usando. El tercero, formato comprimido, puede ser creado sólo con la utilidad myisampack .

Cuando CREATE o ALTER una tabla sin columnas BLOB o TEXT , puede forzar el formato de tabla a FIXED o DYNAMIC con la optión de tabla ROW_FORMAT . Esto hace que las columnas CHAR y VARCHAR sean CHAR para formato FIXED, o VARCHAR para formato DYNAMIC .

Puede comprimir o descomprimir tablas especificando ROW_FORMAT={COMPRESSED | DEFAULT} con ALTER TABLE. Consulte Sección 13.1.5, “Sintaxis de CREATE TABLE”.

14.1.3.1. Características de tablas estáticas (con ancho fijo o Fixed-Length)

El formato por defecto para MyISAM es el estático. Se usa cuando la tabla no contiene columnas de longitud variable (VARCHAR, BLOB, o TEXT). Cada registro se almacena usando un número de bytes fijo.

De los tres formatos de almacenamiento MyISAM, el formato estático es el más simple y seguro (menos sujeto a corrupción). También es el más rápido de los formatos sobre disco. La velocidad proviene de la facilidad con que se encuentran los registros en el fichero de datos en disco: Cuando se busca un registro basándose en un número de registro en el índice, multiplica el número de registro por la longitud de registro. También , al escanear una tabla, es muy fácil leer un número constante de registro con cada operación de lectura de disco.

La seguridad se evidencia si su máquina falla mientras el servidor MySQL está escribiendo en un fichero MyISAM de formato fijo. En este caso, myisamchk puede determinar fácilmente dónde comienza cada registro y dónde acaba, así que usualmente puede recuperar todos los registros excepto los parcialmente escritos. Tenga en cuenta que los índices de tabla MyISAM siempre pueden reconstruirse basados en los registros de datos.

Características generales de tablas de formato estático:

- Las columnas CHAR añaden espacios hasta la anchura de columna. Esto también es cierto para columnas NUMERIC, y DECIMAL creadas antes de MySQL 5.0.3.
- Muy rápido.
- Fácil de cachear.
- Fácil de reconstruir tras un fallo, ya que los registros se localizan en posiciones fijas.
- La reorganización no es necesaria a no ser que borre un gran número de registros y quiera devolver espacio libre al sistema operativo. Para ello, use OPTIMIZE TABLE o myisamchk -r.
- Usualmente requiere más espacio de disco que para tablas de formato dinámico.

14.1.3.2. Características de tablas dinámicas

El formato de almacenamiento dinámico se usa si una tabla MyISAM contiene alguna columna de longitud variable (VARCHAR, BLOB, o TEXT), o si la tabla se crea con la opción ROW_FORMAT=DYNAMIC .

Este formato es un poco más complejo ya que cada columna tiene una cabecera que indica la longitud. Un registro puede acabar en más de una localización cuando es alarga como resultado de una actualización.

Puede usar OPTIMIZE TABLE o myisamchk para defragmentar una tabla. Si tiene columnas de longitud fija a las que accede o cambia frecuentemente en una tabla que también contenga alguna columna de longitud variable, puede ser buena idea mover las columnas de longitud variable a otras tablas para evitar fragmentación.

Características generales de tablas de formato dinámico:

- Todas las columnas de cadenas de carácteres son dinámicas excepto aquéllas con longitud menor a cuatro.
- Cada registro viene precedido por un bitmap que indica qué columnas contienen la cadena vacía (para columnas de cadenas) o cero (para columnas numéricas). Tenga en cuenta que esto no incluye columnas que contienen valores NULL. Si una columna de cadena de carácteres tiene una longitud de cero tras eliminar los espacios en blanco finales, o una columna numérica tiene un valor de cero, se marca en el bitmap y no se guarda en disco. Las cadenas no vacías se guardan como un byte de longitud más al de los contenidos de la cadena.
- Para tablas de longitud fija normalmente se necesita mucho menos espacio de disco.
- Cada registro usa sólo tanto espacio como necesita. Sin embargo, si un registro crece, se divide en tantos trozos como haga falta, resultando en una fragmentación de registro. Por ejemplo, si actualiza un registro con información que alarga la longitud del registro, el registro se fragmenta. En este caso, puede que tenga que ejecutar OPTIMIZE TABLE o myisamchk -r de vez en cuando para mejorar el rendimiento. Use myisamchk -ei para obtener estadísticas de tabla.
- Más difícil de reconstruir tras un fallo que las tablas de formato estático, ya que los registros pueden fragmentarse en varios trozos y puede faltar algún enlace (fragmento).
- La longitud de registro esperada para registros de longitud dinámica se calcula usando la siguiente expresión:

```
3
+ (número de columnas + 7) / 8
+ (número de columnas de carácteres)
+ (tamaño comprimido de columnas de cadenas)
+ (longitud de cadenas)
+ (número de columnas NULL + 7) / 8
```

  Hay una penalización de 6 bytes para cada enlace. Un registro dinámico se enlaza si una actualización provoca aumentar el tamaño de un registro. Cada nuevo enlace es al menos de 20 bytes, así que la siguiente ampliación probablemente irá en el mismo enlace. Si no es así, se crea otro enlace. Puede encontrar el número de enlaces usando myisamchk -ed. Pueden eliminarse todos los enlaces con myisamchk -r.

14.1.3.3. Características de las tablas comprimidas

El formato de almacenamiento comprimido es de sólo lectura generado con la herramienta myisampack.

Todas las distribuciones MySQL incluyen por defecto myisampack. Los escaneos de tablas comprimidas son descomprimidas por myisamchk.

Las tablas comprimidas tienen las siguientes características:

- Las tablas comprimidas ocupan muy poco espacio. Esto minimiza el uso de disco, lo que es útil al usar discos lentos (como CD-ROMs).
- Cada registros se comprime por separado, así que hay poca sobrecarga de acceso. La cabecera de un registro ocupa de 1 a 3 bytes en función del registro más grande en la tabla. Cada columna está comprimida de forma distinta. Usualmente hay una árbol de Huffman para cada columna. Algunos de los tipos de compresión son:
  - Compresión espacial de sufijo.
  - Compresión espacial de prefijo.
  - Números con valor de cero se almacenan usando un bit.
  - Si los valores de una columna entara tienen un rango pequeño, la columna se almacena usando el tipo menor posible. Por ejemplo, una columna `BIGINT` (ocho bytes) puede almacenarse como columna `TINYINT` (un byte) si todos los valores están en el rango de `-128` a `127`.
  - Si una columna tiene sólo un pequeño conjunto de valores posibles, el tipo de columna se convierte a `ENUM`.
  - Una columna puede usar cualquier combinación de los tipos de compresión precedentes.
- Pueden tratar registros de longitud fija o variable.

14.1.4. Problemas en tablas `MyISAM`

El formato de fichero que usa MySQL para almacenar datos se ha probado extensivamente, pero siempre hay circunstancias que pueden hacer que las tablas se corrompan.

14.1.4.1. Tablas `MyISAM` corruptas

Incluso el formato de tabla `MyISAM` es muy fiable (todos los cambios hechos en una tabla por un comando SQL se escriben antes que retorne el comando), puede obtener tablas corruptas si cualquiera de los siguientes eventos ocurre:

- El proceso `mysqld` muere durante una escritura.
- La máquina se apaga inesperadamente.
- Fallos de hardware.
- Usa un programa externo (como `myisamchk`) en una tabla que está siendo modificada por el servidor a la vez.
- Un bug en el código de MySQL o `MyISAM`.

Los síntomas típicos de una tabla corrupta son:

- Obtiene el siguiente error al seleccionar datos de una tabla:

```
Incorrect key file for table: '...'. Try to repair it
```

- Las consultas no obtienen registros en la tabla o retornan datos incompletos.

Puede chequear la salud de una tabla MyISAM usando el comando CHECK TABLE , y reparar una tabla MyISAM corrupta con REPAIR TABLE. Cuando mysqld no está en ejecución, puede chequear o reparar una tabla con el comando myisamchk . Consulte Sección 13.5.2.3, “Sintaxis de CHECK TABLE”, Sección 13.5.2.6, “Sintaxis de REPAIR TABLE”, and Sección 5.8.3.1, “Sintaxis para invocar myisamchk”.

Si sus tablas se corrompen frecuentemente, debe tratar de determinar porqué ocurre. Lo más importante es saber si la tabla se corrompe como resultado de un fallo de servidor. Puede verificarlo fácilmente buscando un mensaje restarted mysqld reciente en el log de errores. Si encuentra dicho mensaje, es probable que la corrupción de tabla sea resultado de la caída del servidor. De otro modo, la corrupción pudo haber ocurrido durante operaciones normales. Esto es un bug. Debe tratar de crear un caso de test reproducible que demuestre el problema. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)” y Sección D.1.6, “Crear un caso de prueba tras haber encontrado una tabla corrupta”.

14.1.4.2. Problemas debidos a tablas que no se han cerrado debidamente

Cada fichero índice MyISAM (.MYI) tiene un contador en la cabecera que puede usarse para chequear si una tabla se ha cerrado correctamente. Si obtiene la siguiente advertencia de CHECK TABLE o myisamchk, significa que el contador se ha desincronizado:

```
clients are using or haven't closed the table properly
```

Esta advertencia no significa necesariamente que la tabla esté corrupta, pero al menos debe chequear la tabla.

El contador funciona como se muesta:

- La primera vez que se actualiza la hora de una tabla en MySQL, se incrementa un contador en la cabecera del fichero índice.
- El contador no cambia durante otras actualizaciones.
- Cuando se cierra la última instancia de una tabla (debido a una operación FLUSH TABLES o porque no hay espacio en la caché de la tabla), el contador se decrementa si la tabla se ha actualizado en cualquier punto.
- Cuando repara la tabla o chequea la tabla y está correcta, el contador se resetea a cero.
- Para evitar problemas con interacciones con otros procesos que pueden chequear la tabla, el contador no se decrementea al cerrar si era cero.

En otras palabras, el contador puede desincronizarse sólo bajo las siguientes condiciones:

- Las tablas MyISAM se copian sin ejecutar en primer lugar LOCK TABLES y FLUSH TABLES.
- MySQL falla entre una actualización y el cierre final. (Tenga en cuenta que la tabla puede estar ok, ya que MySQL siempre realiza escrituras entre cada comando.)
- Una tabla se modifica con myisamchk --recover o myisamchk --update-state a la vez que se usa con mysqld.
- Usando múltiples servidores mysqld usando la tabla y un servidor realiza un REPAIR TABLE o CHECK TABLE en la tabla mientras estaba en uso por otro servidor. En este caso, se puede usar CHECK TABLE, aunque puede obtener una advertencia de otros servidores. Sin embargo, REPAIR TABLE debe evitarse ya que cuando un servidor reemplaza el fichero de datos con uno nuevo, no se envía a los otros servidores.

En general, no es buena idea compartir un directorio de datos entre varios servidores. Consulte Sección 5.11, “Ejecutar más de un servidor MySQL en la misma máquina” para más informaicón.

14.2. El motor de almacenamiento MERGE

El motor de almacenamiento MERGE , también conocido como MRG_MyISAM , es una colección de tablas MyISAM idénticas que pueden usarse como una. "Idéntica"significa que todas lsas tablas tienen información de columna e índice idéntica. No puede mezclar tablas en que las columnas se listen en orden distinto, no tengan exactamente las mismas columnas, o tengan los índices en orden distinto. Sin embargo, alguna o todas las tablas pueden comprimirse con myisampack. Consulte Sección 8.2, “myisampack, el generador de tablas comprimidas de sólo lectura de MySQL”. Diferencias en las opciones de las tablas tales como AVG_ROW_LENGTH, MAX_ROWS, o PACK_KEYS no importan.

Cuando crea una tabla MERGE , MySQL crea dos ficheros en disco. Los ficheros tienen nombres que comienzan con el nombre de la tabla y tienen una extensión para indicar el tipo de fichero, Un fichero .frm almacena la definición de tabla , y un fichero .MRG contiene los nombres de las tablas que deben usarse como una. Las tablas no tienen que estar en la misma base de datos que la tabla MERGE misma.

Puede usar SELECT, DELETE, UPDATE, y INSERT en la colección de tablas, Debe tener permisos de SELECT, UPDATE, y DELETE en las tablas que mapea a una tabla MERGE.

Si hace un DROP de la tabla MERGE , sólo borra la especificación MERGE . Las tablas subyacentes no se ven afectadas.

Cuando crea una tabla MERGE , debe especificar una cláusula UNION=(*list-of-tables*) que indica qué tablas quiere usar como una. Puede especificar opcionalmente una opción INSERT_METHOD si quiere que las inserciones en la tabla MERGE se realicen en la primera o última tabla de la lista UNION . Use un valor de FIRST o LAST para hacer que las inserciones se hagan en la primera o última tabla, respectivamente. Si no especifica una opción INSERT_METHOD o si la especifica con un valor de NO, intentos de insertar registros en la tabla MERGE producen un error.

El siguiente ejemplo muestra cómo crear una tabla MERGE :

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20));
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20));
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))
    ->    TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Tenga en cuenta que la columna a está indexada en la tabla MERGE , pero no está declarada como PRIMARY KEY como lo está en las tablas MyISAM subyacente. Esto es necesario ya que una tabla MERGE no puede forzar unicidad en un conjunto de tablas subyacentes.

Tras crear la tabla MERGE, puede realizar consultas que operen en el grupo de tablas como unidad:

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
```

```
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

Tenga en cuenta que puede manipular el fichero `.MRG` directamente desde fuera del servidor MySQL :

```
shell> cd /mysql-data-directory/current-database
shell> ls -1 t1 t2 > total.MRG
shell> mysqladmin flush-tables
```

Para rempaear una tabla `MERGE` a una colección diferente de tablas `MyISAM`, puede realizar una de las siguientes opciones:

- `DROP` la tabla `MERGE` y recrearla.
- Use `ALTER TABLE tbl_name UNION=(...)` para cambiar la lista de tablas subyacentes.
- Cambie el fichero `.MRG` y realice un comando `FLUSH TABLE` para la tabla `MERGE` y todas las tablas subyacentes para forzar al motor de almacenamiento a leer el nuevo fichero de definición.

Las tablas `MERGE` pueden ayudarle a arreglar los siguientes problemas:

- Administrar fácilmente un largo conjunto de tablas. Por ejemplo, puede poner datos de meses distintos en tablas separadas, comprimir algunos de ellos con `myisampack`, y luego crear una tabla `MERGE` para usarlas como una.
- Obtener más velocidad. Puede dividir una tabla grande de sólo lectura basándose en algún criterio, y luego poner las tablas individuales en distintos discos. Una tabla `MERGE` puede ser mucho más rápida que usar una tabla grande.
- Realizar búsquedas más eficientes. Si conoce exactamente lo que busca, puede buscar en sólo una de las tablas divididas y usar una tabla `MERGE` para las otras. Puede tener distintas tablas `MERGE` que usen conjuntos no disjuntos de tablas.
- Realizar reparaciones más eficientes. Es más fácil de reparar tablas individuales mapeadas en una tabla `MERGE` que reparar una única tabla grande.
- Mapear instantáneamente varias tablas como una. Una tabla `MERGE` no necesita mantener un índice propio ya que usa los índices de las tablas individuales. Como resultado, las colecciones de tablas `MERGE` son *muy* rápidas de crear o remapear. (Tenga en cuenta que debe especificar las definiciones de índices cuando crea una tabla `MERGE` , incluso cuando no se crean índices.)
- Si tiene un conjunto de tablas que une como una tabla grande bajo demanda o batch, debe crear una tabla `MERGE` sobre ellas bajo demanda. Esto es mucho más rápido y ahorra mucho espacio de disco.
- Excede el límite de tamaño del sistema operativo. Cada tabla `MyISAM` está ligada a este límite, pero una colección de tablas `MyISAM` no lo está.
- Puede crear un alias o sinónimo para una tabla `MyISAM` definiendo una tabla `MERGE` que mapee a una tabla. No debería haber un impacto de rendimiento realmente impactante al hacer esto (sólo un par de llamadas indirectas y llamadas `memcpy()` para cada lectura).

Las desventajas de las tablas `MERGE` son:

- Sólo puede usar tablas `MyISAM` idénticas para una tabla `MERGE` .

- No puede usar un número de características MyISAM en tablas MERGE . Por ejemplo, no puede crear índices FULLTEXT en tablas MERGE . (Puede crear índices FULLTEXT en las tablas MyISAM subyacentes, puero no puede buscar en la tabla MERGE con búsquedas full-text.)
- Si la tabla MERGE no es temporal, todas las tablas subyacentes MyISAM deben ser permanentes. Si la tabla MERGE es temporal, las tablas MyISAM pueden ser cualquier mezcla de tablas temporales y no temporales.
- Las tablas MERGE usan más descriptores de fichero. Si 10 clientes usan una tabla MERGE que mapee a 10 tablas, el servidor usa (10*10) + 10 descriptores de fichero. (10 descriptores de ficheros de datos para cada uno de los 10 clientes, y 10 descriptores de ficheros de índice compartidos entre los clients.)
- Las lecturas de claves son más lentas. Cuando lee una clave, el motor de almacenamiento MERGE necesita leer en todas las tablas subyacentes para chequear cuál se parece más a la clave dada. Si luego lee el siguiente, el motor MERGE necesita buscar en los buffers de lectura para buscar la siguiente clave. Sólo cuando se usa un búffer de claves el motor necesita leer el siguiente bloque de claves. Esto hace que las claves MERGE sean mucho más lentas en búsquedas eq_ref , pero no mucho más lentas en búsquedas ref . Consulte Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)” para más información sobre eq_ref y ref.

14.2.1. Problemas con tablas MERGE

A continuación se describen problemas conocidos con tablas MERGE:

- Si usa ALTER TABLE para cambiar una tabla MERGE a otro tipo de tabla, el mapeo de las tablas subyacentes se pierde. En su lugar, los registros de las tablas subyacentes MyISAM se copian en la tabla alterada, que luego se asigna al nuevo tipo.
- REPLACE no funciona.
- No puede usar DROP TABLE, ALTER TABLE, DELETE FROM sin una cláusula WHERE , REPAIR TABLE, TRUNCATE TABLE, OPTIMIZE TABLE, o ANALYZE TABLE en alguna de las tablas que se mapean en una tabla MERGE abierta. Si lo hace, la tabla MERGE puede referirse a la tabla original, lo que conduce a resultados inesperados. La forma más fácil de solucionar esto es realizar un comando FLUSH TABLES antes de realizar ninguna de estas operaciones para segurar que no quedan abiertas tablas MERGE .
- Una tabla MERGE no puede mantener restricciones UNIQUE sobre la tabla entera. Cuando realiza un INSERT, los datos van a la primera o última tabla MyISAM (dependiendo del valor de la opción INSERT_METHOD). MySQL se asegura que los valores de clave única sean únicos dentro de la tabla MyISAM , pero no entre todas las tablas en la colección.
- Cuando crea una tabla MERGE , no se chequea que las tablas subyacentes existan y tengan una estructura idéntica. Cuando se usa la tabla MERGE , MySQL chequea que la longitud del registro para todas las tablas mapeadas sea la misma, pero esto no es "a prueba de bombas". Si crea una tabla MERGE de tablas MyISAM disimilares, es muy posible que tenga problemas extraños.
- El orden de índices en la tabla MERGE y sus tablas subyacentes debe ser el mismo. Si usa ALTER TABLE para añadir un índice UNIQUE a una tabla usada en una tabla MERGE , y después usa ALTER TABLE para añadir un índice no único en la tabla MERGE , la ordenación de índice es distinta para las tablas si ya hay un índice no único en las tabla subyacentees. (Esto es porque ALTER TABLE pone los índices UNIQUE antes de los índices no únicos para facilitar detección rápida de claves duplicadas.) Consecuentemente, las consultas en tablas con tales índices pueden retornar resultados no esperados.
- DROP TABLE en una tabla en uso por una tabla MERGE no funciona en Windows ya que el mapeo del motor de almacenamiento MERGE está escondido por la capa superior de MySQL. Desde Windows no se permite el borrado de ficheros abiertos, debe volcar todas las tablas MERGE (con FLUSH TABLES) o borrar la tabla MERGE antes de borrar la tabla.

14.3. El motor de almacenamiento MEMORY (HEAP)

El motor de almacenamiento MEMORY crea tablas con contenidos que se almacenan en memoria.Éstas se conocían préviamente como HEAP . En MySQL 5.0, MEMORY es el término preferido, aunque HEAP se soporta para compatibilidad con versiones anteriores.

Cada tabla MEMORY está asociada con un fichero de disco. El nombre de fichero comienza con el nombre de la tabla y tiene una exensión de .frm para indicar que almacena la definición de la tabla.

Para especificar explícitamente que quiere una tabla MEMORY, indíquelo con una opción ENGINE :

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

Como indica su nombre, las tablas MEMORY se almacenan en memoria y usan índices hash por defecto. Esto las hace muy rápidas, y muy útiles para crear tablas temporales. Sin embargo, cuando se apaga el servidor, todos los datos almacenados en las tablas MEMORY se pierde. Las tablas por sí mismas continúan existiendo ya que sus definiciones se almacenan en ficheros .frm en disco, pero están vacías cuando reinicia el servidor.

Este ejemplo muestra cómo puede crear, usar , y borrar una tabla MEMORY :

```
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;
```

Las tablas MEMORY tienen las siguientes características:

- El espacio para tablas MEMORY se reserva en pequeños bloques. Las tablas usan el 100% del hashing dinámico para insrciones. No se necesita área de desbordamiento o espacio extra para claves. No se necesita espacio extra para listas libres. Los registros borrados se ponen en una lista encadenada y se reúsan cuando inserta nuevos datos en la tabla. Las tablas MEMORY no tienen ninguno de los problemas asociados con borrados más inserciones en tablas hasheadas.
- Las tablas MEMORY pueden tener hasta 32 índices por tabla, 16 columnas por índice y una clave de longitud máxima de 500 bytes.
- En MySQL 5.0, el motor MEMORY implementa índices HASH y BTREE . Puede espcificar uno u otro para cada índice añadiendo una cláusula USING tal y como se muestra:

  ```
  CREATE TABLE lookup
      (id INT, INDEX USING HASH (id))
      ENGINE = MEMORY;
  CREATE TABLE lookup
      (id INT, INDEX USING BTREE (id))
      ENGINE = MEMORY;
  ```

  Las características generales de B-trees e índices hash se describen en Sección 7.4.5, “Cómo utiliza MySQL los índices”.
- Puede tener claves no únicas en una tabla MEMORY. (Esta es una característica no común de implementaciones de índices hash.)
- En MySQL 5.0, puede usar INSERT DELAYED con tablas MEMORY . Consulte Sección 13.2.4.2, “Sintaxis de INSERT DELAYED”.

- Si tiene un índice hash en una tabla MEMORY que tenga un alto índice de duplicación de claves (muchas entradas de índice con el mismo valor), las actualizaciones a la tabla que afecten valores claves y todos los borrados son significativamente más lentos. El rango de esta ralentización es proporcional al rango de duplicación (o inversamente proporcional al grado cardinalidad). Pude usar un índice BTREE para evitar este problema.
- Las tablas MEMORY usan una longitud de registro fija.
- MEMORY no soporta columnas BLOB o TEXT .
- MEMORY en MySQL 5.0 incluye soporte para columnas AUTO_INCREMENT e índices en columnas que contengan valores NULL .
- Las tablas MEMORY se comparten entre todos los clientes (como cualquier otra tabla no-TEMPORARY).
- Los contenidos de las tablas MEMORY se almacenan en memora , lo que es una propiedad que las tablas MEMORY comparten con las tablas internas que el servidor va creando al procesar consultas. Sin embargo, los dos tipos de tablas difierne en que las tablas MEMORY no están sujetas a conversión de almacenamiento, mientras que las tablas internas sí:
  - Si una tabla interna llega a ser demasiado grande, el servidor la convierte automáticamente a una tabla en disco. El límite de tamaño lo determina la variable de sistema tmp_table_size .
  - Las tablas MEMORY nunca se convieren en tablas de disco. Para segurar que no comete un error accidentalmente, puede cambiar la variable de sistema max_heap_table_size para que imponga un tamaño máximo de tablas MEMORY . Para tablas individuales, puede especificar la opción de tabla MAX_ROWS en el comando CREATE TABLE .
- El servidor necesita suficiente memoria para mantener todas las tablas MEMORY en uso a la vez.
- Para liberar memoria usada por una tabla MEMORY cuando no se requiere su contenido, debe ejecutar DELETE FROM o TRUNCATE TABLE, o borrar toda la tabla con DROP TABLE.
- Si quiere rellenar una tabla MEMORY cuando arranca el servidor MySQL, puede usar la opción --init-file . Por ejemplo, puede usar comandos como INSERT INTO ... SELECT o LOAD DATA INFILE en este fichero para cargar la tabla de una fuente de datos persistente. Consulte Sección 5.3.1, “Opciones del comando mysqld” y Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.
- Si usa replicación, las tablas MEMORY del servidor maestro se vacían cuando se apaga y reinicia. Sin embargo, un esclavo no es consciente que se vacían estas tablas, así que retorna contenido desfasado si selecciona datos del mismo. En MySQL 5.0, cuando se usa una tabla MEMORY en el maestro por primera vez desde que arrancó el maestro, se escribe un comando DELETE FROM en el log binario del maestro automáticamente, resincronizando el maestro y el esclavo otra vez. Tenga en cuenta que incluso con esta estrategia, el esclavo tiene datos desfasados en la tabla en el intervalo entre el reinicio del maestro y el primer uso de la tabla. Sin embargo, si usa la opción --init-file para rellenar la tabla MEMORY al arrancar el maestro, se asegura que este intervalo sea cero.
- La memoria necesaria por un registro en una tabla MEMORY se calcula con la siguiente expresión:

```
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( ) representa un factor de redondeo para que la longitud del registro sea un múltiplo exacto del tamaño del puntero char . sizeof(char*) es 4 en máquinas de 32-bit y 8 en máquinas de 64-bit.

14.4. El motor de almacenamiento BDB (BerkeleyDB)

Sleepycat Software ha proporcionado a MySQL el motor de almacenamiento transaccional Berkeley DB. A este motor de almacenamiento se le llama tradicionalmente BDB. Se incluye soporte para BDB en distribuciones fuentes de MySQL y en distribuciones binarias MySQL-Max .

Las tablas BDB pueden tener una gran probabilidad de sobrevivir a fallos del sistema y ser capaces de realizar COMMIT y ROLLBACK en transacciones. La distibución fuente MySQL incluye una distribución BDB preparada para funcionar con MySQL. No puede usar una versión de BDB no preparada para funcionar con MySQL.

En MySQL AB trabajamos codo a codo con Sleepycat para mantener la calidad de la interfaz MySQL/BDB . (Incluso aunque Berkeley DB está muy testeado y es muy fiable, la interfaz MySQL se considera de calidad gamma. Continuamos mejorando y optimizándola.)

Cuando hay algún problema con tablas BDB, ayudamos a nuestros usuarios a localizar el problema y reproducir casos de test. Cualquiera de estos casos de test se envía a Sleepycat, que nos ayuda a encontrar y arreglar el problema. Con esta operación de dos fases, cualquier problema con tablas BDB puede tardar un poco más en ser resuelto que con nuestros motores de almacenamiento. Sin embargo, no anticipamos dificultades significativas con este procedimiento ya que el código Berkeley DB se usa en muchas aplicaciones distintas a MySQL.

Para información general sobre Berkeley DB, visite el sitio web de Sleepycat , http://www.sleepycat.com/.

14.4.1. Sistemas operativos que soporta BDB

Actualmente, sabemos que el motor de almacenamiento BDB funcioina con los siguientes sistemas operativos:

- Linux 2.x Intel
- Sun Solaris (SPARC y x86)
- FreeBSD 4.x/5.x (x86, sparc64)
- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x
- Windows NT/2000/XP

BDB *no* funciona con los siguientes sistemas operativos:

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA-64
- Linux 2.x s390
- Mac OS X

Nota: La lista precedente no está completa. La actualizamos cuando tenemos más información.

Si compila MySQL de las fuentes con soporte para tablas BDB , pero ocurre el siguiente error cuando arranca mysqld, significa que BDB no está soportado por su arquitectura:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded
Can't init databases
```

En este caso, debe recompilar MySQL sin soporte para tablas BDB o arrancar el servidor con la opción `--skip-bdb` .

14.4.2. Instalación de BDB

Si ha bajado una versión binaria de MySQL que incluya soporte para Berkeley DB, símplemente siga las instrucciones de instalación usuales. (Distribuciones MySQL-Max incluyen soporte BDB .)

Si compila MySQL de las fuentes, puede activar soporte BDB ejecutando `configure` con la opción `--with-berkeley-db` además de cualquier otra distribución que use normalmente. Descargue una distribución MySQL 5.0 , cambie la localización en su directorio más alto, y ejecute este comando:

```
shell> ./configure --with-berkeley-db [other-options]
```

Para más información consulte Sección 2.7, “Instalación de MySQL en otros sistemas similares a Unix”, Sección 5.1.2, “El servidor extendido de MySQL `mysqld-max`”, y Sección 2.8, “Instalación de MySQL usando una distribución de código fuente”.

14.4.3. Opciones de arranque de BDB

Las siguientes opciones de `mysqld` pueden usarse para cambiar el comportamiento del motor de almacenamiento BDB:

- `--bdb-home=`*`path`*

  Directorio base para tablas BDB . Debe ser el mismo directorio que use para `--datadir`.

- `--bdb-lock-detect=`*`method`*

  El método de bloqueo BDB . El valor debe ser `DEFAULT`, `OLDEST`, `RANDOM`, o `YOUNGEST`.

- `--bdb-logdir=`*`path`*

  El directorio del fichero de log BDB .

- `--bdb-no-recover`

  No arranca Berkeley DB en modo recuperación.

- `--bdb-no-sync`

  No vuelca síncronamente los logs BDB . Esta opción está obsolta; use `--skip-sync-bdb-logs` en su lugar (consulte la descripción de `--sync-bdb-logs`).

- `--bdb-shared-data`

  Arranca Berkeley DB en modo multi proceso. (No use `DB_PRIVATE` al inicializar Berkeley DB.)

- `--bdb-tmpdir=`*`path`*

  El directorio de ficheros temporales de BDB .

- `--skip-bdb`

  Desactiva el motor de almacenamiento BDB .

- `--sync-bdb-logs`

  Vuelca síncronamente los logs `BDB` . Esta opción está activada por defecto ; use `--skip-sync-bdb-logs` para desactivarla.

See Sección 5.3.1, “Opciones del comando `mysqld`”.

Si usa la opción `--skip-bdb` , MySQL no inicializa la biblioteca Berkeley DB library y esto ahorra mucha memoria. Sin embargo, si usa esta opción, no puede usar tablas `BDB` . Si trata de crear una tabla `BDB` , MySQL crea una tabla `MyISAM` en su lugar.

Normalmente, debe arrancar `mysqld` sin la opción `--bdb-no-recover` si quiere usar tablas `BDB` . Sin embargo, esto puede causar problemas cuando trata de arrancar `mysqld` si los ficheros de log de `BDB` están corruptos. Consulte Sección 2.9.2.3, “Arrancar y resolver problemas del servidor MySQL”.

Con la variable `bdb_max_lock` , puede especificar el máximo número de bloqueos que pueden estar activos en una tabla `BDB` . Por defecto son 10,000. Debe incrementar esto si ocurren errores como el siguiente cuando realiza transacciones largas o cuando `mysqld` tiene que examinar muchos registros para ejecutar una consulta:

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

Puede cambiar las variables `binlog_cache_size` y `max_binlog_cache_size` si usa transacciones de varios comandos muy largas. Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

Consulte Sección 5.3.3, “Variables de sistema del servidor”.

14.4.4. Características de las tablas `BDB`

Cada tabla `BDB` se almacena en disco en dos ficheros. Los ficheros tienen nombres que comienzan con el nombre de la tabla y tienen una extensión para indicar el tipo de fichero. Un fichero `.frm` almacena la definición de tabla, y un fichero `.db` contiene los datos de tabla e índices.

Para especificar explícitamente que quiere una tabla `BDB`, indíquelo con una opición de tabla `ENGINE` o `TYPE`:

```
CREATE TABLE t (i INT) ENGINE = BDB;
CREATE TABLE t (i INT) TYPE = BDB;
```

`BerkeleyDB` es sinónimo de `BDB` en la opción `ENGINE` o `TYPE` .

El motor de almacenamiento `BDB` proporciona tablas transaccionales. La forma de usar estas tablas depende del modo autocommit:

- Si está ejecutando con autocommit activado (por defecto), los cambios en las tablas `BDB` se efectúan inmediatamente y no pueden deshacerse.
- Si está ejecutando con autocommit desactivado, los cambios no son permanentes hasta que ejecuta un comando `COMMIT` . En lugar de hacer un commit puede ejecutar `ROLLBACK` para olvidar los cambios.

  Puede comenzar una transacción con el comando `BEGIN WORK` para suspender autocommit, o con `SET AUTOCOMMIT=0` para desactivar autocommit explícitamente.

Consulte Sección 13.4.1, “Sintaxis de `START TRANSACTION`, `COMMIT` y `ROLLBACK`”.

El motor de almacenamiento `BDB` tiene las siguientes características:

- En MySQL 5.0, las tablas BDB pueden tener hasta 31 índices por tabla, 16 columnas por índice, y un máximo de tamaño de clave de 1024 bytes.
- MySQL necesita una PRIMARY KEY en cada tabla BDB para que cada registro pueda identificarse unívocamente. Si no crea una explícitamente, MySQL crea y mantiene una PRIMARY KEY oculta. La clave oculta tiene una longitud de 5 bytes y se incrementa para cada intento de inserción. Esta clave no aparece en la salida de SHOW CREATE TABLE o DESCRIBE.
- La PRIMARY KEY es más rápida que cualquier otro índice, ya que la PRIMARY KEY se almacena junto con los datos. Los otros índices se almacenan como los datos claves + la PRIMARY KEY, por lo que es importante mantener la PRIMARY KEY tan pequeña como sea posible para ahorrar espacio en disco y tener más velocidad.

  Este comportamiento es similar al de InnoDB, donde las claves primarias pequeñas ahorran espacio no sólo en el índice primario sino también en índices secundarios.
- Si todas las columnas a las que accede en una tabla BDB son parte del mismo índice o parte de la clave primaria, MySQL puede ejcutar la consulta sin tener que acceder al registro. En una tabla MyISAM , esto sólo puede hacerse si las columnas son parte del mismo índice.
- El escaneo secuencial es más lento que para tablas MyISAM ya que los datos en tablas BDB se almacena en B-trees y no en un fichero de datos separado.
- Los valores clave no tienen compresión de prefijo ni sufijo como los valores clave en tablas MyISAM . En otras palabras, la información clave ocupa más espacio en tablas BDB en comparación con MyISAM .
- A menudo hay huecos en la tabla BDB que le permiten insertar nuevos registros en medio del árbol índice. Esto hace que las tablas BDB sean algo más grandes que las MyISAM.
- SELECT COUNT(*) FROM *tbl_name* es lento para tablas BDB , ya que no se mantiene conteo de registros en la tabla.
- El optimizador necesita saber el número aproximado de registros en la tabla. MySQL resuelve esto contando inerciones y manteniendo esto en un segmento separado en cada tabla BDB . Si no realiza muchos comandos DELETE o ROLLBACK , este número debe ser lo bastante adecuado para el optimizador MySQL . Sin embargo, MySQL almacena el número sólo al cerrar, así que puede ser incorrecto si el servidor termina inesperadamente. Puede que no sea fatal incluso si este número no es 100% correcto. Puede actualizar el contador de registros usando ANALYZE TABLE o OPTIMIZE TABLE. Consulte Sección 13.5.2.1, “Sintaxis de ANALYZE TABLE” y Sección 13.5.2.5, “Sintaxis de OPTIMIZE TABLE”.
- Bloqueo interno en tablas BDB se realiza a nivel de páginas.
- LOCK TABLES funciona en tablas BDB como con otras tablas. Si no usa LOCK TABLES, MySQL realiza un bloqueo interno de múltiple escritura en la tabla (un bloqueo que no bloquea otros escritores) para asegurar que la tabla está bloqueada apropiadamente si otro flujo realiza un bloqueo de tabla.
- Para poder deshacer transacciones, el motor de almacenamiento BDB mantiene ficheros de log. Para máximo rendimiento puede usar la opción --bdb-logdir para guardar los logs BDB en un disco distinto al que tiene las bases de datos.
- MySQL realiza un checkpoint cada vez que se comienza un nuevo fichero log BDB , y borra cualquier fichero de log BDB no necesario para transacciones actuales. Puede usar FLUSH LOGS en cualquier momento para hacer un checkpoint de tablas Berkeley DB.

  Para recuperación de desastres, debe usar copias de seguridad de tablas más el log binario de MySQL. Consulte Sección 5.8.1, “Copias de seguridad de bases de datos”.

Atención: Si borra ficheros de log antiguos que estén en uso, BDB no es capaz de recuperar todo y puede perder datos si algo no va bien.

- Las aplicaciones deben estar siempre preparadas para tratar casos donde cualquier cambio en una tabla BDB pueda causar un rollback automático y cualquier lectura puede fallar con un error de deadlock.
- Si obtiene un disco entero con una tabla BDB , obtiene un error (probablemente error 28) y la transacción debe deshacerse. Esto contrasta con tablas MyISAM en las que mysqld espera a tener más espacio libre para continuar.

14.4.5. Temas pendientes de arreglo para BDB

- Abrir muchas tablas BDB a la vez puede ser muy lento. Si va a usar tablas BDB no debe tener una caché de tabla muy grande (por ejemplo, con tamaño superior a 256) y debe usar la opción --no-auto-rehash cuando use el cliente mysql .
- SHOW TABLE STATUS no proporciona alguna información para tablas BDB :

```
mysql> SHOW TABLE STATUS LIKE 'bdbtest'\G
*************************** 1. row ***************************
           Name: bdbtest
         Engine: BerkeleyDB
        Version: 10
     Row_format: Dynamic
           Rows: 154
 Avg_row_length: 0
    Data_length: 0
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: NULL
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

- Optimizar rendimiento.
- No use bloqueos de página para operaciones de escaneo de tabla..

14.4.6. Limitaciones en las tablas BDB

La siguiente lista muestra restricciones a tener en cuenta al usar tablas BDB :

- Cada tabla BDB almacena en el fichero .db la ruta al fichero como si se hubiera creado. Esto se hizo para ser capaz de detectar bloqueos en un entorno multi usuario que soporte enlaces simbólicos. Como consecuencia, no es posible mover ficheros de tablas BDB de un directorio de base de datos a otro.
- Al hacer copias de seguridad de tablas BDB, debe usar mysqldump o hacer una copia de seguridad que incluya los ficheros para cada tabla BDB (los ficheros .frm y .db) así como los ficheros de log BDB . El motor de almacenamiento BDB almacena transacciones no acabadas en su log y las necesita cuando arranca mysqld . Los logs BDB son los ficheros en el directorio de datos con nombres de la forma log.XXXXXXXXXX (10 dígitos).
- Si una columan que permite valores NULL tiene un índice único, sólo un se permite un valor NULL. Esto difiere de otros motores de almacenamiento.

14.4.7. Errores que pueden darse en el uso de tablas BDB

- Si ocurre el siguiente error cuando arranca mysqld tras actualizar, significa que la nueva versión BDB no soporta el antiguo formato de log:

```
bdb:  Ignoring log file: .../log.XXXXXXXXXX:
unsupported log version #
```

  En este caso, debe borrar todos los logs BDB de su directorio de datos (los fichero con nombres que tienen el formato log.XXXXXXXXXX) y reinicie mysqld. También recomendamos que use mysqldump --opt para volcar las tablas BDB , borre las tablas, y las restaure del fichero de volcado.

- Si el modo autocommit está desactivado y borra una tabla BDB referenciada por otra transacción, puede obtener un mensaje de error de la siguiente forma en el log de error MySQL:

```
001119 23:43:56  bdb:  Missing log fileid entry
001119 23:43:56  bdb:  txn_abort: Log undo failed for LSN:
                      1 3644744: Invalid
```

  Esto no es fatal, pero hasta arreglar el problema, recomendamos que no borre tablas BDB excepto con el modo autocommit activado. (La forma de arreglarlo no es trivial.)

14.5. El motor de almacenamiento EXAMPLE

El motor de almacenamiento EXAMPLE es un motor de pruebas que no hace nada. Su propósito es servir como ejemplo en el código fuente MySQL para ilustrar cómo empezar a escribir nuevos motores de almacenamiento. Como tal, tiene interés principalmente para desarrolladores.

Para examinar la fuente del motor EXAMPLE , consulte el directorio sql/examples de distribuciones fuentes MySQL 5.0.

Para activar este motor de almacenamiento, use la opción --with-example-storage-engine con configure al compilar MySQL.

Cuando crea una tabla EXAMPLE , el servidor crea un fichero de definición de tabla en el directorio de base de datos. El fichero comienza con el nombre de tabla y tiene una extensión .frm . No se crean más ficheros. No puede almacenarse ni recuperarse datos de la tabla.

```
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)
```

El motor EXAMPLE no soporta indexación.

14.6. El motor de almacenamiento FEDERATED

El motor FEDERATED está disponible desde MySQL 5.0.3. Es un motor que accede a datos en tablas de bases de datos remotas en lugar de tablas locales.

Para examinar la fuente para el motor FEDERATED, consulte el directorio sql de una distribución fuente de MySQL 5.0.3 o posterior.

14.6.1. Instalación del motor de almacenamiento FEDERATED

Para activar este motor, use la opción `--with-federated-storage-engine` con `configure` al compilar MySQL.

14.6.2. Descripción del motor de almacenamiento FEDERATED

Cuando crea una tabla `FEDERATED` , el servidor crea un fichero de definición de tabla en el directorio de base de datos. El fichero comienza con el nombre de tabla y tiene extensión `.frm` . No se crean más ficheros, ya que los datos reales están en la base de datos remota. Esto difiere de cómo funcionan los motores con tablas locales.

Para tablas de bases de datos locales, los ficheros de datos son locales. Por ejemplo, si crea una tabla `MyISAM` llamada `users`, el handler `MyISAM` crea un fichero de datos llamado `users.MYD`. Un handler para tablas locales lee, inserta, borra y actualiza datos en ficheros de datos locales, y los registros se guardan en un formato particular del handler. Para leer registros, el handler debe parsear los datos en columnas. Para escribir registros, los valores de la columna deben convertirse al formato de registro usado por el handler y escribirse en el fichero de datos local.

Con el motor MySQL `FEDERATED` no hay ficheros de datos locales para una tabla (por ejemplo, no hay fichero `.MYD`). En su lugar, una base de datos remota almacena los datos que normalmente estarían en la tabla. Esto necesita el uso de la API del cliente MySQL para leer, borrar, actualizar e insertar datos. La recuperación de datos se inicia via un comando `SELECT * FROM tbl_name` . Para leer el resultado, los registros se tratan uno a uno usando la función de la API C `mysql_fetch_row()` y luego se convierten desde las columnas del conjunto de resultados `SELECT` al formato que el handler `FEDERATED` espera.

El flujo básico es el siguiente:

1. Llamadas SQL efectuadas localmente
2. API del handler MySQL (datos en formato del handler)
3. API del cliente MySQL (datos convertidos a llamadas SQL)
4. Base de datos remota -> API del cliente MySQL
5. Convierte el conjunto de resultados al formato del handler
6. API del handler -> registros resultado o conteo de registros afectados a local

14.6.3. Cómo usar las tablas FEDERATED

El procedimiento para usar tablas `FEDERATED` es muy simple. Normalmente, tiene dos servidores en ejecución, en la misma máquina o en distintas. (También es posible para una tabla `FEDERATED` usar otra tabla administrada por el mismo servidor, aunque no tiene mucho sentido.)

Primero, tiene que tener una tabla en el servidor remoto que quiera acceder con la tabla `FEDERATED` . Suponga que la tabla remota está en la base de datos `federated` y se define así:

```
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),
    KEY name (name),
    KEY other_key (other)
```

```
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1;
```

La opción de tabla ENGINE puede nombrar cualquier motor de almacenamiento; la tabla no tiene porqué ser MyISAM .

A continuación, cree una tabla FEDERATED en el servidor local para acceder a la tabla remota:

```
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),
    KEY name (name),
    KEY other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://root@remote_host:9306/federated/test_table';
```

La estructura de esta tabla debe ser exactamente la misma que la de la tabla remota, excepto que la opción de tabla ENGINE debe ser FEDERATED y la opción de tabla CONNECTION es una cadena de conexión que indica al motor FEDERATED cómo conectar al servidor remoto.

El motor FEDERATED crea sólo el fichero test_table.frm en la base de datos federated .

La información del equipo remoto indica el servidor remoto al que se conecta el servidor local, y la información de base de datos y tabla indica la tabla remota a usar como fichero de datos. En este ejemplo, el servidor remoto está indicado para ser remote_host corriendo en el puerto 9306, así que puede arrancar ese servidor para que escuche en el puerto 9306.

La forma general de la cadena de conexión en la opción CONNECTION es la siguiente:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Sólo se soporta mysql como *scheme* en este punto; la contraseña y el número de puerto son opcionales.

Aquí hay algunas cadenas de conexión de ejemplo:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

El uso de CONNECTION para especificar la cadena de conexión no es óptima y posiblemente cambiará en MySQL 5.1. Téngalo en cuenta al usar tablas FEDERATED, ya que significa que habrá que hacer modificaciones cuando ocurra.

Como cualquier contraseña usada se almacena en la cadena de conexión como texto plano, puede ser leído por cualquier usuario que pueda usar SHOW CREATE TABLE o SHOW TABLE STATUS para la tabla FEDERATED , o consultar la tabla TABLES en la base de datos INFORMATION_SCHEMA .

14.6.4. Limitaciones del motor de almacenamiento FEDERATED

Lo que el motor FEDERATED soporta y lo que no:

- En la primera versión, el servidor remoto debe ser un servidor MySQL. El soporte de FEDERATED para otros motores de bases de datos se añadirá en el futuro.

- La tabla remota a la que apunta una tabla FEDERATED *debe* existir antes de intentar acceder a ella.
- Es posible para una tabla FEDERATED apuntar a otra, pero debe tener cuidado de no crear un bucle.
- No hay soporte para transacciones.
- No hay forma que el motor FEDERATED sepa si la tabla remota ha cambiado. La razión es que la tabla debe funcionar como un fichero de datos que nunca se escribirá para hacer algo distinto a la base de datos. La integridad de los datos en la tabla local debe comprobarse si hay algún cambio en la base de datos remota.
- El motor FEDERATED soporta SELECT, INSERT, UPDATE, DELETE, e índices. No soporta ALTER TABLE, DROP TABLE, o cualquier otro comando Data Definition Language . La implementación actual no usa comandos preparados.
- La implementación usa SELECT, INSERT, UPDATE, y DELETE, pero no HANDLER.
- Las tablas FEDERATED no funcionan con la caché de consultas.

Algunas de estas limitaciones pueden mejorarse en futuras versiones del handler FEDERATED .

14.7. El motor de almacenamiento ARCHIVE

El motor de almacenamiento ARCHIVE se usa para guardar grandes cantidades de datos sin índices con una huella relativamente pequeña.

Para activar este motor, use la opción --with-archive-storage-engine con configure al compilar MySQL.

Cuando crea una tabla ARCHIVE , el servidor crea un fichero de definición de tabla en el directorio de base de datos. El fichero comienza con el nombre de tabla y tiene una extensión de .ARZ y .ARM. Un fichero .ARN puede aparecer durante operaciones de optimización.

El motor ARCHIVE soporta sólo INSERT y SELECT. Esto significa que no puede ejecutar DELETE, REPLACE, o update . Un SELECT realiza un escaneo de tabla completo. Los registros se comprimen al insertarse. OPTIMIZE TABLE puede usarse para comprimir la tabla.

El motor ARCHIVE usa bloqueo a nivel de registro.

14.8. El motor de almacenamiento CSV

El motor de almacenamiento CSV almacena datos en ficheros de texto usando valores separados por comas.

Para activar este motor de almacenamiento, use la opción --with-csv-storage-engine con configure al compilar MySQL.

Cuando crea una tabla CSV , el servidor crea un fichero de definición de tabla en el directorio de base de datos. El fichero comienza con el nombre de tabla y tienen una extensión .frm. El motor de almacenamiento crea un fichero de datos. Su nombre comienza con el nombre de tabla y tiene extensión .CSV. El fichero de datos es un fichero de texto. Cuando almacena datos en la tabla, el motor la guarda en el fichero de datos en formato CVS.

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) 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)
```

Si examina el fichero `test.CSV` en el directorio de base de datos creado al ejecutar los comandos precedentes, su contenido debe ser como este:

```
"1","record one"
"2","record two"
```

El motor de almacenamiento CSV no soporta indexación.

Capítulo 15. El motor de almacenamiento InnoDB

Tabla de contenidos

15.1. Panorámica de InnoDB

InnoDB dota a MySQL de un motor de almacenamiento transaccional (conforme a ACID) con capacidades de commit (confirmación), rollback (cancelación) y recuperación de fallas. InnoDB realiza bloqueos a nivel de fila y también porporciona funciones de lectura consistente sin bloqueo al estilo Oracle en sentencias SELECT. Estas características incrementan el rendimiento y la capacidad de gestionar múltiples usuarios simultáneos. No se necesita un bloqueo escalado en InnoDB porque los bloqueos a nivel de fila ocupan muy poco espacio. InnoDB también soporta restricciones FOREIGN KEY. En consultas SQL, aún dentro de la misma consulta, pueden incluirse libremente tablas del tipo InnoDB con tablas de otros tipos.

InnoDB se diseñó para obtener el máximo rendimiento al procesar grandes volúmenes de datos. Probablemente ningún otro motor de bases de datos relacionales en disco iguale su eficiencia en el uso de CPU.

A pesar de estar totalmente integrado con el servidor MySQL, el motor de almacenamiento InnoDB mantiene su propio pool de almacenamiento intermedio para tener un cache de datos e índices en la memoria principal. InnoDB almacena sus tablas e índices en un espacio de tablas, el cual puede consistir de varios ficheros (o particiones disco). Esto difiere de, por ejemplo, el motor MyISAM, donde cada tabla se almacena empleando ficheros separados. Las tablas InnoDB pueden ser de cualquier tamaño, aún en sistemas operativos donde el tamaño de los ficheros se limita a 2GB.

En MySQL 5.0, InnoDB viene incluido por defecto en las distribuciones binarias. El instalador Windows Essentials configura a InnoDB como el tipo de base de datos MySQL por defecto en Windows.

InnoDB se utiliza en muchos grandes sitios de bases de datos que necesitan alto rendimiento. El famoso sitio de noticias de Internet Slashdot.org corre sobre InnoDB. Mytrix, Inc. almacena más de 1TB de datos en InnoDB, y otros sitios manejan una carga promedio de 800 inserciones y actualizaciones por segundo en InnoDB.

InnoDB se publica bajo la misma licencia GNU GPL Versión 2 (de Junio de 1991) que MySQL. Para más información sobre el licenciamiento de MySQL, consulte http://www.mysql.com/company/legal/licensing/.

15.2. Información de contacto de InnoDB

Información de contacto para Innobase Oy, creador del motor InnoDB:

```
Sitio web: http://www.innodb.com/
Correo electrónico: <sales@innodb.com>
Teléfonos: +358-9-6969 3250 (oficina)
           +358-40-5617367 (móvil)

Innobase Oy Inc.
World Trade Center Helsinki
Aleksanterinkatu 17
P.O.Box 800
00101 Helsinki
Finland
```

15.3. Configuración de InnoDB

En MySQL 5.0, el motor de almacenamiento InnoDB está habilitado por defecto. Si no se desean emplear tablas InnoDB, puede agregarse la opción skip-innodb al fichero de opciones de MySQL.

Dos recursos basados en disco muy importantes que gestiona el motor de almacenamiento InnoDB son sus ficheros de datos de espacios de tablas y sus ficheros de registro (log).

Si no se especifican opciones de configuración para InnoDB, MySQL 5.0 crea en el directorio de datos de MySQL un fichero de datos de 10MB (autoextensible) llamado ibdata1 y dos ficheros de registro (log) de 5MB llamados ib_logfile0 y ib_logfile1.

Nota: InnoDB dota a MySQL de un motor de almacenamiento transaccional (conforme a ACID) con capacidades de commit (confirmación), rollback (cancelación) y recuperación de fallas. **Esto no es posible** si el sistema operativo subyacente y el hardware no funcionan como se requiere. Muchos sistemas operativos o subsistemas de disco podrían diferir o reordenar operaciones de escritura a fin de mejorar el rendimiento. En algunos sistemas operativos, la propia llamada del sistema (fsync()), que debería esperar hasta que todos los datos no guardados de un fichero se graben a disco, en realidad puede retornar antes de que los datos se guarden en las tablas de almacenamiento. Debido a esto, una caída del sistema operativo o un corte en el suministro eléctrico pueden destruir datos recientemente grabados, o, en el peor de los casos, corromper la base de datos debido a que las operaciones de escritura han sido reordenadas. Si la integridad de los datos es importante, se deberían llevar a cabo algunas pruebas que simulen caídas (“pull-the-plug”) e interrupciones súbitas, antes de comenzar el uso para producción. En Mac OS X 10.3 y posteriores, InnoDB emplea un método especial de volcado a fichero llamado fcntl(). Bajo Linux, es aconsejable **deshabilitar el write-back cache**.

En discos duros ATAPI, un comando como hdparm -W0 /dev/*hda* puede funcionar. **Hay que tener en cuenta que algunas unidades o controladores de disco podrían estar imposibilitados de desactivar el write-back cache.**

Nota: Para obtener un buen desempeño, se deberían proveer expresamente los parámetros de InnoDB como se explica en los siguientes ejemplos. Naturalmente, habrá que editar la configuración para acomodarla a los requerimientos del hardware en uso.

Para configurar los ficheros de espacio de tablas de InnoDB, debe utilizarse la opción innodb_data_file_path en la sección [mysqld] del fichero de opciones my.cnf. En Windows, se puede emplear en su lugar my.ini. El valor de innodb_data_file_path debería ser una lista de una o más especificaciones de ficheros. Si se incluirá más de un fichero de datos, habrá que separarlos con punto y coma (';'):

```
innodb_data_file_path=espec_fichero_datos1
[;espec_fichero_datos2]...
```

Por ejemplo, la siguiente es una configuración que creará explícitamente un espacio de tablas con las mismas características que el predeterminado:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

Esto configura un único fichero de 10MB llamado ibdata1 el cual es autoextensible. No se suministra la ubicación del fichero, por lo tanto, el directorio predeterminado es el directorio de datos de MySQL.

El tamaño del fichero se especifica empleando como sufijo las letras M o G para indicar unidades de MB o GB.

A continuación se configura un espacio de tablas que contiene un fichero de datos de tamaño fijo de 50MB llamado ibdata1 y un fichero autoextensible de 50MB llamado ibdata2, ambos en el directorio de datos:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

La sintaxis completa para especificar un fichero de datos incluye el nombre del fichero, su tamaño, y varios atributos opcionales:

```
nombre_de_fichero:tamaño_de_fichero[:autoextend[:max:tamaño_máximo_de_fichero]]
```

El atributo autoextend y aquellos que lo siguen sólo pueden emplearse con el último fichero en la línea de innodb_data_file_path.

Si se especifica la opción autoextend para el último fichero de datos, InnoDB incrementará el tamaño del fichero si se queda sin capacidad para el espacio de tablas. El incremento es de 8MB cada vez.

Si se agotara la capacidad del disco, podría desearse agregar otro fichero de datos en otro disco. Las instrucciones para reconfigurar un espacio de tablas existente se encuentran en Sección 15.7, “Añadir y suprimir registros y ficheros de datos InnoDB”.

InnoDB no detecta el tamaño máximo de fichero, por lo tanto, hay que ser cuidadoso en sistemas de ficheros donde el tamaño máximo sea de 2GB. Para especificar el tamaño máximo de un fichero autoextensible, se emplea el atributo max. La siguiente configuración le permite a ibdata1 crecer hasta un límite de 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB crea los ficheros de espacios de tablas en el directorio de datos de MySQL en forma predeterminada. Para especificar una ubicación expresamente, se emplea la opción innodb_data_home_dir. Por ejemplo, para crear dos ficheros llamados ibdata1 e ibdata2 pero creándolos en el directorio /ibdata, InnoDB se configura de este modo:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Nota: InnoDB no crea directorios, de modo que hay que estar seguro de que el directorio /ibdata existe antes de iniciar el servidor. Esto se aplica también a cualquier directorio de ficheros de registro (log) que se configure. Para crear los directorios necesarios se emplea el comando mkdir que existe en Unix y DOS.

InnoDB forma el directorio para cada fichero de datos concatenando el valor textual de innodb_data_home_dir con el nombre del fichero, agregando una barra o barra invertida entre ellos si se necesita. Si la opción innodb_data_home_dir no aparece en my.cnf, el valor predeterminado es el directorio ./, lo cual indica el directorio de datos de MySQL.

Si se especifica una cadena vacía en innodb_data_home_dir, se pueden especificar rutas absolutas para los ficheros de datos listados en el valor de innodb_data_file_path. El siguiente ejemplo es equivalente al anterior:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

Un ejemplo sencillo de my.cnf . Suponiendo que se posee un ordenador con 128MB de RAM y un disco duro, el siguiente ejemplo muestra posibles parámetros de configuración InnoDB en my.cnf o my.ini incluyendo el atributo autoextend.

Este ejemplo satisface las necesidades de la mayoría de los usuarios, tanto en Unix como en Windows, que no deseen distribuir los ficheros de datos InnoDB en varios discos. Crea un fichero de datos

autoextensible llamado ibdata1 y dos ficheros de registro (log) de InnoDB llamados ib_logfile0 y ib_logfile1 en el directorio de datos de MySQL. También, el fichero de registros archivados de InnoDB ib_arch_log_0000000000 que MySQL crea automáticamente, termina ubicado en el directorio de datos.

```
[mysqld]
# Las demas opciones del servidor MySQL pueden escribirse aquí
# ...
# Los ficheros de datos deben ser capaces de contener datos e índices
# Hay que asegurarse de tener suficiente espacio en disco.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Establecer el tamaño del buffer en un 50-80% de la memoria del ordenador
set-variable = innodb_buffer_pool_size=70M
set-variable = innodb_additional_mem_pool_size=10M
#
# Establecer el tamaño del fichero de registro (log) en un 25% del tamaño del
buffer
set-variable = innodb_log_file_size=20M
set-variable = innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Hay que asegurarse de que el servidor MySQL tiene los derechos de acceso apropiados para crear ficheros en el directorio de datos. Más generalmente, el servidor debe tener derechos de acceso a cualquier directorio donde necesite crear ficheros de datos o registro (logs).

Notar que los ficheros de datos deben ser menores de 2GB en algunos sistemas de ficheros. El tamaño combinado de los ficheros de registro debe ser menor de 4GB. El tamaño combinado de los ficheros de datos debe ser de por lo menos 10MB.

Cuando se crea un espacio de tablas InnoDB por primera vez, es mejor iniciar el servidor MySQL desde la línea de comandos. Entonces, InnoDB imprimirá en pantalla la información acerca de la creación de bases de datos, de forma que se podrá ver lo que está ocurriendo. Por ejemplo, en Windows, si mysqld-max se ubica en C:\mysql\bin, se puede iniciar de este modo:

```
C:\> C:\mysql\bin\mysqld-max --console
```

Si no se envía la salida del servidor a la pantalla, se puede ver el fichero de registro de errores del servidor para averiguar lo que InnoDB imprime durante el proceso de inicio.

Consulte Sección 15.5, “Crear el espacio de tablas InnoDB” para un ejemplo de cómo debería lucir la información mostrada por InnoDB.

¿Dónde deben especificarse las opciones en Windows? Las reglas para ficheros de opciones en Windows son las siguientes:

- Solo debe crearse el fichero my.cnf o my.ini, pero no los dos.
- El fichero my.cnf debe colocarse en el directorio raíz de la unidad C:.
- El fichero my.ini debería colocarse en el directorio WINDIR; por ejemplo, C:\WINDOWS o C:\WINNT. Puede utilizarse el comando SET en una ventana de consola para mostrar el valor de WINDIR:

  ```
  C:\> SET WINDIR
  windir=C:\WINNT
  ```

- Si el ordenador emplea un gestor de arranque donde la unidad C: no es la unidad de arranque, sólo es posible emplear el fichero my.ini.

- Si se instaló MySQL empleando los asistentes de instalación y configuración, el fichero `my.ini` se ubica en el directorio de instalación de MySQL. Consulte Sección 2.3.5.14, “Dónde está el fichero my.ini”.

¿Dónde deben especificarse las opciones en Unix? En Unix, `mysqld` lee las opciones en los siguientes ficheros, si existen, en el siguiente orden:

- `/etc/my.cnf`

  Opciones globales.

- `$MYSQL_HOME/my.cnf`

  Opciones específicas del servidor.

- `defaults-extra-file`

  El fichero especificado con la opción `--defaults-extra-file`.

- `~/.my.cnf`

  Opciones específicas del usuario.

`MYSQL_HOME` representa una variable de entorno la cual contiene la ruta al directorio que hospeda al fichero específico de servidor `my.cnf`.

Si se desea estar seguro de que `mysqld` lee sus opciones únicamente desde un fichero determinado, se puede emplear `--defaults-option` como la primera opción en la línea de comandos cuando se inicia el servidor:

```
mysqld --defaults-file=ruta_a_my_cnf
```

Un ejemplo avanzado de `my.cnf` . Suponiendo que se posee un ordenador Linux con 2GB de RAM y tres discos duros de 60GB (en los directorios `/`, `/dr2` y `/dr3`). El siguiente ejemplo muestra posibles parámetros de configuración `InnoDB` en `my.cnf`.

```
[mysqld]
# Las demas opciones del servidor MySQL pueden escribirse aquí
# ...
innodb_data_home_dir =
#
# Los ficheros de datos deben ser capaces de contener datos e índices
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:2000M:autoextend
#
# Establecer el tamaño del buffer en un 50-80% de la memoria del ordenador,
# pero hay que asegurarse que en Linux x86 el uso total de memoria es < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
innodb_log_files_in_group = 2
#
# Establecer el tamaño del fichero de registro (log) en un 25% del tamaño del
buffer
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
```

```
# Quitar marca de comentario a las siguientes lineas si se desea usarlas
#innodb_thread_concurrency=5
```

Nótese que el ejemplo ubica los dos ficheros de datos en discos diferentes. InnoDB llena el espacio de tablas comenzando por el primer fichero de datos. En algunos casos, el rendimiento de la base de datos mejorará si no se colocan todos los datos en el mismo disco físico. Colocar los ficheros de registro (log) en un disco diferente a los datos, a menudo es beneficioso para el rendimiento. También se pueden utilizar dispositivos en bruto (raw devices) como ficheros de datos InnoDB, lo cual mejorará la velocidad de E/S. Consulte Sección 15.14.2, “Usar dispositivos en bruto (raw devices) para espacios de tablas”.

Advertencia: En GNU/Linux x86 de 32 bits, se debe tener cuidado con no establecer el uso de memoria en un número demasiado alto. glibc le puede permitir al heap de proceso que crezca por sobre la pila de los subprocesos, lo cual hará caer el servidor. Es arriesgado que el resultado del siguiente cálculo exceda los 2GB:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Cada hilo emplea una pila (a menudo de 2MB, pero de solamente 256KB en los binarios de MySQL AB) y en el peor caso también empleará una cantidad de memoria adicional igual a `sort_buffer_size + read_buffer_size`.

Compilando MySQL por sí mismo, el usuario puede emplear hasta 64GB de memoria física en Windows de 32 bits. Consulte la descripción de `innodb_buffer_pool_awe_mem_mb` en Sección 15.4, “Opciones de arranque de InnoDB”.

¿Cómo deben ajustarse otros parámetro del servidor `mysqld`? Los siguientes son valores típicos adecuados para la mayoría de los usuarios:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Establecer key_buffer a un 5 - 50% de la RAM., dependiendo de cuánto se usen
# tablas MyISAM, pero manteniendo key_buffer_size + InnoDB
# buffer pool size < 80% de la RAM
key_buffer_size=value
```

15.4. Opciones de arranque de InnoDB

Esta sección describe las opciones de servidor relacionadas con InnoDB. En MySQL 5.0, todas son especificadas con la forma `--opt_name=value` en la línea de comandos o en ficheros de opciones.

- `innodb_additional_mem_pool_size`

  El tamaño del pool de memoria que InnoDB utiliza para almacenar información del diccionario de datos y otras estructuras de datos internas. Mientras más tablas se tengan en la aplicación, mayor será este tamaño. Si InnoDB se queda sin memoria en este pool, comenzará a tomar memoria del sistema operativo, y dejará mensajes de advertencia en el log de errores de MySQL. El valor por defecto es 1MB.

- `innodb_autoextend_increment`

El tamaño a incrementar (en megabytes) cuando se extiende el tamaño de un espacio de tablas autoextensible, luego de llenarse. El valor por defecto es 8. Esta opción puede cambiarse en tiempo de ejecución como una variable de sistema global.

- innodb_buffer_pool_awe_mem_mb

  El tamaño (en MB) del pool de buffer, si está ubicado en la memoria AWE en Windows de 32 bits, y sólo relevante en este tipo de sistemas operativos. Si el sistema operativo Windows de 32 bits en uso soporta más de 4GB de memoria, usualmente llamado “Address Windowing Extensions”, se puede ubicar el pool del buffer de InnoDB dentro de la memoria física AWE utilizando este parámetro. El máximo valor posible es de 64000. Si se especifica este parámetro, innodb_buffer_pool_size es la ventana en el espacio de direcciones de 32 bits de mysqld donde InnoDB direcciona la memoria AWE. Un valor adecuado para innodb_buffer_pool_size son 500MB.

- innodb_buffer_pool_size

  El tamaño del buffer de memoria que InnoDB emplea para el almacenamiento intermedio de los datos e índices de sus tablas. Mientras más grande sea este valor, menores operaciones de E/S en disco serán necesarias para acceder a los datos de las tablas. En un servidor de bases de datos dedicado, se puede establecer este valor en hasta el 80% de la memoria física del ordenador. Sin embargo, no debe establecerse en un valor demasiado grande porque la pugna por la memoria física podría causar que el sistema operativo comience a paginar.

- innodb_checksums

  InnoDB emplea validación por sumas de verificación (checksums) en todas las páginas leídas desde el disco, para asegurar una tolerancia extra contra fallas frente a hardware averiado o ficheros corruptos. Sin embargo, bajo ciertas circunstancias inusuales (por ejemplo al ejecutar pruebas de rendimiento) esta característica extra de seguridad es innecesaria. En tales casos, esta opción (que está habilitada por defecto) puede deshabilitarse con --skip-innodb-checksums. Esta opción fue agregada en MySQL 5.0.3.

- innodb_data_file_path

  Las rutas a los ficheros individuales de datos y sus tamaños. La ruta de directorio completa a cada fichero de datos se obtiene concatenando innodb_data_home_dir con cada ruta especificada aquí. Los tamaños de fichero se especifican en megabytes o gigabytes (1024MB) agregando M o G al valor que representa el tamaño. La sumatoria de los tamaños de fichero debe ser de al menos 10MB. En algunos sistemas operativos, los ficheros deben tener menos de 2GB. Si no se indica innodb_data_file_path, el comportamiento predeterminado de inicio es crear un único fichero autoextensible de 10MB llamado ibdata1. En aquellos sistemas operativos que soporten ficheros grandes, se puede establecer el tamaño de fichero en más de 4GB. También pueden utilizarse como ficheros de datos particiones de dispositivos en bruto. Consulte Sección 15.14.2, “Usar dispositivos en bruto (raw devices) para espacios de tablas”.

- innodb_data_home_dir

  La porción común de la ruta de directorio para todos los ficheros de datos InnoDB. Si este valor no se establece, por defecto será el directorio de datos de MySQL. También puede especificarse como una cadena vacía, en cuyo caso se podrán utilizar rutas absolutas en innodb_data_file_path.

- innodb_doublewrite

  Por defecto, InnoDB almacena todos los datos dos veces, la primera en el buffer de escritura doble (o doublewrite), y luego a los ficheros de datos reales. Esta opción puede emplearse para desactivar

dicha funcionalidad. Al igual que innodb_checksums, esta opción está habilitada por defecto; puede desactivarse con --skip-innodb-doublewrite en pruebas de rendimiento o casos en que el máximo desempeño prevalezca sobre la preocupacion por la integridad de los datos o las posibles fallas. Esta opción se agregó en MySQL 5.0.3.

- innodb_fast_shutdown

  Si se establece a 0, InnoDB efectúa una descarga completa y vuelca los buffers de inserción antes de llevar a cabo el cierre del servidor. Estas operaciones pueden tomar minutos o incluso horas en casos extremos. Si se establece en 1, InnoDB pasa por alto estas operaciones al cierre. El valor por defecto es 1. Si se establece en 2 (opción que está disponible desde MySQL 5.0.5, excepto en Netware), InnoDB simplemente vuelca a disco sus registros (logs) y se cierra en frío, como si hubiera ocurrido una caída; ninguna transacción confirmada se perderá, pero en el siguiente inicio se ejecutará una recuperación ante caídas.

- innodb_file_io_threads

  El número de subprocesos (threads) de E/S de fichero en InnoDB. Normalmente esto debería ser dejado en el valor predeterminado de 4, pero la E/S de disco en Windows puede beneficiarse con un número mayor. En Unix, incrementar el número no tiene efecto; InnoDB siempre utiliza el valor predeterminado.

- innodb_file_per_table

  Esta opción provoca que InnoDB cree cada nueva tabla utilizando su propio fichero .ibd para almacenar datos e índices, en lugar de colocarlo en el espacio de tablas compartidas. Consulte Sección 15.6.6, “Usar un espacio de tablas para cada tabla”.

- innodb_flush_log_at_trx_commit

  Cuando innodb_flush_log_at_trx_commit se establece en 0, una vez por segundo el buffer de registros (log buffer) se graba en el fichero de registro y se vuelca a disco, pero no se hace nada al confirmar una transacción. Cuando este valor es 1 (predeterminado), cada vez que se confirma una transacción el buffer de registros (log buffer) se graba en el fichero de registro y se vuelca a disco Cuando se establece en 2, el buffer de registros (log buffer) se graba en el fichero de registro, pero no se vuelca a disco. Sin embargo, el volcado a disco del fichero de registro se produce una vez por segundo también cuando vale 2. Se debe tener en cuenta que el volcado una vez por segundo no está 100% garantizado que ocurra en cada segundo, debido a cuestiones de programación (scheduling) de procesos. Se puede alcanzar un mayor rendimiento estableciendo un valor diferente de 1, pero en caso de caída se puede perder un segundo de transacciones. Si se establece el valor en 0, cualquier caída en un proceso de mysqld puede borrar el último segundo de transacciones. Si se establece el valor en 2, entonces únicamente una caída del sistema operativo o una interrupción de energía pueden borrar el último segundo de transacciones. Hay que notar que muchos sistemas operativos y algunos tipos de discos puedne ser engañosos en las operaciones de volcado a disco. Podrían indicarle a mysqld que el volcado ha tenido lugar, aunque no sea así. En tal caso la estabilidad de las transacciones no está garantizada ni aún con el valor 1, y en el peor de los casos una interrupción de energía puede incluso corromper la base de datos InnoDB. Utilizar un caché de disco apoyado por baterías en el controlador de disco SCSI o en el propio disco, acelera los volcados a disco, y hace más segura la operación. También puede intentarse con el comando de Unix hdparm, el cual deshabilita el almacenamiento en caches de hardware de las operaciones de escritura a disco, o utilizar algún otro comando específico del fabricante del hardware. El valor por defecto de esta opción es 1

- innodb_flush_method

  Esta opción solamente es relevante en sistemas Unix. Si se establece en fdatasync (el valor predeterminado), InnoDB utiliza fsync() para volcar tanto los ficheros de datos como de registro (log).

Si se establece en O_DSYNC, InnoDB emplea O_SYNC para abrir y volcar los ficheros de registro, pero utiliza fsync() para volcar los ficheros de datos. Si se especifica O_DIRECT (disponible en algunas versiones de GNU/Linux), InnoDB utiliza O_DIRECT para abrir los ficheros de datos, y fsync() para volcar tanto los ficheros de datos como de registro. Nótese que InnoDB emplea fsync() en lugar de fdatasync(), y no emplea O_DSYNC por defecto porque han ocurrido problemas con éste en muchas variedades de Unix.

- innodb_force_recovery

  Advertencia: Esta opción debería ser definida solamente en una situación de emergencia cuando se desean volcar las tablas desde una base de datos corrupta. Los posibles valores van de 1 a 6. Los significados de estos valores se describen en Sección 15.8.1, “Forzar una recuperación”. Como una medida de seguridad, InnoDB impide que un usuario modifique datos cuando esta opción tiene un valor mayor a 0.

- innodb_lock_wait_timeout

  El límite de tiempo, en segundos, que una transacción InnoDB puede esperar por un bloqueo antes de ser cancelada. InnoDB automáticamente detecta bloqueos mutuos (deadlocks) en su propia tabla de bloqueos, y cancela la transacción. InnoDB detecta los bloqueos por el uso de la sentencia LOCK TABLES. El valor predeterminado es de 50 segundos.

  Para conseguir la mayor estabilidad y consistencia posibles en una configuración de replicación, se debería utilizar innodb_flush_logs_at_trx_commit=1, sync-binlog=1, y innodb_safe_binlog en el fichero my.cnf principal.

- innodb_locks_unsafe_for_binlog

  Esta opción desactiva el bloqueo de la clave siguiente en búsquedas y exploraciones de índices InnoDB. El valor por defecto de esta opción es falso.

  Normalmente, InnoDB utiliza un algoritmo denominado *bloqueo de clave siguiente (next-key)*. InnoDB efectúa un bloqueo a nivel de fila de tal forma que cuando busca o explora el índice de una tabla, establece bloqueos compartidos o exclusivos en cualquier registro de índice que encuentre. El bloqueo que InnoDB establece en registros de índice también afecta al “vacío” que precede a ese registro. Si un usuario tiene un bloqueo compartido o exclusivo sobre el registro *R* en un índice, otro usuario no puede insertar un nuevo registro de índice inmediatamente antes de *R* en el orden del índice. Esta opción provoca que InnoDB no utilice el bloqueo de clave siguiente en búsquedas o exploraciones de índices. El bloqueo de clave siguiente es todavía utilizado para asegurar las restricciones de claves foráneas y la verificación de claves duplicadas. Nótese que el uso de esta opción puede provocar problemas secundarios: suponiendo que se deseen leer y bloquear todos los registros hijos de la tabla child que tengan un identificador mayor a 100, junto al posterior intento de actualizar algunas columnas en las filas seleccionadas:

  ```
  SELECT * FROM child WHERE id > 100 FOR UPDATE;
  ```

  Supóngase que hay un índice sobre la columna id. La consulta explora aquel índice comenzando por el primer registro en que id sea mayor que 100. Si el bloqueo efectuado sobre los registros del índice no bloquea las inserciones realizadas en los espacios vacíos, en la tabla se insertará un nuevo registro. Si se ejecuta el mismo SELECT dentro de la misma transacción, se verá un nuevo registro en el conjunto de resultados devuelto por la consulta. Esto también significa que si se agregan nuevos elementos a la base de datos, InnoDB no garantiza la serialización; sin embargo, los conflictos de serialización aún están garantizados. Por lo tanto, si esta opción se utiliza, InnoDB garantiza como mucho el nivel de aislamiento READ COMMITTED.

A partir de MySQL 5.0.2 esta opción es aún más insegura. InnoDB en un UPDATE o DELETE solamente bloquea los registros que se actualizan o borran. Esto reduce notablemente la probabilidad de bloqueos mutuos (deadlocks), pero aún pueden ocurrir. Nótese que esta opción todavía no le permite a operaciones como UPDATE predominar sobre otras operaciones similares (como otro UPDATE) aún en el caso en que actúen sobre registros diferentes. Considérese lo siguiente: example:

```
CREATE TABLE A(A INT NOT NULL, B INT);
INSERT INTO A VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

Si una conexión realiza una consulta:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 5 WHERE B = 3;
```

y la otra conexión ejecuta otra consulta a continuación de la primera:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 4 WHERE B = 2;
```

La consulta dos tendrá que esperar la confirmación o la cancelación de la consulta uno, porque ésta tiene un bloqueo exclusivo en el registro (2,3), y la consulta dos, mientras explora los registros, también intenta colocar un bloqueo exclusivo en la misma fila, cosa que no puede hacer. Esto se debe a que la consulta dos primero establece el bloqueo sobre un registro y luego determina si el registro pertenece al conjunto de resultados, y si no es así libera el bloqueo innecesario, cuando se emplea la opción innodb_locks_unsafe_for_binlog.

Por lo tanto, la consulta uno se ejecuta de este modo:

```
x-lock(1,2)
unlock(1,2)
x-lock(2,3)
update(2,3) to (2,5)
x-lock(3,2)
unlock(3,2)
x-lock(4,3)
update(4,3) to (4,5)
x-lock(5,2)
unlock(5,2)
```

entonces la consulta dos se ejecuta así:

```
x-lock(1,2)
update(1,2) to (1,4)
x-lock(2,3) - wait for query one to commit or rollback
```

- innodb_log_arch_dir

  El directorio donde los ficheros de registro (logs) terminados se archivarán si se utiliza el archivo de ficheros de registro. Si se utiliza, el valor de este parámetro debería ser el mismo que innodb_log_group_home_dir. Sin embargo, no es requerido.

- innodb_log_archive

Este valor generalmente debería establecerse a 0. Debido a que la recuperación a partir de una copia de respaldo es realizada por MySQL empleando sus propios ficheros de registro (log), en general no hay necesidad de archivar los ficheros de registro de InnoDB. El valor predeterminado para esta opción es 0.

- innodb_log_buffer_size

  El tamaño del buffer que InnoDB emplea para escribir los ficheros de registro (logs) en disco. Los valores razonables se encuentran entre 1MB y 8MB. El valor predeterminado es 1MB. Un buffer de fichero de registro grande le permite a las transacciones extensas ejecutarse sin necesidad de guardar el fichero de registro en disco antes de que la transacción se confirme. Por lo tanto, si se manejan grandes transacciones, incrementar el tamaño del buffer de ficheros de registro ahorra operaciones de E/S en disco.

- innodb_log_file_size

  El tamaño de cada fichero de registro (log) en un grupo de ficheros de registro. El tamaño combinado de estos ficheros debe ser inferior a 4GB en ordenadores de 32 bits. El valor predeterminado es de 5MB. El rango de valores razonables va desde 1MB hasta la 1/*N* parte del tamaño del pool de buffer, donde *N* es la cantidad de ficheros de registro en el grupo. Mientras mayor es el valor, menor es la cantidad de guardado de puntos de verificación que se necesitan en el pool de buffer, ahorrando operaciones de E/S en disco. Pero tener ficheros de registro más grandes también significa que la recuperación es más lenta en caso de caídas.

- innodb_log_files_in_group

  En un grupo de ficheros de registro (logs), es la cantidad de ficheros que contiene. InnoDB escribe en los ficheros siguiendo una forma circular. El valor predeterminado es 2 (recomendado).

- innodb_log_group_home_dir

  La ruta de directorio a los ficheros de registro (log) de InnoDB. Debe tener el mismo valor que innodb_log_arch_dir. Si no se especifican parámetros de ficheros de registro InnoDB, la acción predeterminada es crear dos ficheros de 5MB llamados ib_logfile0 y ib_logfile1 en el directorio de datos de MySQL.

- innodb_max_dirty_pages_pct

  Un entero en el rango de 0 a 100. El valor por defecto es 90. El subproceso (thread) principal en InnoDB intenta volcar páginas desde el pool de buffer de modo que a lo sumo este porcentaje de las páginas aún sin volcar sea volcado en un momento determinado. Si se tiene el privilegio SUPER, este porcentaje pude cambiarse mientras el servidor está en ejecución:

  ```
  SET GLOBAL innodb_max_dirty_pages_pct = value;
  ```

- innodb_max_purge_lag

  Esta opción controla la forma de retrasar las operaciones INSERT, UPDATE y DELETE cuando las operaciones de descarga (ver Sección 15.12, “Implementación de multiversión”) están sufiendo demoras. TEl valor por defecto de este parámetro es cero, lo que significa que no se retrasarán. Esta opción puede modificarse en tiempo de ejecución como una variable global de sistema.

  El sistema de transacciones de InnoDB mantiene una lista de transacciones que tienen entradas en los índices marcadas para ser eliminadas por operaciones UPDATE o DELETE. Se deja que la longitud de esta lista sea *purge_lag*. Cuando *purge_lag* excede a innodb_max_purge_lag, cada operación

de INSERT, UPDATE y DELETE se retrasa durante ((*purge_lag*/innodb_max_purge_lag)*10)-5 milisegundos. El retraso se computa en el comienzo de un lote de depuración, cada diez segundos. Las operaciones no se retrasan si no puede ejecutarse la depuración debido a una vista de lectura consistente (consistent read) anterior que contenga los registros a ser depurados.

Un escenario típico para una carga de trabajo problemática podría ser 1 millón, asumiendo que las transacciones son pequeñas, sólo 100 bytes de tamaño, y se pueden permitir 100 MB de registros sin descargar en las tablas.

- innodb_mirrored_log_groups

  El número de copias idénticas de grupos de ficheros de registro que se mantienen para la base de datos. Actualmente debería establecerse en 1.

- innodb_open_files

  Esta opción sólo es relevante si se emplean múltiples espacios de tablas en InnoDB. Especifica el número máximo de ficheros .ibd que InnoDB puede mantener abiertos al mismo tiempo. El mínimo es 10. El valor predeterminado es 300.

  Los descriptores de fichero empleados para ficheros .ibd son únicamente para InnoDB. Son independientes de los especificados por la opción de servidor --open-files-limit, y no afectan la operación del caché de tablas.

- innodb_safe_binlog

  Contribuye a asegurar la consistencia entre el contenido de las tablas InnoDB y el registro binario (binary log). Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

- innodb_status_file

  Esta opción provoca que InnoDB cree un fichero <*datadir*>/innodb_status.<*pid*> para la salida periódica de SHOW INNODB STATUS. Disponible desde MySQL 4.0.21.

- innodb_table_locks

  InnoDB respeta lo establecido por LOCK TABLES, y MySQL no retorna desde un LOCK TABLE .. WRITE hasta que todos los otros flujos (threads) han levantado sus bloqueos a la tabla. El valor por defecto es 1, lo cual significa que LOCK TABLES causará que InnoDB bloquee una tabla internamente. En aplicaciones que emplean AUTOCOMMIT=1, los bloqueos internos de tabla de InnoDB pueden originar bloqueos mutuos (deadlocks). Se puede establecer innodb_table_locks=0 en my.cnf (o my.ini en Windows) para eliminar ese problema.

- innodb_thread_concurrency

  InnoDB intenta mantener el número de flujos (threads) del sistema operativo que concurren dentro de InnoDB en un valor menor o igual al límite especificado por este parámetro. Antes de MySQL 5.0.8, el valor por defecto es 8. Si se tienen dificultades de rendimiento, y SHOW INNODB STATUS indica que hay muchos subprocesos esperando por semáforos, se podrían tener subprocesos pugnando por recursos, y se debería establecer este parámetro en un número mayor o menor. Si se posee un ordenador con varios procesadores y discos, se puede intentar aumentar el valor para hacer mejor uso de los recursos del ordenador. Un valor recomendado es la suma del número de procesadores y discos que tiene el sistema. Un valor de 500 o mayor deshabilitará la verificación de concurrencia. A partir de MySQL 5.0.8, el valor por defecto es 20, y la verificación de concurrencia se deshabilita si se establece en 20 o más.

- innodb_status_file

  Esta opción provoca que InnoDB cree un fichero <datadir>/innodb_status.<pid> para almacenar periódicamente la salida de SHOW INNODB STATUS.

15.5. Crear el espacio de tablas InnoDB

Suponiendo que se ha instalado MySQL y se editó el fichero de opciones para que contenga los parámetros de InnoDB necesarios, antes de iniciar MySQL se debería verificar que los directorios indicados para los ficheros de datos y de registro (log) InnoDB existen y que el servidor MySQL tiene permisos de acceso a dichos directorios. InnoDB no puede crear directorios, solamente ficheros. Hay que verificar también que se tiene suficiente espacio en disco para los ficheros de datos y de registro.

Cuando se crea una base de datos InnoDB, es mejor ejecutar el servidor MySQL mysqld desde la línea de comandos, no desde el envoltorio mysqld_safe o como un servicio de Windows. Cuando se lo ejecuta desde la línea de comandos, se puede ver lo que mysqld imprime y qué está ocurriendo. En Unix, simplemente debe invocarse mysqld. En Windows, hay que usar la opción --console.

Cuando se inicia el servidor MySQL luego de la configuración inicial de InnoDB en el fichero de opciones, InnoDB crea los ficheros de datos y de registro e imprime algo como lo siguiente:

```
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
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
```

Se ha creado una nueva base de datos InnoDB. Se puede conectar al servidor MySQL con los programas cliente acostumbrados, como mysql. Cuando se detiene el servidor MySQL, con mysqladmin shutdown, la salida es como la siguiente:

```
010321 18:33:34  mysqld: Normal shutdown
010321 18:33:34  mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

Se puede mirar en los directorios de ficheros de datos y registro y se verán los ficheros creados. El directorio de registro (log) también contiene un pequeño fichero llamado ib_arch_log_0000000000. Ese fichero resulta de la creación de la base de datos, luego de lo cual InnoDB desactivó el guardado de registros (log). Cuando MySQL inicia de nuevo, los ficheros de datos y de registro ya han sido creados, por lo que la salida es más breve:

```
InnoDB: Started
mysqld: ready for connections
```

Es posible agregar la opción innodb_file_per_table a my.cnf, y hacer que InnoDB almacene cada tabla en su propio fichero .ibd en un directorio de bases de datos de MySQL. Consulte Sección 15.6.6, “Usar un espacio de tablas para cada tabla”.

15.5.1. Resolución de problemas en la inicialización de InnoDB

Si InnoDB imprime un error de sistema operativo en una operación de ficheros, generalmente el problema es uno de los siguientes:

- No se creó el directorio para los ficheros de datos o de registros (log) de InnoDB.
- mysqld no tiene los permisos de acceso para crear ficheros en aquellos directorios.
- mysqld no puede leer el fichero de opciones my.cnf o my.ini adecuado, y por lo tanto no ve las opciones especificadas.
- El disco está lleno o se excedió la cuota de disco.
- Se ha creado un subdirectorio que tiene el mismo nombre que uno de los ficheros de datos especificados.
- Hay un error de sintaxis en innodb_data_home_dir o innodb_data_file_path.

Si algo va mal durante el intento de InnoDB de inicializar el espacio de tablas o los ficheros de registro, se deberán borrar todos los ficheros creados por InnoDB. Esto comprende todos los ficheros ibdata y todos los ib_logfile. En caso de haber creado alguna tabla InnoDB, habrá que borrar del directorio de datos de MySQL los correspondientes ficheros .frm de estas tablas (y cualquier fichero .ibd si se están empleando múltiples espacios de tablas). Entonces puede intentarse nuevamente la creación de la base de datos InnoDB. Es mejor iniciar el servidor MySQL desde una línea de comandos de modo que pueda verse lo que ocurre.

15.6. Crear tablas InnoDB

Suponiendo que se ha iniciado el cliente MySQL con el comando mysql test, para crear una tabla InnoDB se debe especificar la opción ENGINE = InnoDB o TYPE = InnoDB en la sentencia SQL de creación de tabla:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) TYPE=InnoDB;
```

La sentencia SQL crea una tabla y un índice en la columna a en el espacio de tablas InnoDB que consiste en los ficheros de datos especificados en my.cnf. Adicionalmente, MySQL crea un fichero customers.frm en el directorio test debajo del directorio de bases de datos de MySQL. Internamente, InnoDB agrega a su propio diccionario de datos una entrada para la tabla 'test/customers'. Esto significa que puede crearse una tabla con el mismo nombre customers en otra base de datos, y los nombres de las tablas no entrarán en conflicto dentro de InnoDB.

Se puede consultar la cantidad de espacio libre en el espacio de tablas InnoDB dirigiendo una sentencia SHOW TABLE STATUS para cualquier tabla InnoDB. La cantidad de espacio libre en el espacio de tablas aparece en la sección Comment en la salida de SHOW TABLE STATUS. Un ejemplo:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

Nótese que las estadísticas que SHOW muestra acerca de las tablas InnoDB son solamente aproximadas. Se utilizan en la optimización SQL. No obstante, los tamaños en bytes reservados para las tablas e índices son exactos.

15.6.1. Cómo utilizar transacciones en InnoDB con distintas APIs

En forma predeterminada, cada cliente se que conecta al servidor MySQL comienza con el modo de autocommit habilitado, lo cual automáticamente confirma (commit) cada sentencia SQL ejecutada. Para utilizar transacciones de múltiples sentencias se puede deshabilitar el modo autocommit con la sentencia SQL SET AUTOCOMMIT = 0 y emplear COMMIT y ROLLBACK para confirmar o cancelar la transacción. Si se desea dejar activado autocommit, se pueden encerrar las transacciones entre las sentencias START TRANSACTION y COMMIT o ROLLBACK. El siguiente ejemplo muestra dos transacciones. La primera se confirma, la segunda se cancela.

```
shell> mysql test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.50-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
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;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+------+--------+
| A    | B      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

En APIs como PHP, Perl DBI/DBD, JDBC, ODBC, o la interface de llamadas C estándar de MySQL, se pueden enviar sentencias de control de transacciones como COMMIT al servidor MySQL en forma de cadenas, igual que otras sentencias SQL como SELECT o INSERT. Algunas APIs también ofrecen funciones o métodos especiales para confirmación y cancelación de transacciones.

15.6.2. Pasar tablas MyISAM a InnoDB

Importante: No se deben convertir las tablas del sistema en la base de datos mysql (por ejemplo, user o host) al tipo InnoDB. Las tablas del sistema siempre deben ser del tipo MyISAM.

Si se desea que todas las tablas que no sean de sistema se creen como tablas InnoDB, simplemente debe agregarse la línea default-table-type=innodb a la sección [mysqld] del fichero my.cnf o my.ini.

InnoDB no posee una optimización especial para la creación separada de índices en la misma forma que la tiene el motor de almacenamiento MyISAM. Por lo tanto, no hay beneficio en exportar e importar la tabla y crear los índices posteriormente. La manera más rápida de cambiar una tabla al motor InnoDB

es hacer las inserciones directamente en una tabla InnoDB. Es decir, utilizar ALTER TABLE ... ENGINE=INNODB, o crear una tabla InnoDB vacía con idénticas definiciones e insertar las filas con INSERT INTO ... SELECT * FROM ....

Si se tienen restricciones UNIQUE sobre claves secundarias, se puede acelerar la importación de una tabla desactivando temporalmente la verificación de unicidad durante la sesión de importación: SET UNIQUE_CHECKS=0;. Para tablas grandes, esto ahorra gran cantidad de operaciones de E/S en disco, debido a que InnoDB puede emplear su buffer de inserciones para grabar registros de índices secundarios en lote.

Para obtener un mejor control sobre el proceso de inserción, podría ser mejor llenar la tablas grandes por partes:

```
INSERT INTO nuevatabla SELECT * FROM viejatabla
   WHERE clave > valor1 AND clave <= valor2;
```

Luego de que todos los registros se hayan insertado, se pueden renombrar las tablas.

Durante la conversión de tablas grandes, se puede reducir la cantidad de operaciones de E/S en disco incrementando el tamaño del pool de buffer de InnoDB. No debe usarse más del 80% de la memoria física. También pueden aumentarse los tamaños de los ficheros de registro (log) de InnoDB.

Hay que asegurarse de no llenar completamente el espacio de tablas: las tablas InnoDB necesitan mucho más espacio que las tablas MyISAM. Si una sentencia ALTER TABLE se queda sin espacio, realizará una cancelación (rollback), y esto puede tomar horas si lo hace sobre el disco. Para las inserciones, InnoDB emplea el buffer de inserción para combinar en lotes los registros secundarios de índices con los índices. Esto ahorra gran cantidad de operaciones de E/S en disco. Durante la cancelación no se emplea ese mecanismo, de modo que puede llevar más de 30 veces el tiempo insumido por la inserción.

Si se produjera una de estas cancelaciones fuera de control, sino se tienen datos valiosos en la base de datos, puede ser preferible matar el proceso de la base de datos en lugar de esperar que se completen millones de operaciones de E/S en disco. Para el procedimiento completo, consulte Sección 15.8.1, “Forzar una recuperación”.

15.6.3. Cómo funciona una columna AUTO_INCREMENT en InnoDB

Si se especifica que una columna de una tabla es AUTO_INCREMENT, la entrada para la tabla InnoDB en el diccionario de datos contiene un contador especial llamado "contador de auto incremento", que se utiliza para asignar nuevos valores a la columna. El contador de auto incremento se almacena sólo en la memoria principal, no en disco.

InnoDB utiliza el siguiente algoritmo para inicializar el contador de auto incremento para una tabla T que contiene una columna AUTO_INCREMENT llamada ai_col: Luego de iniciarse el servidor, cuando un usuario realiza por primera vez una inserción en una tabla T, InnoDB ejecuta el equivalente de esta sentencia:

```
SELECT MAX(ai_col) FROM T FOR UPDATE;
```

El valor retornado por la sentencia se incrementa en uno y se asigna a la columna, y al contador de auto incremento de la tabla. Si la tabla está vacía, se asigna el valor 1. Si el contador aún no se ha inicializado y se ejecuta una sentencia SHOW TABLE STATUS que muestre su salida para la tabla T, el contador se inicializa (pero no se incrementa) y se almacena para usos posteriores. Nótese que en esta inicialización se realiza una lectura normal con bloqueo exclusivo y el bloqueo permanece hasta el final de la transacción.

InnoDB sigue el mismo procedimiento para inicializar el contador de auto incremento de una tabla recientemente creada.

Nótese que si durante un INSERT el usuario especifica un valor NULL o 0 para una columna AUTO_INCREMENT, InnoDB trata a la columna como si no se hubiera especificado un valor y genera un nuevo valor para ella.

Luego de que el contador de auto incremento ha sido inicializado, si un usuario inserta una fila que explícitamente indica para la columna auto incremental un valor mayor que el valor actual del contador, éste se establece al valor actual de la columna. Si no se indica un valor, InnoDB incrementa el valor del contador en uno y lo asigna a la columna.

Al acceder al contador de auto incremento, InnoDB emplea un nivel de bloqueo de tabla especial AUTO-INC, que mantiene hasta el final de la sentencia SQL actual y no hasta el final de la transacción. Esta estrategia de bloqueo especial fue introducida para mejorar la concurrencia de inserciones en una tabla que contiene una columna AUTO_INCREMENT. Dos transacciones no pueden tener el bloqueo AUTO-INC simultáneamente en la misma tabla.

Nótese que pueden observarse valores faltantes en la secuencia de valores asignados a la columna AUTO_INCREMENT si se cancelan transacciones que ya han obtenido números desde el contador.

El comportamiento del mecanismo de auto incremento no se encuentra definido si un usuario asigna un valor negativo a la columna o si el valor se vuelve mayor que el entero más grande que puede almacenarse en el tipo de entero especificado.

A partir de MySQL 5.0.3, InnoDB soporta la opción AUTO_INCREMENT = *n* en sentencias CREATE TABLE y ALTER TABLE, para establecer inicialmente o modificar el valor actual del contador. El efecto de esta acción es cancelado al reiniciar el servidor, por las razones tratadas anteriormente en esta sección.

15.6.4. Restricciones (constraints) FOREIGN KEY

InnoDB también soporta restricciones de claves foráneas. La sintaxis para definir una restricción de clave foránea en InnoDB es así:

```
[CONSTRAINT símbolo] FOREIGN KEY [id] (nombre_índice, ...)
    REFERENCES nombre_de_tabla (nombre_índice, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Las definiciones de claves foráneas están sujetas a las siguientes condiciones:

- Ambas tablas deben ser InnoDB y no deben ser tablas temporales.
- En la tabla que hace referencia, debe haber un índice donde las columnas de clave extranjera estén listadas en *primer* lugar, en el mismo orden.
- En la tabla referenciada, debe haber un índice donde las columnas referenciadas se listen en *primer* lugar, en el mismo orden. En MySQL/InnoDB 5.0, tal índice se creará automáticamente en la tabla referenciada si no existe aún.
- No están soportados los índices prefijados en columnas de claves foráneas. Una consecuencia de esto es que las columnas BLOB y TEXT no pueden incluirse en una clave foránea, porque los índices sobre dichas columnas siempre deben incluir una longitud prefijada.
- Si se proporciona un CONSTRAINT*símbolo*, éste debe ser único en la base de datos. Si no se suministra, InnoDB crea el nombre automáticamente.

InnoDB rechaza cualquier operación INSERT o UPDATE que intente crear un valor de clave foránea en una tabla hija sin un valor de clave candidata coincidente en la tabla padre. La acción que InnoDB lleva a cabo para cualquier operación UPDATE o DELETE que intente actualizar o borrar un valor de clave candidata en la tabla padre que tenga filas coincidentes en la tabla hija depende de la *accion referencial* especificada utilizando las subcláusulas ON UPDATE y ON DETETE en la cláusula FOREIGN KEY. Cuando el usuario intenta borrar o actualizar una fila de una tabla padre, InnoDB soporta cinco acciones respecto a la acción a tomar:

- CASCADE: Borra o actualiza el registro en la tabla padre y automáticamente borra o actualiza los registros coincidentes en la tabla hija. Tanto ON DELETE CASCADE como ON UPDATE CASCADE están disponibles en MySQL 5.0. Entre dos tablas, no se deberían definir varias cláusulas ON UPDATE CASCADE que actúen en la misma columna en la tabla padre o hija.

- SET NULL: Borra o actualiza el registro en la tabla padre y establece en NULL la o las columnas de clave foránea en la tabla hija. Esto solamente es válido si las columnas de clave foránea no han sido definidas como NOT NULL. MySQL 5.0 soporta tanto ON DELETE SET NULL como ON UPDATE SET NULL.

- NO ACTION: En el estándar ANSI SQL-92, NO ACTION significa *ninguna acción* en el sentido de que unintento de borrar o actualizar un valor de clave primaria no sera permitido si en la tabla referenciada hay una valor de clave foránea relacionado. (Gruber, *Mastering SQL*, 2000:181). En MySQL 5.0, InnoDB rechaza la operación de eliminación o actualización en la tabla padre.

- RESTRICT: Rechaza la operación de eliminación o actualización en la tabla padre. NO ACTION y RESTRICT son similares en tanto omiten la cláusula ON DELETE u ON UPDATE. (Algunos sistemas de bases de datos tienen verificaciones diferidas o retrasadas, una de las cuales es NO ACTION. En MySQL, las restricciones de claves foráneas se verifican inmediatamente, por eso, NO ACTION y RESTRICT son equivalentes.)

- SET DEFAULT: Esta acción es reconocida por el procesador de sentencias (parser), pero InnoDB rechaza definiciones de tablas que contengan ON DELETE SET DEFAULT u ON UPDATE SET DEFAULT.

InnoDB soporta las mismas opciones cuando se actualiza la clave candidata en la tabla padre. Con CASCADE, las columnas de clave foránea en la tabla hija son establecidas a los nuevos valores de la clave candidata en la tabla padre. Del mismo modo, las actualizaciones se producen en cascada si las columnas actualizadas en la tabla hija hacen referencia a claves foráneas en otra tabla.

Nótese que InnoDB soporta referencias de clave foránea dentro de una tabla, y, en estos casos, la tabla hija realmente significa registros dependientes dentro de la tabla.

InnoDB necesita que haya índices sobre las claves foráneas y claves referenciadas, así las verificaciones de claves foráneas pueden ser veloces y no necesitan recorrer la tabla. En MySQL 5.0, el índice en la clave foránea se crea automáticamente. Esto contrasta con versiones más antiguas (anteriores a 4.1.8), donde los índices debían crearse explícitamente o fallaba la creación de restricciones de claves foráneas.

Las columnas involucradas en la clave foránea y en la clave referenciada deben tener similares tipos de datos internos dentro de InnoDB, de modo que puedan compararse sin una conversión de tipo. La **longitud y la condición de con o sin signo de los tipos enteros deben ser iguales**. La longitud de los tipos cadena no necesita ser la misma. Si se especifica una acción SET NULL, hay que asegurarse de que **las columnas en la tabla hija no se han declarado** como NOT NULL.

Si MySQL informa que ocurrió un error número 1005 en una sentencia CREATE TABLE y la cadena con el mensaje de error se refiere al errno (número de error) 150, significa que la creación de una tabla falló debido a una restricción de clave foránea formulada incorrectamente. Del mismo modo, si un ALTER TABLE falla y hace referencia al número de error 150, significa que se ha formulado incorrectamente una

restricción de clave extranjera cuando se alteró la tabla. En MySQL 5.0, puede emplearse SHOW INNODB STATUS para mostrar una explicación detallada del último error de clave foránea sufrido por InnoDB en el servidor.

Nota: InnoDB no verifica las restricciones de claves foráneas en las claves foráneas o valores de claves referenciados que contengan una columna NULL.

Una desviación del estándar SQL: Si en la tabla padre hay varios registros que contengan el mismo valor de clave referenciada, entonces InnoDB se comporta en las verificaciones de claves extranjeras como si los demás registros con el mismo valor de clave no existiesen. Por ejemplo, si se ha definido una restricción del tipo RESTRICT, y hay un registro hijo con varias filas padre, InnoDB no permite la eliminación de ninguna de éstas.

InnoDB lleva a cabo las operaciones en cascada a través de un algoritmo de tipo depth-first, basado en los registros de los indices correspondientes a las restricciones de claves foráneas.

Una desviación del estándar SQL: Si ON UPDATE CASCADE u ON UPDATE SET NULL vuelven a modificar la *misma tabla* que se está actualizando en cascada, el comportamiento es como en RESTRICT. Esto significa que en una tabla no se pueden ejecutar operaciones ON UPDATE CASCADE u ON UPDATE SET NULL que hagan referencia a ella misma. De ese modo se previenen bucles infinitos resultantes de la actualización en cascada. En cambio, una operación ON DELETE SET NULL, puede hacer referencia a la misma tabla donde se encuentra, al igual que ON DELETE CASCADE. En MySQL 5.0, las operaciones en cascada no pueden anidarse en más de 15 niveles de profundidad.

Una desviación del estándar SQL: Como sucede en MySQL en general, en una sentencia SQL que realice inserciones, eliminaciones o actualizaciones en varias filas, InnoDB verifica las restricciones UNIQUE y FOREIGN KEY fila a fila. De acuerdo con el estándar SQL, el comportamiento predeterminado debería ser que las restricciones se verifiquen luego de que la sentencia SQL ha sido procesada por completo.

Note: Actualmente, los disparadores no son activados por acciones de claves foráneas en cascada.

Un ejemplo sencillo que relaciona tablas padre e hijo a través de una clave foránea de una sola columna:

```
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
) ENGINE=INNODB;
```

Aquí, un ejemplo más complejo, en el cual una tabla product_order tiene claves foráneas hacia otras dos tablas. Una de las claves foráneas hace referencia a un índice de dos columnas en la tabla product. La otra hace referencia a un índice de una sola columna en la tabla customer:

```
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),
  FOREIGN KEY (product_category, product_id)
    REFERENCES product(category, id)
    ON UPDATE CASCADE ON DELETE RESTRICT,
  INDEX (customer_id),
  FOREIGN KEY (customer_id)
    REFERENCES customer(id)
) ENGINE=INNODB;
```

InnoDB permite agregar una nueva restricción de clave foránea a una tabla empleando ALTER TABLE:

```
ALTER TABLE yourtablename
    ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Debe recordarse crear en primer lugar los índices necesarios.. También se puede agregar una clave foránea autoreferente a una tabla empleando ALTER TABLE.

InnoDB también soporta el uso de ALTER TABLE para borrar claves foráneas:

```
ALTER TABLE nombre_tabla DROP FOREIGN KEY símbolo_clave_foránea;
```

Si la cláusula FOREIGN KEY incluye un nombre de CONSTRAINT cuando se crea la clave foránea, se puede utilizar ese nombre para eliminarla. En otro caso, el valor símbolo_clave_foránea es generado internamente por InnoDB cuando se crea la clave foránea. Para saber cuál es este símbolo cuando se desee eliminar una clave foránea, se emplea la sentencia SHOW CREATE TABLE. Un ejemplo:

```
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;
```

El procesador de sentencias (parser) de InnoDB permite emplear acentos graves (ASCII 96) para encerrar los nombres de tablas y columnas en una clásusula FOREIGN KEY ... REFERENCES ... . El parser de InnoDB también toma en cuenta lo establecido en la variable de sistema lower_case_table_names.

InnoDB devuelve las definiciones de claves foráneas de una tabla como parte de la salida de la sentencia SHOW CREATE TABLE:

```
SHOW CREATE TABLE tbl_name;
```

A partir de esta versión, mysqldump también produce definiciones correctas de las tablas en el fichero generado, sin omitir las claves foráneas.

Se pueden mostrar las restricciones de claves foráneas de una tabla de este modo:

```
SHOW TABLE STATUS FROM nombre_bd LIKE 'nombre_tabla';
```

Las restricciones de claves foráneas se listan en la columna Comment de la salida producida.

Al llevar a cabo verificaciones de claves foráneas, InnoDB establece bloqueos compartidos a nivel de fila en los registros de tablas hijas o padres en los cuales deba fijarse. InnoDB verifica las restricciones de claves foráneas inmediatamente, la verificación no es demorada hasta la confirmación de la transacción.

Para facilitar la recarga de ficheros de volcado de tablas que tengan relaciones de claves foráneas, mysqldump incluye automáticamente una sentencia en la salida del comando para establecer FOREIGN_KEY_CHECKS a 0. Esto evita problemas con tablas que tengan que ser creadas en un orden particular cuando se recarga el fichero de volcado. También es posible establecer el valor de esta variable manualmente:

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
mysql> SOURCE dump_file_name;
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

Esto permite importar las tablas en cualquier orden si el fichero de volcado contiene tablas que no están ordenadas según lo requieran sus claves foráneas. También acelera la operación de importación. Establecer FOREIGN_KEY_CHECKS a 0 también puede ser útil para ignorar restricciones de claves foráneas durante operaciones LOAD DATA y ALTER TABLE.

InnoDB no permite eliminar una tabla que está referenciada por una restricción FOREIGN KEY, a menos que se ejecute SET FOREIGN_KEY_CHECKS=0. Cuando se elimina una tabla, las restricciones que fueron definidas en su sentencia de creación también son eliminadas.

Si se recrea una tabla que fue eliminada, debe ser definida de acuerdo a las restricciones de claves foráneas que están haciendo referencia a ella. Debe tener los tipos y nombres correctos de columnas, y debe tener índices sobre las tablas referenciadas, como se estableció anteriormente. Si estas condiciones no se cumplen, MySQL devuelve un error número 1005 y menciona el error número 150 en el mensaje de error.

15.6.5. InnoDB y replicación MySQL

La replicación en MySQL funciona para tablas InnoDB del mismo modo que lo hace para tablas MyISAM. Es posible usar replicación en una forma en que el tipo de tabla en el servidor esclavo no es igual a la tabla original en el servidor maestro. Por ejemplo, se pueden replicar modificaciones de una tabla InnoDB en el servidor maestro sobre una tabla MyISAM en el servidor esclavo.

Para configurar un nuevo esclavo para un servidor maestro, se debe realizar una copia del espacio de tablas `InnoDB` y de los ficheros de registro, así como de los ficheros `.frm` de las tablas `InnoDB`, y mover las copias al servidor esclavo. El procedimiento adecuado para esto se encuenta en Sección 15.9, “Trasladar una base de datos `InnoDB` a otra máquina”.

Si se puede detener el servidor maestro o un esclavo existente, se puede tomar un backup en frío del espacio de tablas `InnoDB` y de los ficheros de registro y utilizarlos para configurar un servidor esclavo. Para crear un nuevo esclavo sin detener ningún servidor, se puede utilizar la herramienta comercial `InnoDB Hot Backup` tool.

Una limitación menor en la replicación `InnoDB` es que `LOAD TABLE FROM MASTER` no funciona con tablas de tipo `InnoDB`. Hay dos posibles soluciones:

- Hacer un volcado de la tabla en el maestro e importarlo dentro del esclavo.
- Utilizar `ALTER TABLE` *`nombre_tabla`* `TYPE=MyISAM` en el maestro antes de realizar la replicación con `LOAD TABLE` *`nombre_tabla`* `FROM MASTER`, y luego emplear `ALTER TABLE` para convertir la tabla en el maestro nuevamente a `InnoDB`.

Las transacciones que fallen en el maestro no afectan en absoluto la replicación. La replicación en MySQL se basa en el registro (log) binario donde MySQL registra las sentencias SQL que modifican datos. Un esclavo lee el registro binario del maestro y ejecuta las mismas sentencias SQL. Sin embargo, las sentencias emitidas dentro de una transacción no se graban en el registro binario hasta que se confirma la transacción, en ese momento todas las sentencias son grabadas de una vez. Si una sentencia falla, por ejemplo por infringir una clave foránea, o si se cancela una transacción, ninguna sentencia se guarda en el registro binario y la transacción no se ejecuta en absoluto en el servidor esclavo.

15.6.6. Usar un espacio de tablas para cada tabla

En MySQL 5.0, se puede almacenar cada tabla `InnoDB` y sus índices en su propio fichero. Esta característica se llama “multiple tablespaces” (espacios de tablas múltiples) porque, en efecto, cada tabla tiene su propio espacio de tablas.

El uso de múltiples espacios de tablas puede ser beneficioso para usuarios que desean mover tablas específicas a discos físicos separados o quienes deseen restaurar respaldos de tablas sin interrumpir el uso de las demás tablas InnoDB.

Se pueden habilitar múltiples espacios de tablas agregando esta línea a la sección `[mysqld]` de `my.cnf`:

```
[mysqld]
innodb_file_per_table
```

Luego de reiniciar el servidor, `InnoDB` almacenará cada nueva tabla creada en su propio fichero *`nombre_tabla`*`.ibd` en el directorio de la base de datos a la que pertenece la tabla. Esto es similar a lo que hace el motor de almacenamiento `MyISAM`, pero `MyISAM` divide la tabla en un fichero de datos *`tbl_name`*`.MYD` y el fichero de índice *`tbl_name`*`.MYI`. Para `InnoDB`, los datos y los índices se almacenan juntos en el fichero `.ibd`. El fichero *`tbl_name`*`.frm` se sigue creando como es usual.

Si se quita la línea `innodb_file_per_table` de `my.cnf` y se reinicia el servidor, `InnoDB` creará nuevamente las tablas dentro de los ficheros del espacio de tablas compartido.

`innodb_file_per_table` afecta solamente la creación de tablas. Si se inicia el servidor con esta opción, las tablas nuevas se crearán empleando ficheros `.ibd`, pero aún se puede acceder a las tablas existentes en el espacio de tablas compartido. Si se remueve la opción, las nuevas tablas se crearán en el espacio compartido, pero aún se podrá acceder a las tablas creadas en espacios de tablas múltiples.

`InnoDB` siempre necesita del espacio de tablas compartido. Los ficheros `.ibd` no son suficientes para que funcione `InnoDB`. El espacio de tablas compartido consiste de los ya conocidos ficheros `ibdata`, donde `InnoDB` coloca su diccionario de datos interno y los registros para deshacer cambios (undo logs).

No se puede mover libremente ficheros `.ibd` entre directorios de bases de datos en la misma forma en que se hace con ficheros de tablas `MyISAM`. Esto se debe a que la definición de las tablas se almacena en el espacio de tablas compartido de `InnoDB`, y también porque `InnoDB` debe preservar la consistencia de los identificadores de transacciones y los números de secuencia de registros (log).

Dentro de una determinada instalación MySQL, se puede mover un fichero `.ibd` y las tablas asociadas de una base de datos a otra con la conocida sentencia `RENAME TABLE`:

```
RENAME TABLE nombre_bd_anterior.nombre_tabla TO nombre_bd_nuevo.nombre_tabla;
```

Si se tiene un respaldo “limpio” de un fichero `.ibd`, se lo puede restaurar dentro de la instalación MySQL de donde proviene del siguiente modo:

1. Utilizando esta sentencia `ALTER TABLE`:

   ```
   ALTER TABLE nombre_tabla DISCARD TABLESPACE;
   ```

   Precaución: Esto eliminará el actual fichero `.ibd`.

2. Colocando el fichero `.ibd` nuevamente en el directorio de la base de datos adecuada.

3. Utilizando esta sentencia `ALTER TABLE`:

   ```
   ALTER TABLE nombre_tabla IMPORT TABLESPACE;
   ```

En este contexto, un respaldo “limpio” de un fichero `.ibd` significa:

- ç El fichero `.ibd` no contiene modificaciones realizadas por transacciones sin confirmar.
- No quedan entradas sin combinar en el buffer de inserciones en el fichero `.ibd`.
- Se han quitado todos los registros de índice marcados para eliminación en el fichero `.ibd`.
- `mysqld` ha descargado todas las páginas modificadas del fichero `.ibd` desde el buffer pool hacia el fichero.

Se puede realizar un respaldo limpio del fichero `.ibd` con el siguiente método:

1. Detener toda actividad del servidor `mysqld` y confirmar todas las transacciones.

2. Esperar hasta que `SHOW INNODB STATUS` indique que no hay transacciones activas en la base de datos, y el estado del subproceso (trhead) principal de `InnoDB` sea `Waiting for server activity` (Esperando por actividad del servidor). Entonces, se puede hacer una copia del fichero `.ibd`.

Otro método para hacer una copia limpia de un fichero `.ibd` es utilizar la herramienta comercial `InnoDB Hot Backup`:

1. Utilizar `InnoDB Hot Backup` para respaldar la instalación `InnoDB`.

2. Iniciar un segundo servidor `mysqld` sobre el respaldo y permitirle limpiar los ficheros `.ibd` del mismo.

Figura en la lista de pendientes (TODO) para permitir mover ficheros .ibd limpios a otra instalación MySQL. Esto necesita que se inicialicen los IDs (identificadores) de transacciones y los números de secuencia de registros (log) en el fichero .ibd.

15.7. Añadir y suprimir registros y ficheros de datos InnoDB

Esta sección describe lo que se puede hacer cuando el espacio de tablas InnoDB se queda sin espacio o cuando se desea cambiar el tamaño de los ficheros de registro (log).

La manera más sencilla de incrementar el tamaño del espacio de tablas InnoDB es configurarlo desde un principio para que sea autoextensible, especificando el atributo autoextend para el último fichero de datos en la definición del espacio de tablas. Entonces, InnoDB incrementa el tamaño de ese fichero automáticamente en intervalos de 8MB cuando se queda sin espacio. El tamaño del intervalo a incrementar puede configurarse estableciendo el valor de innodb_autoextend_increment, el cual está expresado en megabytes, y cuyo valor predeterminado es 8.

Alternativamente, se puede incrementar el tamaño del espacio de tablas agregando otro fichero de datos. Para hacer esto, se debe detener el servidor MySQL, editar el fichero my.cnf para agregar un nuevo fichero de datos al final de innodb_data_file_path, e iniciar nuevamente el servidor.

Si el último fichero de datos especificado tiene la palabra clave autoextend, el procedimiento para editar a my.cnf debe tener en cuenta el tamaño que ha alcanzado este último fichero. Hay que obtener el tamaño del fichero de datos, redondearlo hacia abajo a la cantidad de megabytes (1024 * 1024 bytes) más cercana, y especificar este número explícitamente en innodb_data_file_path. Entonces se podrá agregar otro fichero de datos. Hay que recordar que solamente el último fichero de datos en innodb_data_file_path puede especificarse como autoextensible.

Como ejemplo, se asumirá que el espacio de tablas tiene sólo un fichero de datos autoextensible ibdata1:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suponiendo que este fichero de datos, a lo largo del tiempo, ha crecido hasta 988MB, debajo se ve la línea de configuración luego de agregar otro fichero de datos autoextensible.

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

Cuando se agrega un nuevo fichero al espacio de tablas, hay que asegurarse de que no exista. InnoDB crea e inicializa el fichero al reiniciar el servidor.

Actualmente no es posible quitar un fichero de datos del espacio de tablas. Para reducir el tamaño del espacio de tablas, emplear este procedimiento:

1. Utilizar mysqldump para hacer un volcado de todas las tablas InnoDB.
2. Detener el servidor.
3. Eliminar todos los ficheros existentes del espacio de tablas.
4. Configurar un nuevo espacio de tablas.
5. Reiniciar el servidor.
6. Importar el fichero de volcado de tablas.

Si se desea modificar la cantidad o tamaño de los ficheros de registro (log) de InnoDB, se debe detener el servidor MySQL y asegurarse de que se cerró sin errores. Luego, copiar los ficheros de registro antiguos en un lugar seguro, sólo para el caso de que algo haya fallado en el cierre del servidor y se necesite recuperar el espacio de tablas. Eliminar los antiguos ficheros de registro del directorio de ficheros de registro, editar my.cnf para modificar la configuración de los ficheros de registro, e iniciar nuevamente el servidor MySQL. mysqld verá al iniciar que no hay ficheros de registro e informará que está creando nuevos.

15.8. Hacer una copia de seguridad y recuperar una base de datos InnoDB

La clave de una administración de bases de datos segura es realizar copias de respaldo regularmente.

InnoDB Hot Backup es una herramienta de respaldo en línea que puede utilizarse para respaldar la base de datos InnoDB mientras ésta se está ejecutando. InnoDB Hot Backup no necesita que se detenga la base de datos y no establece ningún bloqueo ni dificulta el normal procesamiento de la base de datos. InnoDB Hot Backup es una herramienta adicional comercial (no grautita) cuyo cargo anual de licencia es de €390 por cada ordenador en el que se ejecute el servidor MySQL. Consulte la página de Internet de InnoDB Hot Backup para obtener información detallada y ver capturas de pantallas.

Si se está en condiciones de detener el servidor MySQL, puede realizarse una copia de respaldo binaria, que consiste en todos los ficheros usados por InnoDB para administrar sus tablas. Se utiliza el siguiente procedimiento:

1. Detener el servidor MySQL y asegurarse de que lo hace sin errores.
2. Copiar todos los ficheros de datos (ficheros ibdata e .ibd) en un lugar seguro.
3. Copiar todos los ficheros ib_logfile en un lugar seguro.
4. Copiar el o los ficheros de configuración my.cnf en un lugar seguro.
5. Copiar todos los ficheros .frm de las tablas InnoDB en un lugar seguro.

La replicación funciona con tablas InnoDB, de forma que puede emplearse para mantener una copia de la base de datos en sitios de bases de datos que necesiten alta disponibilidad.

Adicionalmente a la realización de copias de respaldo binarias como se ha descripto, también se deberían realizar regularmente volcados de las tablas con mysqldump. El motivo de esto es que un fichero binario podría corromperse sin que el usuario lo note. El volcado de las tablas se almacena en ficheros de texto que son legibles por los seres humanos, facilitando la localización de corrupción en las tablas. Además, puesto que el formato es más simple, las probabilidades de una corrupción seria de datos son menores. mysqldump también tiene una opción --single-transaction que puede usarse para capturar una imagen consistente de la base de datos sin bloquear otros clientes.

Para estar en condiciones de recuperar una base de datos InnoDB a partir del respaldo binario descripto anteriormente, se debe ejecutar el servidor MySQL con el registro binario (binary logging) activo. Entonces se puede aplicar el log binario al respaldo de la base de datos para lograr la recuperación a una fecha y hora determinadas:

```
mysqlbinlog nombre_de_host-bin.123 | mysql
```

Para recuperarse de una caida del servidor, sólo se requiere reiniciarlo. InnoDB verifica automáticamente los registros (logs) y ejecuta una recuperación de la base de datos del tipo roll-forward, es decir, hasta el

momento anterior a la falla. InnoDB revierte automáticamente las transacciones no grabadas que existían al momento de la caída. Durante la recuperación, mysqld muestra información parecida a esta:

```
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
```

Si la base de datos se corrompe o falla el disco, habrá que efectuar la recuperación desde una copia de respaldo. En el caso de corrupción, primero habría que encontrar una copa de respaldo realizada antes de la corrupción. Luego de restaurar la copia de respaldo base, debe realizarse la recuperación a partir de los ficheros de registro binario.

En algunos casos de corrupción de base de datos es suficiente con volcar, eliminar, y volver a crear una o unas pocas tablas corruptas. Se puede emplear la sentencia SQL CHECK TABLE para verificar si una tabla está corrupta, aunque CHECK TABLE, naturalmente, no puede detectar cada posible clase de corrupción. Se puede emplear innodb_tablespace_monitor para verificar la integridad de la gestión de espacio de ficheros dentro de los ficheros de espacio de tablas.

En algunos casos, una aparente corrupción de página de base de datos se debe en realidad a que el sistema operativo está corrompiendo su propio cache de ficheros, y los datos en el disco podrían estar en buenas condiciones. Lo mejor es, antes que nada, intentar el reinicio del ordenador. Ello puede eliminar errores que dan la sensación de tener corrupción en la página de base de datos.

15.8.1. Forzar una recuperación

Si hay corrupción de página de base de datos, es posible que se desee hacer un volcado de tablas desde la base de datos con SELECT INTO OUTFILE; usualmente, la mayoría de los datos obtenidos de esta manera están intactos. Aún así, la corrupción puede hacer que SELECT * FROM *tbl_name* o las operaciones en segundo plano de InnoDB caigan o, incluso activar la recuperación roll-forward de InnoDB. Sin embargo, se puede forzar el motor de almacenamiento de InnoDB para que se inicie mientras se evitan las operaciones en segundo plano, de forma que se pueda realizar un volcado de las tablas. Por ejemplo, puede agregarse la siguiente linea a la sección [mysqld] del fichero de opciones antes de reiniciar el servidor:

```
[mysqld]
innodb_force_recovery = 4
```

Debajo se listan los valores mayores a cero permitidos para innodb_force_recovery. Un número mayor incluye todas las precauciones de los números por debajo. Si se logra hacer un volcado de tablas con un valor de a lo sumo 4, se puede estar relativamente seguro de que solamente se han perdido

algunos datos en páginas individuales corruptas. Un valor de 6 es más dramático, porque las páginas de base de datos han quedado obsoletas, lo que a su vez puede introducir más corrupción en B-trees y otras estructuras de bases de datos.

- `1` (`SRV_FORCE_IGNORE_CORRUPT`)

  Le permite al servidor ejecutarse aún si detecta una página corrupta; intenta hacer que `SELECT * FROM tbl_name` salte sobre los registros de índice y páginas corruptos, lo cual es de ayuda en el volcado de las tablas.

- `2` (`SRV_FORCE_NO_BACKGROUND`)

  Impide que se ejecute el subproceso (thread) principal. Si durante la operación de descarga (purge) ocurriese una caida, esto la previene.

- `3` (`SRV_FORCE_NO_TRX_UNDO`)

  No revierte transacciones luego de la recuperación.

- `4` (`SRV_FORCE_NO_IBUF_MERGE`)

  También evita las operaciones de combinación del buffer de inserciones. Si ello pudiese causar una caida, es mejor no hacerlo; no calcula estadísticas de tablas.

- `5` (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

  No tiene en cuenta el contenido de los registros para revertir cambios (undo logs) al iniciar la base de datos: `InnoDB` considera como confirmadas inclusive a las transacciones incompletas.

- `6` (`SRV_FORCE_NO_LOG_REDO`)

  Omite la creación del registro (log) que permite la recuperación de tipo roll-forward.

En otro caso la base de datos no debe emplearse con ninguna de estas opciones habilitadas. Como una medida de seguridad, `InnoDB` impide que los usuarios lleven a cabo operaciones `INSERT`, `UPDATE`, o `DELETE` cuando `innodb_force_recovery` está configurado en un valor mayor a 0.

Se pueden ejecutar sentencias `DROP` y `CREATE` para eliminar y crear tablas incluso cuando se está empleando la recuperación forzada. Si se sabe que una determinada tabla está provocando caídas al hacer una cancelación (rollback), se la puede eliminar. También puede utilizarse para detener una cancelación fuera de control causada por una importación o un `ALTER TABLE` en masa. Se puede interrumpir el proceso `mysqld` y establecer `innodb_force_recovery` a un valor de `3` para poner a funcionar la base de datos sin que ejecute cancelación (rollback), luego emplear `DROP` para eliminar la tabla que está causando la cancelación fuera de control.

15.8.2. Marcadores

`InnoDB` implementa un mecanismo de puntos de verificación llamado *fuzzy checkpoint* (punto de verificación difuso). `InnoDB` descarga las páginas modificadas de la base de datos desde el pool de buffers en lotes pequeños. No es necesario descargar al disco el pool de buffers en una sola acción, lo cual en la práctica podría detener temporalmente el procesamiento de sentencias SQL del usuario.

Durante la recuperación de caídas, `InnoDB` busca un marcador de checkpoint escrito en los ficheros de registro (log). Ya sabe que todas las modificaciones a la base de datos anteriores al marcador están presentes en la imagen en disco de la misma. Entonces `InnoDB` recorre los ficheros de registro (log) desde el checkpoint hacia adelante, aplicando sobre la base de datos las modificaciones registradas.

InnoDB escribe en los ficheros de registro en una forma rotativa. Todas las modificaciones confirmadas que hagan a las páginas de la base de datos en el pool de buffers diferentes de las grabadas en disco deben estar disponibles en los ficheros de registro en caso de que InnoDB tenga que hacer una recuperación. Esto significa que cuando InnoDB comienza a reutilizar un fichero de registro, tiene que asegurarse de que las imágenes en disco de las páginas de base de datos contienen las modificaciones asentadas en el fichero de registro que se va a reutilizar. En otras palabras, InnoDB debe crear un punto de verificación (checkpoint) y esto a menudo implica la descarga a disco de las páginas de base de datos modificadas.

La descripción anterior explica porqué hacer los ficheros de registro muy grandes puede ahorrar operaciones de E/S en disco destinadas a la creación de puntos de verificación. A menudo se hace hincapié en establecer el tamaño total de los ficheros de registro en lo mismo que el pool de buffers o aún más grande. La desventaja que tienen los ficheros de registro grandes es que la recuperación ante una caída puede tomar más tiempo debido a que hay más información que debe aplicarse a la base de datos.

15.9. Trasladar una base de datos InnoDB a otra máquina

En Windows, InnoDB siempre almacena internamente en minúsculas los nombres de bases de datos y tablas. Para mover bases de datos en un formato binario de Unix a Windows o de Windows a Unix, se deberían tener en minúsculas todos los nombres de tablas y bases de datos. Una forma apropiada de cumplir con esto es agregar la siguiente línea a la sección [mysqld] de los ficheros my.cnf o my.ini antes de crear cualquier base de datos o tablas:

```
[mysqld]
lower_case_table_names=1
```

Al igual que los ficheros de datos MyISAM, los ficheros de datos y de registro (log) de InnoDB son compatibles a nivel binario en todas las plataformas que tengan el mismo formato de números de coma flotante. Se puede mover una base de datos InnoDB simplemente copiando todos los ficheros relevantes que se listan en Sección 15.8, “Hacer una copia de seguridad y recuperar una base de datos InnoDB”. Si los formatos de número de coma flotante difieren pero no se han empleado tipos de datos FLOAT o DOUBLE en las tablas, el procedimiento es el mismo: copiar los ficheros necesarios. Si los formatos difieren y las tablas contienen datos de coma flotante, se deberá emplear mysqldump para volcar las tablas en un ordenador e importar los ficheros de volcado en otro.

Una forma de incrementar el rendimiento es desactivar el modo autocommit (ejecución automática) al importar datos, asumiendo que el espacio de tablas tiene suficiente sitio para el extenso segmento de cancelación (rollback) que generará la importación de transacciones. La confirmación (commit) se hará luego de importar una tabla entera o un segmento de una tabla.

15.10. Bloqueo y modelo de transacciones de InnoDB

En el modelo de transacciones de InnoDB, la meta es combinar las mejores propiedades de una base de datos multiversión con el tradicional bloqueo de dos fases. InnoDB bloquea a nivel de fila y ejecuta consultas por defecto como lecturas consistentes (consistent reads) no bloqueadas, al estilo de Oracle. La tabla de bloqueo en InnoDB se almacena en forma tan eficiente que no se necesitan bloqueos escalables: generalmente varios usuarios están habilitados a bloquear cada fila de la base de datos, o cualquier subconjunto de filas, sin que InnoDB incurra en falta de memoria.

15.10.1. Modos de bloqueo InnoDB

InnoDB implementa un bloqueo a nivel de fila estándar, donde hay dos tipos de bloqueos:

- Compartido (Shared) (*S*) le permite a una transacción leer una fila.

- Exclusivo (Exclusive) (*X*) le permite a una transacción actualizar o eliminar una fila.

Si una transacción *A* sostiene un bloqueo exclusivo (*X*) sobre una tupla *t*, entonces una solicitud de otra transacción *B* para establecer un bloqueo de cualquier tipo sobre *t* no puede ser atendida inmediatamente. En lugar de eso, la transacción *B* debe esperar a que la transacción *A* libere el bloqueo en la tupla *t*.

Si la transacción *A* sostiene un bloqueo compartido (*S*) sobre una tupla *t*, entonces

- Una solicitud de otra transacción *B* para un bloqueo *X* sobre *t* no puede ser atendido inmediatamente.
- Una solicitud de otra transacción *B* para un bloqueo *S* sobre *t* puede ser atendido inmediatamente. En consecuencia, tanto *A* como *B* sostendrán un bloqueo *S* sobre *t*.

Adicionalmente, InnoDB soporta *bloqueo de granularidad múltiple* (multiple granularity locking), el cual permite que existan simultáneamente bloqueos en registros y bloqueos en tablas enteras. Para hacer práctico el nivel de bloqueo de granularidad múltiple, se emplean tipos adicionales de bloqueo, llamados *bloqueos de intención* (intention locks). Los bloqueos de intención son bloqueos de tabla en InnoDB. La idea detrás de los mismos es que una transacción indique qué tipo de bloqueo (compartido o exclusivo) requerirá más tarde sobre una fila de esa tabla. En InnoDB se utilizan dos tipos de bloqueos de intención (asumiendo que la transacción *T* ha solicitado un bloqueo del tipo indicado en la tabla *R*):

- Intención compartida (Intention shared) (*IS*): La transacción *T* trata de establecer bloqueos *S* en tuplas individuales de la tabla *T*.
- Intención exclusiva (Intention exclusive) (*IX*): La transacción *T* trata de establecer bloqueos *X* en las tuplas.

Luego, el protocolo de bloqueo de intención es el siguiente:

- Antes de que de una determinada transacción logre un bloqueo *S* en una determinada fila, primero debe conseguir establecer un bloqueo *IS* o superior en la tabla que contiene a la fila.
- Antes de que de una determinada transacción logre un bloqueo *X* en una determinada fila, primero debe conseguir establecer un bloqueo *IX* en la tabla que contiene a la fila.

Estas reglas pueden resumirse convenientemente por medio de una *matriz de compatibilidad entre tipos de bloqueo*:

	X	IX	S	IS	-
X	N	N	N	N	S
IX	N	S	N	S	S
S	N	S	S	S	S
IS	N	S	S	S	S
-	S	S	S	S	S

Por lo tanto, los bloqueos de intención solamente bloquean solicitudes sobre tablas completas (Ej: `LOCK TABLES ... WRITE`). El propósito principal de *IX* y *IS* es mostrar que alguien está bloqueando una fila, o va a bloquear una fila en la tabla.

15.10.2. InnoDB y AUTOCOMMIT

En InnoDB, toda la actividad del usuario se produce dentro de una transacción. Si el modo de ejecución automática (autocommit) está activado, cada sentencia SQL conforma una transacción individual por sí misma. MySQL siempre comienza una nueva conexión con la ejecución automática habilitada.

Si el modo de ejecución automática se deshabilitó con `SET AUTOCOMMIT = 0`, entonces puede considerarse que un usuario siempre tiene una transacción abierta. Una sentencia SQL `COMMIT` o `ROLLBACK` termina la transacción vigente y comienza una nueva. Ambas sentencias liberan todos los bloqueos `InnoDB` que se establecieron durante la transacción vigente. Un `COMMIT` significa que los cambios hechos en la transacción actual se convierten en permanentes y se vuelven visibles para los otros usuarios. Por otra parte, una sentencia `ROLLBACK`, cancela todas las modificaciones producidas en la transacción actual.

Si la conexión tiene la ejecución automática habilitada, el usuario puede igualmente llevar a cabo una transacción con varias sentencias si la comienza explícitamente con `START TRANSACTION` o `BEGIN` y la termina con `COMMIT` o `ROLLBACK`.

15.10.3. InnoDB y TRANSACTION ISOLATION LEVEL

En los términos de los niveles de aislamiento de transacciones SQL:1992, el nivel predeterminado en `InnoDB` es `REPEATABLE READ`. En MySQL 5.0, `InnoDB` ofrece los cuatro niveles de aislamiento de transacciones descriptos por el estándar SQL. Se puede establecer el nivel predeterminado de aislamiento por todas las conexiones mediante el uso de la opción `--transaction-isolation` en la línea de comandos o en ficheros de opciones. Por ejemplo, se puede establecer la opción en la sección `[mysqld]` de `my.cnf` de este modo:

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                         | REPEATABLE-READ | SERIALIZABLE}
```

Un usuario puede cambiar el nivel de aislamiento de una sesión individual o de todas las nuevas conexiones con la sentencia `SET TRANSACTION`. Su sintaxis es la siguiente:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
                       {READ UNCOMMITTED | READ COMMITTED
                        | REPEATABLE READ | SERIALIZABLE}
```

Nótese que se usan guiones en los nombres de niveles de la opción `--transaction-isolation`, pero no en la sentencia `SET TRANSACTION`.

El comportamiento predeterminado es establecer el nivel de aislamiento a partir de la próxima transacción que se inicie. Si se emplea la palabra clave `GLOBAL`, la sentencia establece el nivel predeterminado de la transacción globalmente para todas las nuevas conexiones creadas a partir de ese punto (pero no en las existentes). Se necesita el privilegio `SUPER` para hacer esto. Utilizando la palabra clave `SESSION` se establece el nivel de transacción para todas las futuras transacciones ejecutadas en la actual conexión.

Cualquier cliente es libre de cambiar el nivel de aislamiento de la sesión (incluso en medio de una transacción), o el nivel de aislamiento para la próxima transacción.

Los niveles de aislamiento de transacciones globales y de sesión pueden consultarse con estas sentencias:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

En el bloqueo a nivel de fila, `InnoDB` emplea bloqueo de clave siguiente (next-key). Esto significa que, además de registros de índice, `InnoDB` también puede bloquear el “vacío” que precede a un registro de índice para bloquear inserciones de otros usuarios inmediatamente antes del registro de índice. Un

bloqueo de clave siguiente hace referencia a bloquear un registro de índice y la posición vacía antes de él. Bloquear una posición vacía es establecer un bloqueo que actúa solamente sobre el vacío anterior a un registro de índice.

A continuación una descripción detallada de cada nivel de aislamiento en `InnoDB`:

- `READ UNCOMMITTED`

  Las sentencias `SELECT` son ejecutadas sin realizar bloqueos, pero podría usarse una versión anterior de un registro. Por lo tanto, las lecturas no son consistentes al usar este nivel de aislamiento. Esto también se denomina “lectura sucia” (dirty read). En otro caso, este nivel de aislamiento funciona igual que `READ COMMITTED`.

- `READ COMMITTED`

  Similar en parte al mismo nivel de aislamiento de Oracle. Todas las sentencias `SELECT ... FOR UPDATE` y `SELECT ... LOCK IN SHARE MODE` bloquean solamente los registros de índice, no los espacios vacíos que los preceden, por lo tanto se permite la libre inserción de nuevos registros junto a los bloqueados. Las sentencias `UPDATE` and `DELETE` que empleen un índice único con una condición de búsqueda única bloquean solamente el registro de índice hallado, no el espacio que lo precede. En las sentencias `UPDATE` y `DELETE` que actúan sobre rangos de registros, `InnoDB` debe bloquear los espacios vacíos y bloquear las inserciones de otros usuarios en los espacios vacíos que hay dentro del rango. Esto es necesario debido a que las “filas fantasma” deben ser bloqueadas para que funcionen la replicación y recuperación en MySQL.

  Las lecturas consistentes se comportan como en Oracle: Cada lectura consistente, incluso dentro de la misma transacción, establece y lee su propia captura tomada de la base de datos. Consulte Sección 15.10.4, “Lecturas consistentes que no bloquean”.

- `REPEATABLE READ`

  Este es el nivel de aislamiento predeterminado de `InnoDB`. Las sentencias `SELECT ... FOR UPDATE`, `SELECT ... LOCK IN SHARE MODE`, `UPDATE`, y `DELETE` que utilicen un índice único con una condición de búsqueda única, bloquean solamente el registro de índice hallado, no el espacio vacío que lo precede. Con otras condiciones de búsqueda, estas operaciones emplean bloqueo de clave siguiente (next-key), bloqueando el rango de índice cubierto por la operación incluyendo los espacios vacíos, y bloqueando las nuevas inserciones por parte de otros usuarios.

  En lecturas consistentes (consistent reads), hay una importante diferencia con respecto al nivel de aislamiento anterior: En este nivel, todas las lecturas consistentes dentro de la misma transacción leen de la captura de la base de datos tomada por la primer lectura. Esta práctica significa que si se emiten varias sentencias `SELECT` dentro de la misma transacción, éstas serán consistentes unas con otras. Consulte Sección 15.10.4, “Lecturas consistentes que no bloquean”.

- `SERIALIZABLE`

  Este nivel es similar a `REPEATABLE READ`, pero todas las sentencias `SELECT` son convertidas implícitamente a `SELECT ... LOCK IN SHARE MODE`.

15.10.4. Lecturas consistentes que no bloquean

Lectura consistente significa que `InnoDB` utiliza su característica de multiversión para presentar a una consulta una captura de la base de datos en un momento determinado. La consulta ve los cambios realizados exactamente por aquellas transacciones confirmadas antes de ese momento, y no los cambios hechos con posterioridad o por transacciones no confirmadas. La excepción a esto es que la consulta ve los cambios efectuados por la transacción a donde pertenece.

Si se está ejecutando con el nivel de aislamiento predeterminado REPEATABLE READ, entonces todas las lecturas consistentes dentro de la misma transacción leen la captura creada por la primer lectura en esa transacción. Se puede refrescar esta captura confirmando la transacción actual y emitiendo nuevas consultas.

Lectura consistente es el modo por defecto en el cual InnoDB procesa las sentencias SELECT en los niveles de aislamiento READ COMMITTED y REPEATABLE READ. Una lectura consistente no establece ningún bloqueo en las tablas a las que accede, y, por lo tanto, otros usuarios están libres para modificar las tablas sobre las que se está haciendo la lectura consistente.

15.10.5. Bloquear lecturas SELECT ... FOR UPDATE y SELECT ... LOCK IN SHARE MODE

En ciertas circunstancias, no es conveniente una lectura consistente. Por ejemplo, se podría desear agregar una fila en la tabla hijo, y estar seguro de que dicha fila tiene una fila padre en la tabla padre. El siguiente ejemplo muestra cómo implementar integridad referencial en el código de la aplicación.

Suponiendo que se utiliza una lectura consistente para leer la tabla padre y efectivamente puede verse el registro padre para la fila hijo que se agregará, ¿puede agregarse en forma segura la fila hijo dentro de la tabla hijo? No, porque puede haber ocurrido que entretanto otro usuario haya borrado el registro padre de la tabla padre, sin que se tenga conocimiento de ello.

La solución es llevar a cabo el SELECT en un modo con bloqueo, utilizando LOCK IN SHARE MODE:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

Realizar una lectura en modo compartido (share mode) significa que se leen los últimos datos disponibles, y se establece un bloqueo en modo compartido en los registros que se leen. Un bloqueo en modo compartido evita que otros actualicen o eliminen la fila que se ha leido. Además, si los datos más actualizados pertenecen a una transacción todavía no confirmada de otra conexión, se espera hasta que la transacción se confirme. Luego de ver que la mencionada consulta devuelve el registro padre 'Jones', se puede agregar con seguridad el registro hijo en la tabla hijo y confirmar la transacción.

Otro ejemplo: se tiene un campo contador, entero, en una tabla llamada child_codes que se emplea para asignar un identificador único a cada registro hijo agregado a la tabla hijo. Obviamente, utilizar una lectura consistente o una lectura en modo compartido para leer el valor actual del contador no es una buena idea, puesto que dos usuarios de la base de datos pueden ver el mismo valor del contador, y agregar registros hijos con el mismo identificador, lo cual generaría un error de clave duplicada.

En este caso, LOCK IN SHARE MODE no es una buena solución porque si dos usuarios leen el contador al mismo tiempo, al menos uno terminará en un deadlock cuando intente actualizar el contador.

En este caso, hay dos buenas formas de implementar la lectura e incremento del contador: (1), actualizar el contador en un incremento de 1 y sólo después leerlo, o (2) leer primero el contador estableciendo un bloqueo FOR UPDATE, e incrementándolo luego. La última puede ser implementada como sigue:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

Una sentencia SELECT ... FOR UPDATE lee el dato más actualizado disponible, estableciendo bloqueos exclusivos sobre cada fila leída. Es decir, el mismo bloqueo que haría UPDATE.

Nótese que el anterior es un sencillo ejemplo de cómo funciona SELECT ... FOR UPDATE. En MySQL, la tarea específica para generar un identificador único en realidad puede realizarse utilizando un sólo acceso a la tabla:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

La sentencia `SELECT` simplemente recupera la información del identificador (relativa a la conexión actual). No accede ninguna tabla.

15.10.6. Bloqueo de la próxima clave (Next-Key Locking): evitar el problema fantasma

En el bloqueo a nivel de fila, `InnoDB` utiliza un algoritmo llamado *bloqueo de próxima clave*. `InnoDB` lleva a cabo el bloqueo a nivel de fila de tal manera que cuando busca o recorre el índice de una tabla, establece bloqueos compartidos o exclusivos en los registros de índice que encuentra. Por lo tanto, los bloqueos a nivel de fila son en realidad bloqueos sobre registros del índice.

El conjunto de bloqueos de `InnoDB` sobre los registros del índice también afecta al “gap” (posición vacía) que precede al registro de índice. Si un usuario tiene un bloqueo compartido o exclusivo sobre un registro `R` en un índice, otro usuario no puede insertar un nuevo registro inmediatamente antes de `R` en el orden del índice. Este bloqueo de posiciones vacías se hace para evitar el llamado “problema fantasma”. Suponiendo que se desean leer y bloquear todos los hijos de la tabla `hijos` que tengan un identificador mayor a 100, con el posterior intento de actualizar algunas columnas en las filas seleccionadas:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suponiendo que hay un índice sobre la columna `id`, la consulta recorre ese índice comenzando por el primer registro donde `id` es mayor a 100. Si el bloqueo establecido sobre el índice no bloqueara también las inserciones hechas en las posiciones vacías, durante el proceso se podría insertar una nueva fila en la tabla. Si se ejecuta la misma sentencia `SELECT` dentro de la misma transacción, se podría ver una nueva fila en el conjunto de resultados devuelto por la consulta. Esto es contrario al principio de aislamiento de las transacciones: una transacción deberia ejecutarse de forma que los datos que ha leido no cambien en el transcurso de la misma. Si se considera un conjunto de columnas como datos, el nuevo registro hijo “fantasma” violaría el principio de aislamiento.

Cuando `InnoDB` recorre un índice, también puede bloquear la posición vacía después del último registro del índice. Es precisamente lo que ocurre en el ejemplo anterior: Los bloqueos impuestos por `InnoDB` evitan cualquier inserción en la tabla donde `id` fuera mayor de 100.

Se puede emplear bloqueo de próxima clave para efectuar el control de la unicidad en una aplicación: Si se leen los datos en modo compartido y no se ve un duplicado de la fila que se va a insertar, entonces puede hacerse con la seguridad de que el bloqueo de próxima clave establecido sobre el registro que continúa a la fila insertada evita que cualquiera inserte un duplicado de ésta. Por lo tanto, el bloqueo de próxima clave permite “bloquear” la no existencia de algo en la tabla.

15.10.7. Un ejemplo de lectura consistente en `InnoDB`

Suponiendo que se está ejecutando en el nivel de aislamiento predeterminado `REPEATABLE READ`, cuando se realiza una lectura consistente -esto es, una sentencia `SELECT` ordinaria-, `InnoDB` le otorga a la transacción un punto en el tiempo (timepoint) del momento en que se realizó la consulta. Si otra transacción elimina una fila y confirma la acción en un momento posterior a dicho punto, no se verá la fila como borrada. Las inserciones y actualizaciones se tratan del mismo modo.

Se puede obtener un timepoint más reciente confirmando la transacción actual y emitiendo un nuevo `SELECT`.

Esto se llama *control de concurrencia multiversión*.

```
               Usuario A                Usuario B

           SET AUTOCOMMIT=0;      SET AUTOCOMMIT=0;
tiempo
|          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
```

En este ejemplo, el usuario A podrá ver la fila insertada por B solamente cuando B haya confirmado la inserción y A haya confirmado también, de modo que su timepoint avance e incluya la inserción confirmada por B.

Si se desea ver el “más reciente” estado de la base de datos, se debería emplear ya sea el nivel de aislamiento `READ COMMITTED` o bien una lectura con bloqueo:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

15.10.8. Establecimiento de bloqueos con diferentes sentencias SQL en InnoDB

Una lectura con bloqueo, un `UPDATE`, o un `DELETE` generalmente establecen bloqueos sobre cada registro de índice que es examinado durante el procesamiento de la consulta SQL. No importa si en la consulta hay condiciones `WHERE` que excluirían la fila, `InnoDB` no recuerda exactamente la condición `WHERE`, solamente los rangos de índices que fueron examinados. Los bloqueos sobre los registros son normalmente bloqueos de próxima clave, que también impiden las inserciones en las posiciones vacías (“gap”) inmediatamente anteriores a los registros.

Si los bloqueos a establecer son exclusivos, entonces `InnoDB` recupera también los registros de índices agrupados (clustered) y los bloquea.

Si no hay índices apropiados para la consulta y MySQL debe examinar la tabla entera para procesarla, se bloqueará cada fila en la tabla, lo que impide cualquier inserción de otros usuarios. Es importante crear índices adecuados de modo que las consultas no examinen muchas filas innecesariamente.

- `SELECT ... FROM` es una lectura consistente, que lee una captura de la base de datos y no establece bloqueos a menos que el nivel de aislamiento de la transacción sea `SERIALIZABLE`. Para el nivel `SERIALIZABLE`, se establecen bloqueos compartidos de próxima clave en los registros de índice encontrados.
- `SELECT ... FROM ... LOCK IN SHARE MODE` establece bloqueos compartidos de próxima clave en todos los registros de índice hallados por la lectura.
- `SELECT ... FROM ... FOR UPDATE` establece bloqueos exclusivos de próxima clave en todos los registros de índice hallados por la lectura.

- `INSERT INTO ... VALUES (...)` establece un bloqueo exclusivo sobre la fila insertada. Nótese que no se trata de un bloqueo de próxima clave, y no evita que otros usuarios inserten registros en la posición vacía precedente. Si ocurriese un error por duplicación de claves, se establecerá un bloqueo compartido sobre el registro de índice duplicado.
- Mientras se inicializa una columna previamente declarada `AUTO_INCREMENT`, `InnoDB` establece un bloqueo exclusivo al final del índice asociado con dicha columna. Al accederse al contador de autoincremento, `InnoDB` emplea una modo de bloqueo de tabla específico llamado `AUTO-INC`, que dura solamente hasta el final de la actual consulta SQL, en lugar de existir hasta el final de la transacción. Consulte Sección 15.10.2, “`InnoDB` y `AUTOCOMMIT`”.

  En MySQL 5.0, `InnoDB` trae el valor de una columna previamente declarada `AUTO_INCREMENT` sin establecer ningún bloqueo.
- `INSERT INTO T SELECT ... FROM S WHERE ...` establece un bloqueo exclusivo (pero no de próxima clave) en cada fila insertada dentro de `T`. La búsqueda en `S` se hace como una lectura consistente, pero se establecen bloqueos compartidos de próxima clave en `S` si está activado el registro binario (binary logging) de MySQL. `InnoDB` tiene que establecer bloqueos en este último caso: en una recuperación de tipo roll-forward desde una copia de respaldo, cada semtencia SQL debe ser ejecutada en exactamente la misma manera en que se hizo originalmente.
- `CREATE TABLE ... SELECT ...` lleva a cabo el `SELECT` como una lectura consistente o con bloqueos compartidos, como en el punto anterior.
- `REPLACE` se ejecuta del mismo modo que una inserción si no hay colisiones con claves únicas. En otro caso, se coloca un bloqueo exclusivo de próxima clave en la fila que será actualizada.
- `UPDATE ... WHERE ...` establece un bloqueo exclusivo de próxima clave sobre cada registro encontrado por la búsqueda.
- `DELETE FROM ... WHERE ...` establece un bloqueo exclusivo de próxima clave sobre cada registro encontrado por la búsqueda.
- Si se define una restricción `FOREIGN KEY` sobre una tabla, cualquier inserción, actualización o eliminación que necesite la verificación de las condiciones impuestas por la restricción establecerá bloqueos compartidos a nivel de registro sobre los registros examinados durante la verificación. `InnoDB` también establece estos bloqueos en el caso de que la verificación falle.
- `LOCK TABLES` establece bloqueos de tabla, pero es la capa de MySQL de mayor nivel por debajo de la capa de `InnoDB` la que establece estos bloqueos. `InnoDB` tiene conocimiento de los bloqueos de tabla si se establecen `innodb_table_locks=1` y `AUTOCOMMIT=0`, y la capa de MySQL por debajo de `InnoDB` sabe acerca de los bloqueos a nivel de fila. En otro caso, la detección automática de deadlocks de InnoDB no puede detectar los deadlocks donde estén involucradas estas tablas. Además, puesto que la capa superior de MySQL no sabe acerca de bloqueos a nivel de fila, es posible obtener un bloqueo de tabla sobre una tabla donde otro usuario ha colocado bloqueos a nivel de fila. Sin embargo, esto no pone en peligro la integridad de la transacción, como se dice en Sección 15.10.10, “Detección de interbloqueos (deadlocks) y cancelación de transacciones (rollbacks)”. Consulte también Sección 15.16, “Restricciones de las tablas `InnoDB`”.

15.10.9. ¿Cuándo ejecuta o deshace implicitamente MySQL una transacción?

MySQL comienza cada conexión de cliente con el modo de ejecución automática (autocommit) habilitado por defecto. Cuando la ejecución automática está habilitada, MySQL realiza la confirmación luego de cada sentencia SQL, si dicha sentencia no devuelve un error.

Si se tiene desactivado el modo de ejecución automática y se cierra una conexión sin hacer una confirmación explícita de una transacción, MySQL cancelará dicha transacción.

Si una sentencia SQL devuelve un error, la confirmación o cancelación dependen del error. Consulte Sección 15.15, “Tratamiento de errores de InnoDB”.

Las siguientes sentencias SQL (y sus sinónimos) provocan en MySQL una confirmación implícita de la transacción en curso:

- `ALTER TABLE`, `BEGIN`, `CREATE INDEX`, `DROP DATABASE`, `DROP INDEX`, `DROP TABLE`, `LOAD MASTER DATA`, `LOCK TABLES`, `RENAME TABLE`, `SET AUTOCOMMIT=1`, `START TRANSACTION`, `TRUNCATE`, `UNLOCK TABLES`.
- Antes de MySQL 5.0.8, `CREATE TABLE` provocaba la confirmación si se empleaba el registro binario (binary logging). A partir de MySQL 5.0.8, las sentencias `CREATE TABLE`, `TRUNCATE TABLE`, `DROP DATABASE`, y `CREATE DATABASE` provocan una confirmación implícita.
- La sentencia `CREATE TABLE` en `InnoDB` se procesa como una transacción individual. Esto significa que un `ROLLBACK` emitido por el usuario no cancelará las sentencias `CREATE TABLE` hechas durante una transacción.

15.10.10. Detección de interbloqueos (deadlocks) y cancelación de transacciones (rollbacks)

`InnoDB` detecta automáticamente un deadlock de transacciones y cancela una o más transacciones para evitarlo. `InnoDB` intenta escoger para cancelar transacciones pequeñas, el tamaño de la transacción es determinado por el número de filas insertadas, actualizadas, o eliminadas.

`InnoDB` se mantiene al tanto de los bloqueos de tablas si `innodb_table_locks=1` (1 es el valor predeterminado), y la capa MySQL por debajo sabe acerca de bloqueos a nivel de fila. En otro caso, `InnoDB` no puede detectar deadlocks cuando están involucrados un bloqueo de tabla establecido por una sentencia `LOCK TABLES` o por otro motor de almacenamiento que no sea `InnoDB`. Estas situaciones se deben resolver estableciendo el valor de la variable de sistema `innodb_lock_wait_timeout`.

Cuando `InnoDB` lleva a cabo una cancelación completa de una transacción, todos los bloqueos de la transacción son liberados. Sin embargo, si solamente se cancela como resultado de un error una sentencia SQL individual, algunos de los bloqueos impuestos por la sentencia SQL podrían mantenerse. Esto se debe a que `InnoDB` guarda los bloqueos de fila en un formato en el que no puede luego saber qué sentencia SQL originó cada bloqueo.

15.10.11. Cómo tratar con interbloqueos

Los deadlocks son un problema clásico de las bases de datos transaccionales, pero no son peligrosos a menos que sean tan frecuentes que no se puedan ejecutar en absoluto ciertas transacciones. Normalmente, las aplicaciones deben ser escritas de modo que esten preparadas para emitir nuevamente una transacción si ésta es cancelada debido a un deadlock.

`InnoDB` emplea bloqueos automáticos a nivel de fila. Se pueden producir deadlocks aún en el caso de transacciones que solamente insertan o eliminan una fila individual. Esto se debe a que estas operaciones no son realmente “atómicas”; sino que establecen automáticamente bloqueos enlos (posiblemente varios) registros de índice de la fila insertada o eliminada.

Con las siguientes técnicas se puede estar a cubierto de los deadlocks y reducir la probabilidad de que ocurran:

- Emplear `SHOW INNODB STATUS` para determinar la causa del último deadlock. Puede ayudar a afinar la aplicación para evitar que ocurran otros.

- Siempre hay que estar preparado para emitir nuevamente una transacción que haya fallado por un deadlock. Los deadlocks no revisten peligro, simplemente hay que intentar de nuevo.
- Confirmar las transacciones frecuentemente. Las transacciones pequeñas son menos propensas a originar conflictos.
- Si se están usando lecturas que establecen bloqueos (`SELECT ... FOR UPDATE` o `... LOCK IN SHARE MODE`), hay que intentar utilizar un nivel de aislamiento bajo, como `READ COMMITTED`.
- Acceder a las tablas y filas en un orden fijo. Entonces, las transacciones forman secuencias bien definidas y no originan deadlocks.
- Agregar a las tablas índices adecuadamente elegidos. Entonces las consultas necesitarán examinar menos registros de índice y en consecuencia establecerán menos bloqueos. Utilizar `EXPLAIN SELECT` para determinar los índices que MySQL considera más apropiados para las consultas.
- Utilizar menos el bloqueo. Si es aceptable que `SELECT` devuelva datos de una captura de la base de datos que no sea la más actualizada, no hay que agregarle las cláusulas `FOR UPDATE` o `LOCK IN SHARE MODE`. En este caso es adecuado utilizar el nivel de aislamiento `READ COMMITTED`, porque cada lectura consistente dentro de la misma transacción leerá de su propia captura más reciente.
- Si nada de esto ayuda, habrá que serializar las transacciones con bloqueos a nivel de tabla. La forma correcta de emplear `LOCK TABLES` con tablas transaccionales, como InnoDB, es establecer `AUTOCOMMIT = 0` y no invocar a `UNLOCK TABLES` hasta que se haya confirmado explícitamente la transacción. Por ejemplo, si se necesitara escribir en una tabla `t1` y leer desde una tabla `t2`, se puede hacer esto:

  ```
  SET AUTOCOMMIT=0;
  LOCK TABLES t1 WRITE, t2 READ, ...;
  [aquí se hace algo con las tablas t1 y t2];
  COMMIT;
  UNLOCK TABLES;
  ```

  Los bloqueos a nivel de tabla favorecen el funcionamiento de la cola de transacciones, y evitan los deadlocks.
- Otra manera de serializar transacciones es crear una tabla “semáforo” auxiliar que contenga sólo una fila. Hay que hacer que cada transacción actualice esa fila antes de acceder otras tablas. De ese modo, todas las transacciones se producirán en serie. Nótese que el algoritmo de detección instantánea de deadlocks de `InnoDB` también funciona en este caso, porque el bloqueo de serialización es un bloqueo a nivel de fila. Con los bloqueos a nivel de tabla de MySQL, debe emplearse el método de timeout para solucionar deadlocks.
- En aquellas aplicaciones que emplean el comando de MySQL `LOCK TABLES`, MySQL no establece bloqueos de tabla si `AUTOCOMMIT=1`.

15.11. Consejos de afinamiento del rendimiento de InnoDB

- Si la utilidad `top` de Unix o el Administrador de Tareas de Windows muestra que el porcentaje de uso de CPU durante la carga de trabajo es inferior al 70%, probablemente se está trabajando directamente sobre el disco. Podría suceder que se estén produciendo excesivas confirmaciones de transacciones, o que el pool de buffer sea muy pequeño. Puede ser de ayuda incrementar el tamaño del buffer, pero no debe alcanzar ni superar el 80% del total de la memoria física del ordenador.
- Incluir varias modificaciones en una sola transacción. `InnoDB` debe descargar su registro (log) al disco cada vez que se confirma una transacción, si dicha transacción realiza modificaciones en la base

de datos. Dado que la velocidad de rotación de un disco es generalmente de 167 revoluciones por segundo, esto restringe el número de confirmaciones a la misma fracción de segundo si el disco no "engaña" al sistema operativo.

- Si es aceptable la pérdida de alguna de las últimas transacciones confirmadas, se puede establecer en `my.cnf` el parámetro `innodb_flush_log_at_trx_commit` a un valor de 0. `InnoDB` intenta descargar el registro (log) una vez por segundo en cualquier caso, aunque la descarga no está garantizada.

- Incrementar el tamaño de los ficheros de registro (log), incluso hasta equiparar el tamaño del pool de buffer. Cuando `InnoDB` ha colmado la capacidad de los ficheros de log, debe escribir los contenidos modificados desde el pool de buffer al disco en un punto de verificación. Los ficheros de log pequeños pueden causar escrituras en disco innecesarias. La desventaja de los ficheros de log grandes es que la recuperación demanda más tiempo.

- También el buffer del log debe ser suficientemente grande (en el orden de los 8MB).

- Emplear el tipo de columna `VARCHAR` en lugar de `CHAR` si se almacenarán cadenas de longitud variable o si la columna contendrá muchos valores `NULL`. Una columna `CHAR(N)` siempre utiliza *`N`* bytes para almacenar los datos, inclusive si la cadena es más corta o es `NULL`. Las tablas más pequeñas aprovechan mejor el espacio del pool de buffer y reducen las operaciones de E/S en disco.

  Cuando se utiliza `row_format=compact` (el formato de registro predeterminado para InnoDB en MySQL 5.0) y un conjunto de caracteres de longitud variable como `utf8` o `sjis`, `CHAR(N)` ocupará una cantidad variable de espacio, con un mínimo de *`N`* bytes.

- En algunas versiones de GNU/Linux y Unix, descargar ficheros a disco con la función de Unix `fsync()` (la cual es utilizada en forma predeterminada por `InnoDB`) y otros métodos similares, es sorprendentemente lento. Si no se está satisfecho con el rendimiento de las operaciones de escritura de la base de datos, se puede intentar establecer el valor de `innodb_flush_method` en `my.cnf` a `O_DSYNC`, si bien `O_DSYNC` parece ser más lento en otros sistemas.

- Durante el empleo del motor de almacenamiento InnoDB en arquitecturas Solaris 10 para x86_64 (AMD Opteron), es importante usar la opción `forcedirectio` al montar cualquier sistema de ficheros usado para almacenar los ficheros relacionados con InnoDB (el comportamiento predeterminado en Solaris 10/x86_64 es *no* utilizar esta opción al montar el sistema de ficheros). Si no se utiliza `forcedirectio` se producirá una seria degradación en la velocidad y rendimiento de InnoDB en esta plataforma.

- Al importar datos dentro de `InnoDB`, hay que asegurarse de que MySQL no tiene habilitado el modo de ejecución automática (autocommit) porque provocaría una descarga del log a disco en cada inserción. Para desactivar la ejecución automática durante la operación de importación, hay que encerrarla entre sentencias `SET AUTOCOMMIT` y `COMMIT`:

  ```
  SET AUTOCOMMIT=0;
  /* Sentencias de importación SQL ... */
  COMMIT;
  ```

  Si se utiliza la opción `--opt` con `mysqldump`, se obtienen ficheros de voclado que son rápidos de importar en una tabla `InnoDB`, incluso sin encerrarlos en las sentencias `SET AUTOCOMMIT` y `COMMIT`.

- Tener cuidado con las cancelaciones de inserciones masivas: `InnoDB` emplea el buffer de inserciones para reducir la cantidad de operaciones de E/S en disco durante las inserciones, pero ese mecanismo no tiene efecto en la cancelación. Una cancelación efectuada directamente sobre el disco puede tomar 30 veces el tiempo que insumen las correspondientes inserciones. Matar el proceso del servidor de bases de datos no es de ayuda, porque la cancelación recomienza al volver a iniciar el servidor. La única forma de librarse de una cancelación fuera de control es incrementar el tamaño del pool

de buffer para que la cancelación se haga sobre la CPU y se ejecute más rápidamente, o utilizar un procedimiento especial. Consulte Sección 15.8.1, “Forzar una recuperación”.

- También hay que tener cuidado con las operaciones de gran tamaño realizadas directamente sobre el disco. Hay que emplear `DROP TABLE` y `CREATE TABLE` para obtener una tabla vacía, no `DELETE FROM tbl_name`.

- Emplear la sintaxis de múltiples filas de `INSERT` para reducir la carga extra de la comunicación entre el cliente y el servidor si se necesita insertar muchos registros:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

  Esta sugerencia es válida para las inserciones en cualquier tipo de tabla, no solamente en `InnoDB`.

- Si se tienen restricciones `UNIQUE` en claves secundarias, se puede acelerar la importación desactivando temporalmente, durante la importación, la verificación de tales restricciones:

```
SET UNIQUE_CHECKS=0;
```

  En tablas grandes, esto ahorra una gran cantidad de operaciones de E/S en disco debido a que `InnoDB` puede emplear su buffer de inserción para escribir de una vez los registros de indice secundarios.

- Si se tienen restricciones `FOREIGN KEY` en las tablas, se puede acelerar la importación desactivando la verificación de claves foráneas durante la misma:

```
SET FOREIGN_KEY_CHECKS=0;
```

  Para tablas grandes, esto puede ahorrar gran cantidad de operaciones sobre el disco.

- Si a menudo se realizan consultas sobre tablas que no se actualizan con frecuencia, utilizar el cache de consultas:

```
[mysqld]
query_cache_type = ON
query_cache_size = 10M
```

15.11.1. SHOW INNODB STATUS y los monitores InnoDB

`InnoDB` incluye los Monitores `InnoDB` que muestran información relativa al estado interno de `InnoDB`. Se puede emplear la sentencia SQL `SHOW INNODB STATUS` para obtener la salida del Monitor `InnoDB` estándar en el cliente SQL utilizado. Esta información es útil para ajustes de rendimiento. (Si se usa el cliente SQL interactivo `mysql`, la salida es más legible si se reemplaza el punto y coma que usualmente termina cada sentencia por `\G`.) Para más información sobre los modos de bloqueo de `InnoDB` consulte Sección 15.10.1, “Modos de bloqueo `InnoDB`”.

```
mysql> SHOW INNODB STATUS\G
```

Otra forma de emplear los Monitores `InnoDB` es permitirles escribir datos contínuamente en la salida estándar del servidor `mysqld`. En este caso, no se envía la salida a los clientes. Cuando se activa, los Monitores `InnoDB` imprimen datos aproximadamente cada 15 segundos. La salida del servidor normalmente se dirige a un fichero `.err` en el directorio de datos de MySQL. Estos datos son útiles para ajustes de rendimiento. En Windows, se debe iniciar el servidor desde la linea de comandos de una ventana de consola con la opción `--console` si se desea dirigir la salida a la ventana en lugar de usar para ello el registro de errores.

La salida del Monitor incluye información de los siguientes tipos:

- Tablas y bloqueos de registros en uso por cada transacción activa.
- Esperas de transacciones debidas a bloqueos.
- Esperas de subprocesos debidas a semáforos.
- Solicitudes de E/S de ficheros pendientes.
- Estadísticas del pool de buffer.
- Actividad de descarga y mezcla del buffer de inserciones del subproceso principal de InnoDB.

Para que el Monitor estándar InnoDB escriba en la salida estándar de mysqld, se debe emplear la siguiente sentencia SQL:

```
CREATE TABLE innodb_monitor(a INT) ENGINE=INNODB;
```

El monitor puede detenerse emitiendo la siguiente sentencia:

```
DROP TABLE innodb_monitor;
```

La sintaxis de CREATE TABLE es solamente una forma de pasar un comando al motor InnoDB a través del intérprete SQL de MySQL: Lo único importante aquí es que la tabla se llame innodb_monitor y que sea una tabla InnoDB. La estructura de la tabla no es relevante para el MonitorInnoDB. si se apaga el servidor mientras el monitor se está ejecutando, y se desea iniciar el monitor nuevamente, se debe eliminar la tabla antes de emitir una nueva sentencia CREATE TABLE para iniciar el monitor. Esta sintaxis puede cambiar en una entrega futura de MySQL.

De la misma forma se puede emplear innodb_lock_monitor. Esto es lo mismo que innodb_monitor, con la excepción de que también proporciona abundante información sobre bloqueos. Un innodb_tablespace_monitor separado imprime una lista de los segmentos de ficheros creados existentes en el espacio de tablas y valida las estructuras de datos de asignación del espacio de tablas. Adicionalmente, hay un innodb_table_monitor con el que se pueden imprimir los contenidos del diccionario de datos interno de InnoDB.

Un ejemplo de la salida del Monitor InnoDB:

```
mysql> SHOW INNODB STATUS\G
*************************** 1. row ***************************
Status:
====================================
030709 13:00:59 INNODB MONITOR OUTPUT
====================================
Per second averages calculated from the last 18 seconds
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the semaphore:
X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 108462, OS waits 37964; RW-excl spins 681824, OS waits 375485
------------------------
LATEST FOREIGN KEY ERROR
------------------------
```

```
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831 inser
ting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
Foreign key constraint fails for table test/ibtest11a:
,
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`, `D`)
 ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
 0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2: len 4;
 hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N..;; 4: len 7; hex
 00000000c3e0a7; asc .......;; 5: len 4; hex 6b68446b; asc khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex 80000005; a
sc ....;; 2: len 3; hex 6b6864; asc khd;; 3: len 6; hex 0000111ef3eb; asc ......
;; 4: len 7; hex 800001001e0084; asc .......;; 5: len 3; hex 6b6864; asc khd;;
------------------------
LATEST DETECTED DEADLOCK
------------------------
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185, OS thread id 30733 inser
ting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t','e187358f'
,'g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d %H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138; asc a
a35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190, OS thread id 32782 inser
ting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL,'aa3572','','c3572','d6012t','', NULL,'
h396', NULL, NULL, 7.31,7.31,7.31,200)
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290251546 lock_mode X locks rec but not gap
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138; asc a
a35818;; 1:
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290251546 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 82 RECORD: info bits 0 0: len 7; hex 61613335373230; asc aa
35720;; 1:
*** WE ROLL BACK TRANSACTION (1)
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
Total number of lock structs in row lock hash table 70
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3491, OS thread id 42002
MySQL thread id 32, query id 4668737 localhost heikki
show innodb status
---TRANSACTION 0 290328384, ACTIVE 0 sec, process no 3205, OS thread id 38929 in
serting
1 lock struct(s), heap size 320
MySQL thread id 29, query id 4668736 localhost heikki update
```

```
insert into speedc values (1519229,1, 'hgjhjgghggjgjgjgjgjggjgjgjgjgjgggjgjgjlhh
gghgggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjfh
---TRANSACTION 0 290328383, ACTIVE 0 sec, process no 3180, OS thread id 28684 co
mmitting
1 lock struct(s), heap size 320, undo log entries 1
MySQL thread id 19, query id 4668734 localhost heikki update
insert into speedcm values (1603393,1, 'hgjhjgghggjgjgjgjgjggjgjgjgjgjgggjgjgjlh
hgghgggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjf
---TRANSACTION 0 290328327, ACTIVE 0 sec, process no 3200, OS thread id 36880 st
arting index read
LOCK WAIT 2 lock struct(s), heap size 320
MySQL thread id 27, query id 4668644 localhost heikki Searching rows for update
update ibtest11a set B = 'kHdkkkk' where A = 89572
------- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11a index PRIM
ARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656d756d00; asc
 supremum.;;
------------------
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831 ro
llback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190, OS thread id 32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538,'aa95666','bb','c95666','d9486t','e200498f
','g86814','h538',date_format('2001-04-03 12:54:22','%Y-%m-%d %H:%i'),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185, OS thread id 30733 in
serting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL,'aa42837','','c56319','d1719t','', NULL,
'h321', NULL, NULL, 7.31,7.31,7.31,200)
--------
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
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to   18 1212665295
Last checkpoint at  18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size   3200
```

```
Free buffers       110
Database pages     3074
Modified db pages  2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
1 row in set (0.05 sec)
```

Algunas notas acerca de la salida:

- Si la sección `TRANSACTIONS` informa sobre esperas por bloqueo, la aplicación puede tener conflictos causados por bloqueos. La salida también puede ayudar a determinar las razones de interbloqueos en transacciones.
- La seccion `SEMAPHORES` informa sobre esperas de subprocesos debidas a semáforos y brinda estadísticas de cuántas veces los subprocesos han debido esperar por un mutex o un semáforo rw-lock. Una gran cantidad de subprocesos esperando por semáforos puede ser el resultado de operaciones de E/S en disco, o conflictos dentro de `InnoDB`. Los conflictos pueden deberse a un intenso paralelismo en las consultas, o problemas en la planificación de subprocesos del sistema operativo. En tales situaciones puede ser de ayuda establecer `innodb_thread_concurrency` en un valor más bajo.
- La sección `BUFFER POOL AND MEMORY` proporciona estadísticas sobre las páginas leídas y escritas. A partir de estas cifras se puede calcular cuántas operaciones de E/S sobre ficheros de datos están realizando actualmente las consultas emitidas.
- La sección `ROW OPERATIONS` muestra lo que está haciendo el subproceso principal.

`InnoDB` envía su salida de diagnóstico a `stderr` o a ficheros en lugar de a `stdout` o a buffers de memoria de tamaño fijo, para evitar potenciales desbordamientos de buffer. Como efecto secundario, la salida de `SHOW INNODB STATUS` se escribe cada quince segundos en un fichero de estado. El nombre de este fichero es `innodb_status.pid`, donde *`pid`* es el ID del proceso del servidor. Este fichero se crea en el directorio de datos de MySQL. `InnoDB` borra el fichero durante un apagado normal del servidor. Si se producen caídas o terminaciones anormales del servidor, pueden quedar copias de estos ficheros que deberán ser eliminadas manualmente. Antes de eliminarlas, se las podría examinar para ver si contienen información útil relativa a la causa de las caídas. En MySQL 5.0, `innodb_status.pid` solamente se crea si se establece la opción de configuración `innodb_status_file=1`.

15.12. Implementación de multiversión

Debido a que `InnoDB` es una base de datos con multiversión, debe llevar información acerca de las versiones anteriores de una fila en el espacio de tablas. Esta información se almacena en una estructura de datos llamada Rollback Segment (Segmento de Cancelación) al igual que una estructura de datos análoga de Oracle.

Internamente, `InnoDB` agrega dos campos a cada fila almacenada en la base de datos. Un campo de 6 bytes indica el identificador de la última transacción que insertó o actualizó la fila. Además, una

eliminación se trata internamente como una actualización en la que un bit especial en la fila se establece a un valor que la señala como eliminada. Cada registro también contiene un campo de 7 bytes llamado "roll pointer" que apunta a una entrada del registro (log) de cancelación de modificaciones (undo) grabado en el segmento de cancelación (RollBack Segment). Si la fila fue actualizada, la entrada en el registro de cancelación de modificaciones (undo log) contiene la información necesaria para recrear el contenido de la fila tal como estaba antes de actualizarse.

`InnoDB` utiliza la información en el segmento de cancelación para deshacer los cambios durante la cancelación de una transacción. También la emplea para generar versiones anteriores de una fila en una lectura consistente.

Los registros de cancelación de modificaciones en el segmento de cancelación se dividen entre originados por inserciones (insert undo logs) y actualizaciones (update undo logs). Los insert undo logs se necesitan solamente para la cancelación de transacciones y se descartan tan pronto como se confirma la transacción. Los update undo logs se emplean también para lecturas consistentes, y pueden descartarse solamente cuando no quedan transacciones integrando una captura tomada por `InnoDB`, que en una lectura consistente podría necesitar esta información para reconstruir versiones anteriores de una fila.

Se deben confirmar las transacciones regularmente, incluyendo aquellas que solamente realizan lecturas consistentes. De otro modo, `InnoDB` no podrá descartar datos de los update undo logs, y el segmento de cancelación puede crecer en demasía, llenando el espacio de tablas.

El tamaño físico de una entrada en el registro de cancelación de cambios en el segmento de cancelación es generalmente menor que el de la correspondiente fila insertada o actualizada. Se puede emplear esta información para calcular el espacio necesario para el segmento de cancelación.

En el esquema de multiversión de `InnoDB`, una fila no es quitada inmediatamente de la base de datos cuando se la elimina mediante una sentencia SQL. Sólo cuando `InnoDB` pueda descartar la entrada en el registro de cancelación de modificaciones creada para la eliminación, procederá a quitar físicamente de la base de datos la fila y sus entradas en los índices. Esta operación se llama depuración (purge) y es bastante rápida, generalmente requiere el mismo tiempo que la sentencia SQL que realizó la eliminación.

En un escenario donde el usuario inserte y elimine filas aproximadamente en la misma proporción y en pequeños lotes, es posible que el subproceso de depuración comience a sufrir retrasos y la tabla crezca contínuamente, haciendo muy lenta cualquier operación que se realice sobre el disco. Incluso si una tabla contuviera sólo 10 MB de datos útiles, puede crecer hasta ocupar 10 GB con las filas “muertas”. En tal caso podría ser bueno limitar las operaciones de filas nuevas y asignar más recursos al subproceso de depuración. La opción de inicio (y también variable global configurable) `innodb_max_purge_lag` existe precisamente para este propósito. Consulte Sección 15.4, “Opciones de arranque de `InnoDB`” para más información.

15.13. Estructuras de tabla y de índice

MySQL almacena la información de su diccionario de datos de tablas en ficheros `.frm` dentro del directorio de cada base de datos. Esto es así para todos los motores de almacenamiento de MySQL, pero cada tabla `InnoDB` también tiene su propia entrada en los diccionarios de datos internos de `InnoDB` dentro del espacio de tablas. Cuando MySQL elimina una tabla o una base de datos, también debe eliminar uno o más ficheros `.frm`, y las correspondientes entradas dentro del diccionario de datos de `InnoDB`. Esta es la razón por la cual no se pueden mover tablas entre bases de datos sencillamente moviendo los ficheros `.frm`.

Cada tabla `InnoDB` tiene un índice especial llamado índice agrupado (clustered index) donde se almacenan los datos de las filas. Si se define una `PRIMARY KEY` en una tabla, el índice de la clave primaria es el índice agrupado.

Si no se define una `PRIMARY KEY` para la tabla, MySQL toma como clave primaria el primer índice `UNIQUE` que tenga solamente columnas `NOT NULL`, al cual `InnoDB` utiliza como índice agrupado. Si no hay en la tabla un índice con esas características, `InnoDB` generará internamente un índice agrupado donde las filas estarán ordenadas por el identificador de fila (row ID) que `InnoDB` asigna a las columnas en tal tabla. El identificador de fila es un campo de 6 bytes que se incrementa automáticamente a medida que se agregan nuevas filas. Por lo tanto, las filas ordenadas por este identificador están en el orden físico de inserción.

El acceso a una fila a través del índice agrupado es rápido porque la fila de datos se encuentra en la misma página a donde el índice dirige su búsqueda. Si una tabla es grande, la arquitectura del índice agrupado a menudo ahorra operaciones de E/S en disco en comparación a la solución tradicional. (En muchos servidores de bases de datos, los datos se suelen almacenar en una página diferente que la entrada del índice).

En `InnoDB`, las entradas en índices no agrupados (también llamados índices secundarios) contienen el valor de clave primaria de la fila. `InnoDB` utiliza este valor de clave primaria para buscar la fila a partir del índice agrupado. Nótese que si la clave primaria es larga, los índices secundarios utilizan más espacio.

`InnoDB` compara las cadenas `CHAR` y `VARCHAR` de diferente longitud como si el espacio sobrante en la cadena más corta estuviera relleno con espacios.

15.13.1. Estructura física de un índice

Todos los índices en `InnoDB` son árboles binarios (B-trees) donde las entradas de índice se almacenan en páginas que son las hojas del árbol. El tamaño predeterminado de una página de índice es 16KB. Cuando se insertan nuevas entradas, `InnoDB` intenta reservar 1/16 de la página para futuras inserciones y actualizaciones en el índice.

Si las entradas del índice se insertan en un orden secuencial (sea ascendente o descendente), las páginas de índice se llenarán en aproximadamente 15/16 de su capacidad. Si las entradas se insertan en un orden aleatorio, las páginas se llenarán entre 1/2 y 15/16 de su capacidad. Si la proporción de espacio ocupado de una página de índice cae por debajo de 1/2, `InnoDB` intentará reducir el árbol de índice para liberar la página.

15.13.2. Búfer de inserciones

Una situación común en una aplicación de bases de datos es que la clave primaria sea un identificador único y las nuevas filas se inserten en el orden ascendente de esta clave. Por lo tanto, las inserciones en el índice agrupado no necesitan lecturas aleatorias del disco.

Por el otro lado, los índices secundarios generalmente no son únicos, y las inserciones en los mismos ocurren en un orden relativamente aleatorio. Esto podría ocasionar gran cantidad de operaciones de E/S en disco, de no ser por un mecanismo especial utilizado en `InnoDB`.

Si una entrada de índice debiera insertarse en un índice secundario no único, `InnoDB` verifica si la página del índice secundario se encuentra en el pool de buffer. Si ese es el caso, `InnoDB` realizará la inserción directamente en la página de índice. Si dicha página no se encuentra en el pool de buffer, `InnoDB` insertará la entrada en una estructura de buffer de inserciones especial. El buffer de inserciones se mantiene tan pequeño que cabe completamente en el pool de buffer, y las inserciones pueden hacerse muy rápidamente.

Periódicamente, el buffer de inserciones se integra a los árboles de índices secundarios de la base de datos. A menudo es posible integrar varias inserciones en la misma página del árbol de índices, ahorrando operaciones de E/S en disco. Se ha comprobado que el buffer de inserciones puede acelerar hasta 15 veces las inserciones en una tabla.

La integración del buffer de inserciones puede continuar luego de que la transacción que realiza la inserción ha sido confirmada. De hecho, puede continuar despues de que el servidor ha sido detenido y vuelto a iniciar (consulte Sección 15.8.1, “Forzar una recuperación”).

La integración del buffer de inserciones puede tomar muchas horas, cuando hay varios índices secundarios para actualizar y gran cantidad de filas insertadas. Durante este tiempo, las operaciones de E/S en disco se incrementan, lo cual puede lentificar significativamente las consultas sobre el disco. Otra operación de E/S de importancia que se produce en segundo plano es el subproceso de depuración (Consulte Sección 15.12, “Implementación de multiversión”).

15.13.3. Indices hash adaptables

Si una tabla cabe casi completamente en la memoria principal, la manera más rápida de ejecutar consultas sobre ella es empleando índices hash. `InnoDB` posee un mecanismo automático que supervisa las búsquedas realizadas sobre los índices de una tabla. Si `InnoDB` advierte que las consultas se podrían beneficiar con la creación de un índice hash, lo hará automáticamente.

El índice hash siempre está basado en un índice B-tree existente en la tabla. `InnoDB` puede generar un indíce hash sobre un prefijo de la clave definida para el B-tree de cualquier longitud, dependiendo del patrón de búsquedas que `InnoDB` observe en el índice B-tree. Un índice hash puede ser parcial: no se necesita que el índice B-tree completo sea alojado en el pool de buffer. `InnoDB` genera índices hash según sea necesario basándose en las páginas de índice que son utilizadas más frecuentemente.

Podría decirse que, a través del mecanismo de índices hash adaptativos, `InnoDB` se adapta a la memoria principal, acercándose así a la arquitectura de las bases de datos de memoria.

15.13.4. Estructura física de los registros

Los registros de las tablas `InnoDB` tienen las siguientes características:

- Cada registro de índice en `InnoDB` contiene un encabezado de seis bytes. El encabezado se emplea para enlazar juntos registros consecutivos, y también en el bloqueo a nivel de fila.
- Los registros en el índice agrupado contienen campos para todas las columnas definidas por el usuario. Adicionalmente, hay un campo de seis bytes para el IDentificador de transacción y un campo de siete bytes para el roll pointer.
- Si no se definió una clave primaria para la tabla, cada registro de índice agrupado contiene también un campo de IDentificación de fila de seis bytes.
- Cada registro de índice secundario contiene también todos los campos definidos para la clave del índice agrupado.
- Un registro contiene además un puntero a cada campo del mismo. Si la longitud total de los campos en un registro es menos de 128 bytes, el puntero medirá un byte, de lo contrario, tendrá dos bytes de longitud. La matriz de estos punteros se conoce como el directorio de registros. El área a donde señalan estos punteros se denomina la parte de datos del registro.
- Internamente, `InnoDB` almacena las columnas de caracteres de longitud fija (como `CHAR(10)`) en un formato de longitud fija. `InnoDB` trunca los espacios sobrantes de las columnas `VARCHAR`. Nótese que MySQL puede convertir internamente columnas `CHAR` a `VARCHAR`. Consulte Sección 13.1.5.1, “Cambios tácitos en la especificación de columnas”.
- Un valor `NULL` SQL reserva 1 o 2 bytes en el directorio de registros. No reservará ningún byte en la parte de datos del registro si se lo almacena en una columna de longitud variable. En una columna de longitud fija, reservará en la parte de datos la longitud asignada a dicha columna. La razón por la que se

reserva este espacio fijo a pesar de tratarse de un valor `NULL`, es que en el futuro se podrá insertar en su lugar un valor no-`NULL` sin provocar la fragmentación de la página de índice.

15.14. Gestión de espacio de ficheros y de E/S de disco (Disk I/O)

15.14.1. E/S de disco (Disk I/O)

`InnoDB` emplea E/S en disco asíncrona simulada: `InnoDB` crea un número de procesos para hacerse cargo de las operaciones de E/S, tal como lectura por adelantado (read-ahead).

En `InnoDB` hay dos métodos de lectura por adelantado:

- En la lectura por adelantado secuencial, si `InnoDB` advierte que el patrón de acceso a un segmento en el espacio de tablas es secuencial, envía por adelantado al sistema de E/S un lote de lectura de páginas de base de datos.
- En la lectura por adelantado aleatoria, si `InnoDB` advierte que algún sector del espacio de tablas parece estar en proceso de ser completamente leido dentro del pool de búfer, envía las lecturas restantes al sistema de E/S.

`InnoDB` emplea una novedosa técnica de descarga de ficheros llamada *doublewrite*. La misma incrementa la seguridad en la recuperación que sigue a una caida del sistema operativo o una interrupción de energía eléctrica, y mejora el rendimiento en muchas variedades de Unix al reducir la necesidad de usar operaciones `fsync()`.

Doublewrite (escritura doble) significa que antes de escribir páginas en un fichero de datos, `InnoDB` las escribe primero en un área contigua del espacio de tablas llamada el búfer de doublewrite (o de escritura doble). Solamente luego de que la escritura y descarga al búfer de escritura doble se ha completado, `InnoDB` escribe las páginas en el sitio apropiado del fichero de datos. Si el sistema operativo colapsa en el transcurso de una escritura de página, `InnoDB`, posteriormente, durante la recuperación, podrá hallar una copia en buen estado en el búfer de escritura doble.

15.14.2. Usar dispositivos en bruto (raw devices) para espacios de tablas

En MySQL 5.0, se pueden usar particiones de dispositivos en bruto como ficheros de datos del espacio de tablas. Utilizando un dispositivo en bruto, se pueden llevar a cabo operaciones de E/S en Windows y algunas versiones de Unix sin que utilicen el búfer y sin la sobrecarga producida por el sistema de ficheros, lo cual incrementa el rendimiento.

Cuando se crea un nuevo fichero de datos, se debe colocar la palabra clave `newraw` inmediatamente a continuación del tamaño del fichero de datos en `innodb_data_file_path`. La partición deberá ser al menos tan grande como el tamaño que se haya especificado. Nótese que 1MB en `InnoDB` significa 1024 * 1024 bytes, en tanto que 1MB, en las especificaciones de los discos, generalmente significa 1.000.000 de bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

La próxima vez que se inicie el servidor, `InnoDB` advertirá la palabra clave `newraw` e inicializará la nueva partición. Sin embargo, aún no creará ni modificará ninguna tabla `InnoDB`. De lo contrario, la próxima vez que se reiniciase el servidor, `InnoDB` reinicializaría la partición y los cambios se perderían. (A partir de la versión 3.23.44, como medida de seguridad, `InnoDB` impide que los usuarios modifiquen datos cuando se especifica una partición con `newraw`.)

Después que InnoDB ha inicializado la nueva partición, hay que detener el servidor y cambiar newraw por raw en la linea que especifica el fichero de datos:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:5Graw;/dev/hdd2:2Graw
```

Luego, al reiniciar el servidor InnoDB permitirá realizar cambios.

En Windows puede asignarse una partición de disco como fichero de datos de este modo:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

Los caracteres //./ se corresponden con la sintaxis Windows de \\.\ para acceder dispositivos físicos.

Al emplear particiones de dispositivos en bruto, hay que cerciorarse de que la cuenta de usuario usada para ejecutar el servidor MySQL tiene permisos de lectura y escritura sobre ellas.

15.14.3. Gestión del espacio de ficheros

Los ficheros de datos definidos en el fichero de configuración forman el espacio de tablas de InnoDB. Los ficheros simplemente son concatenados para formar el espacio de tablas. No se utiliza striping (grabación de datos a través de varios discos en simultáneo). Actualmente no se puede definir en qué parte del espacio de tablas se ubicarán las tablas. Sin embargo, en un espacio de tablas nuevo, InnoDB asigna el espacio comenzando por el primer fichero de datos.

El espacio de tablas consiste en páginas de base de datos con un tamaño por defecto de 16KB. Las páginas se agrupan en áreas de 64 páginas consecutivas. Los “ficheros” dentro de un espacio de tablas se llaman *segmentos* en InnoDB. El término “segmento de cancelación” (rollback segment) es un tanto confuso porque en realidad contiene varios segmentos del espacio de tablas.

Por cada índice de InnoDB se asignan dos segmentos. Uno es para los nodos que no son hojas del B-tree, el otro es para los nodos hoja. La idea es mejorar la secuencialidad de los nodos hoja, los cuales contienen los datos.

Cuando un segmento crece dentro del espacio de tablas, InnoDB ubica las primeras 32 páginas individualmente. Luego de ello, comienza a ubicar áreas enteras en el segmento. InnoDB puede adicionar a un segmento grande hasta 4 áreas de páginas cada vez, para asegurar una adecuada secuencialidad de los datos.

Algunas páginas en el espacio de tablas contienen bitmaps de otras páginas, por lo tanto unas pocas áreas en un espacio de tablas InnoDB no puede asignarse a segmentos como un todo, sino solamente como páginas individuales.

Cuando se consulta el espacio libre disponible en el espacio de tablas mediante una sentencia SHOW TABLE STATUS, InnoDB informa las áreas que están totalmente libres en el espacio de tablas. InnoDB siempre reserva algunas áreas para depuración y otros propósitos internos; estas áreas reservadas no se cuentan en el espacio libre.

Cuando se eliminan datos de una tabla, InnoDB reduce los correspondientes índices B-tree. Depende del patrón seguido por las eliminaciones, si se liberan páginas individuales o áreas del espacio de tablas, de forma que el espacio desocupado quede disponible para otros usuarios. Eliminar una tabla, o todas las filas que contiene, seguramente servirá para liberar el espacio, pero no hay que olvidar que las

filas eliminadas solamente desaparecen físicamente cuando dejan de ser necesarias para cancelar transacciones o de integrar lecturas consistentes.

15.14.4. Desfragmentar una tabla

Si se producen inserciones o eliminaciones aleatorias en los índices de una tabla, los índices pueden resultar fragmentados. Esto significa que el orden físico de las páginas de índice en el disco no guarda relación con el orden de los registros en las páginas, o que hay muchas páginas en blanco en los bloques de 64 páginas que se asignan al índice.

Un síntoma de la fragmentación es que una tabla ocupa más espacio del que 'debería' ocupar. Es difícil determinarlo con exactitud, ya que todos los datos e índices en InnoDB se almacenan en estructuras B-tree, cuya porporción de espacio ocupado (fillfactor) puede variar entre el 50% y el 100%. Otro síntoma de fragmentación es que una consulta que examine toda la tabla:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

toma más tiempo del que debería. (En la consulta anterior, se ha “engañado” al optimizador SQL para que examine el índice agrupado, no un índice secundario). La mayoría de los discos pueden leer entre 10 y 50 MB por segundo. Esto puede usarse para estimar la velocidad con que debería examinarse una tabla.

Se puede acelerar el examen de los índices si periódicamente se lleva a cabo una operación `ALTER TABLE` “neutra”:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

Esto provoca que MySQL reconstruya la tabla. Otra forma de ejecutar una desfragmentación es emplear `mysqldump` para obtener un volcado de la tabla en un fichero de texto, eliminar la tabla, y volver a crearla a partir del fichero de volcado.

Si las inserciones en un índice se producen siempre en orden ascendente y los registros se eliminan solamente desde el final, el algoritmo de gestión de espacio en fichero que tiene `InnoDB` garantiza que no se produzca la fragmentación del índice.

15.15. Tratamiento de errores de InnoDB

El tratamiento de errores en `InnoDB` no siempre es como se especifica en el estándar SQL. De acuerdo a éste, cualquier error durante la ejecución de una sentencia SQL debería ocasionar su cancelación. `InnoDB` a veces sólo cancela una parte de la sentencia, o la transacción completa. Los siguientes puntos describen cómo `InnoDB` lleva a cabo el tratamiento de errores:

- Si el espacio de tablas agota su espacio disponible en disco, se obtiene el error de MySQL `Table is full` (La tabla está llena) e `InnoDB` cancela la sentencia SQL.
- Un interbloqueo (deadlock) en una transacción o un exceso de espera (timeout) en una espera por bloqueo provocan que `InnoDB` cancele la transacción completa.
- Un error de clave duplicada cancelará la sentencia SQL, si ésta no contiene la opción `IGNORE`.
- Un error `row too long error` (registro demasiado largo) cancela la sentencia SQL.
- Los demás errores son, en su mayoría, detectados por la capa de código MySQL (por encima del nivel del motor de almacenamiento `InnoDB`) y causarán la cancelación de la correspondiente sentencia SQL. Los bloqueos no se liberan al cancelar una única sentencia SQL.

Durante una cancelación implícita, así como durante la ejecución de un comando SQL ROLLBACK explícito, SHOW PROCESSLIST muestra Rolling back en la columna State de la conexión afectada.

15.15.1. Códigos de error de InnoDB

La siguiente es una lista con los errores específicos de InnoDB más comunes, con información acerca de su causa y su solución.

- 1005 (ER_CANT_CREATE_TABLE)

  No se puede crear la tabla. Si el mensaje del error hace referencia al errno 150, la creación de la tabla falló debido a una restricción de clave foránea incorrectamente formulada.

- 1016 (ER_CANT_OPEN_FILE)

  No se puede hallar la tabla InnoDB en los ficheros de datos, si bien existe el fichero .frm para esa tabla. Consulte Sección 15.17.1, “Resolver problemas de las operaciones del diccionario de datos de InnoDB”.

- 1114 (ER_RECORD_FILE_FULL)

  InnoDB se ha quedado sin lugar en el espacio de tablas. Se debería reconfigurar el espacio de tablas para agregar un nuevo fichero de datos.

- 1205 (ER_LOCK_WAIT_TIMEOUT)

  Expiró el tiempo de espera para realizar un bloqueo. La transacción se canceló.

- 1213 (ER_LOCK_DEADLOCK)

  Ocurrió un deadlock durante una transacción. Deberá ser repetida.

- 1216 (ER_NO_REFERENCED_ROW)

  Se está intentando agregar una fila, pero no hay una fila padre, lo que hace que una restricción de clave foránea falle. Se debe insertar antes la fila padre.

- 1217 (ER_ROW_IS_REFERENCED)

  Se está intentando eliminar una fila padre que tiene filas hijas, lo que hace que una restricción de clave foránea falle. Se deben eliminar primero las filas hijas.

15.15.2. Códigos de error del sistema operativo

Para ver el significado de un número de error del sistema operativo, se utiliza el programa perror, que viene con la distribución de MySQL.

La siguiente tabla proporciona una lista con algunos códigos de error de sistema comunes en Linux. Para una lista más completa comsulte El código fuente de Linux.

- 1 (EPERM)

  Operación no permitida

- 2 (ENOENT)

  No existe el fichero o directorio

- `3 (ESRCH)`

  No existe el proceso

- `4 (EINTR)`

  Llamada de sistema interrumpida

- `5 (EIO)`

  Error de E/S

- `6 (ENXIO)`

  No existe el dispositivo o dirección

- `7 (E2BIG)`

  Lista de argumentos demasiado extensa

- `8 (ENOEXEC)`

  Eror de formato ejecutable

- `9 (EBADF)`

  Número de fichero erróneo

- `10 (ECHILD)`

  No hay procesos hijos

- `11 (EAGAIN)`

  Intente nuevamente

- `12 (ENOMEM)`

  Memoria agotada

- `13 (EACCES)`

  Permiso denegado

- `14 (EFAULT)`

  Dirección errónea

- `15 (ENOTBLK)`

  Se necesita un bloque de dispositivo

- `16 (EBUSY)`

  El dispositivo o recurso está ocupado

- `17 (EEXIST)`

  El fichero ya existe

- `18 (EXDEV)`

  Vínculo de dispositivos cruzado (Cross-device link)

- `19 (ENODEV)`

  No existe el dispositivo

- `20 (ENOTDIR)`

  No es un directorio

- `21 (EISDIR)`

  Es un directorio

- `22 (EINVAL)`

  Argumento inválido

- `23 (ENFILE)`

  Desbordamiento de tabla de fichero

- `24 (EMFILE)`

  Demasiados ficheros abiertos

- `25 (ENOTTY)`

  Ioctl no apropiada para el dispositivo

- `26 (ETXTBSY)`

  Fichero de texto ocupado

- `27 (EFBIG)`

  El fichero es demasiado grande

- `28 (ENOSPC)`

  Espacio agotado en el dispositivo

- `29 (ESPIPE)`

  Búsqueda ilegal

- `30 (EROFS)`

  Fichero de sistema de sólo lectura

- `31 (EMLINK)`

  Demasiados vínculos

La siguiente tabla proporciona una lista con algunos códigos de error de sistema comunes en Windows. Para una lista completa consulte el Microsoft sitio web.

- `1 (ERROR_INVALID_FUNCTION)`

Función incorrecta

- `2 (ERROR_FILE_NOT_FOUND)`

  El sistema no puede hallar el fichero especificado

- `3 (ERROR_PATH_NOT_FOUND)`

  El sistema no puede hallar la ruta especificada

- `4 (ERROR_TOO_MANY_OPEN_FILES)`

  El sistema no puede abrir el fichero.

- `5 (ERROR_ACCESS_DENIED)`

  Acceso denegado.

- `6 (ERROR_INVALID_HANDLE)`

  El manejador es inválido.

- `7 (ERROR_ARENA_TRASHED)`

  Los bloques de control de almacenamiento fueron destruidos.

- `8 (ERROR_NOT_ENOUGH_MEMORY)`

  No hay suficiente almacenamiento disponible para procesar este comando.

- `9 (ERROR_INVALID_BLOCK)`

  La dirección del bloque de control de almacenamiento es inválida.

- `10 (ERROR_BAD_ENVIRONMENT)`

  El entorno es incorrecto.

- `11 (ERROR_BAD_FORMAT)`

  Se intentó cargar un programa con un formato incorrecto.

- `12 (ERROR_INVALID_ACCESS)`

  El código de acceso es inválido.

- `13 (ERROR_INVALID_DATA)`

  El dato es inválido.

- `14 (ERROR_OUTOFMEMORY)`

  No hay suficiente espacio de almacenamiento para completar esta operación.

- `15 (ERROR_INVALID_DRIVE)`

  El sistema no puede hallar la unidad especificada.

- `16 (ERROR_CURRENT_DIRECTORY)`

El directorio no puede eliminarse.

- `17 (ERROR_NOT_SAME_DEVICE)`

  El sistema no puede mover el fichero a una unidad de disco diferente.

- `18 (ERROR_NO_MORE_FILES)`

  No hay más ficheros.

- `19 (ERROR_WRITE_PROTECT)`

  El medio de almacenamiento eatá protegido contra escritura.

- `20 (ERROR_BAD_UNIT)`

  El sistema no puede hallar el dispositivo especificado.

- `21 (ERROR_NOT_READY)`

  El dispositivo no está listo.

- `22 (ERROR_BAD_COMMAND)`

  El dispositivo no reconoce el comando.

- `23 (ERROR_CRC)`

  Error de datos (verificación de redundancia cíclica)

- `24 (ERROR_BAD_LENGTH)`

  El programa emitió un comando pero la longitud del comando es incorrecta.

- `25 (ERROR_SEEK)`

  La unidad no puede hallar un área o pista específica en el disco.

- `26 (ERROR_NOT_DOS_DISK)`

  No puede accederse al disco o diskette especificado.

- `27 (ERROR_SECTOR_NOT_FOUND)`

  La unidad no puede hallar el sector solicitado.

- `28 (ERROR_OUT_OF_PAPER)`

  La impresora no tiene papel.

- `29 (ERROR_WRITE_FAULT)`

  El sistema no puede escribir en la unidad especificada.

- `30 (ERROR_READ_FAULT)`

  El sistema no puede leer desde la unidad especificada.

- `31 (ERROR_GEN_FAILURE)`

Un dispositivo conectado al sistema está fuera de funcionamiento.

- 32 (ERROR_SHARING_VIOLATION)

  El proceso no puede acceder al fichero porque está en uso por otro proceso.

- 33 (ERROR_LOCK_VIOLATION)

  El proceso no puede acceder al fichero porque una parte fue bloqueada por otro proceso.

- 34 (ERROR_WRONG_DISK)

  La unidad contiene el diskette incorrecto. Inserte %2 (Número de Serie de Volumen: %3) en la unidad %1.

- 36 (ERROR_SHARING_BUFFER_EXCEEDED)

  Demasiados ficheros abiertos en modo compartido.

- 38 (ERROR_HANDLE_EOF)

  Se alcanzó el fin del fichero.

- 39 (ERROR_HANDLE_DISK_FULL)

  El disco está lleno.

- 87 (ERROR_INVALID_PARAMETER)

  El parámetro es incorrecto. (Si se obtiene este error en Windows, y se ha establecido innodb_file_per_table en my.cnf o my.ini, entonces debe agregarse la línea innodb_flush_method=unbuffered en el fichero my.cnf o my.ini.)

- 112 (ERROR_DISK_FULL)

  El disco está lleno.

- 123 (ERROR_INVALID_NAME)

  El nombre de fichero, de directorio, o la etiqueta de volumen, tienen la sintaxis incorrecta.

- 1450 (ERROR_NO_SYSTEM_RESOURCES)

  No hay suficientes recursos de sistema para completar el servicio requerido.

15.16. Restricciones de las tablas InnoDB

- Una tabla no puede contener más de 1000 columnas.
- La longitud máxima interna de una clave es 3500 bytes, pero MySQL la restringe a 1024 bytes.
- La longitud máxima de fila, excepto para columnas VARCHAR, BLOB y TEXT, es ligeramente inferior a la mitad de una página de base de datos. Es decir, cerca de 8000 bytes. Las columnas LONGBLOB y LONGTEXT deben ser de menos de 4GB, y la longitud total de la fila, incluyendo las columnas BLOB y TEXT, debe ser de menos de 4GB. InnoDB almacena los primeros 768 bytes de una columna VARCHAR, BLOB, o TEXT en la fila, y el resto, en páginas separadas.
- En algunos sistemas operativos antiguos, los ficheros de datos deben ser de menos de 2GB.

- El tamaño combinado de los ficheros de log de InnoDB debe ser inferior a 4GB.
- El tamaño mínimo del espacio de tablas es de 10MB. El tamaño máximo es de cuatrocientos mil millones de páginas de base de datos (64TB). Este es también el tamaño máximo para una tabla.
- Las tablas InnoDB no admiten índices FULLTEXT.
- Las tablas InnoDB no admiten tipos de columna espaciales.
- ANALYZE TABLE determina la cardinalidad efectuando 10 accesos al azar en cada uno de los árboles de índices y actualizando la cardinalidad del índice con una estimación acorde. Dado que son solamente estimaciones, distintas ejecuciones de ANALYZE TABLE pueden producir resultados diferentes. Esto convierte a ANALYZE TABLE en una herramienta rápida sobre tablas InnoDB, pero no con el mismo nivel de exactitud que si considerara todas las filas al hacer el recuento.

  MySQL emplea las estimaciones de cardinalidad de los índices solamente para la optimización de uniones. Si una unión no se optimiza en la manera adecuada, se puede intentar el uso de ANALYZE TABLE. En los pocos casso en que ANALYZE TABLE no produce valores suficientemente buenos para las tablas, se puede emplear FORCE INDEX en las consultas para forzar el uso de un índice en particular, o establecer el valor de max_seeks_for_key para asegurarse de que MySQL dará preferencia a las búsquedas en índices por sobre el examen de las tablas. Consulte Sección 5.3.3, “Variables de sistema del servidor”. Consulte Sección A.6, “Cuestiones relacionadas con el optimizador”.
- En Windows, InnoDB siempre almacena internamente en minúsculas los nombres de tablas y bases de datos. Para mover bases de datos en formato binario desde Unix a Windows o a la inversa, se deberán haber escrito en minúsculas todos los nombres de tablas y bases de datos.
- **Advertencia:** ¡*No* deben convertirse las tablas de sistema de MySQL de la base de datos mysql desde su formato original MyISAM a InnoDB! Esta es una operación no admitida. Si se lleva a cabo, MySQL no se podrá ejecutar hasta que se recuperen las tablas de sistema anteriores desde una copia de respaldo o se las regenere con el script mysql_install_db.
- InnoDB no lleva una cuenta interna de las filas en una tabla. (Esto sería realmente complicado a causa de la multiversión). Para procesar una sentencia SELECT COUNT(*) FROM T, InnoDB debe examinar un índice de la tabla, lo cual lleva algún tiempo si el índice no está completamente dentro del pool de buffer. Para disponer de un recuento más rápido, se debe crear una tabla de recuento y hacer que la aplicación la actualice a medida que se producen inserciones y eliminaciones. Si una tabla no se modifica a menudo, utilizar el cache de consultas (query cache) de MySQL es una buena solución. También puede emplearse SHOW TABLE STATUS si es suficiente un recuento aproximado de filas. Consulte Sección 15.11, “Consejos de afinamiento del rendimiento de InnoDB”.
- Para una columna AUTO_INCREMENT, siempre se debe definir un índice para la tabla, el cual debe contener solamente a la columna AUTO_INCREMENT. En tablas MyISAM, la columna AUTO_INCREMENT puede formar parte de un índice junto a otras columnas.
- InnoDB no admite la opción AUTO_INCREMENT en sentencias CREATE TABLE o ALTER TABLE, la cual sirve para establecer el valor inicial de la secuencia. Para especificar este valor en InnoDB, debe insertarse una fila con un valor que sea uno menos que el deseado, y luego borrarla, o insertar la primera fila especificando un valor determinado.
- Luego de reiniciar el servidor MySQL, InnoDB puede reutilizar un valor antiguo para una columna AUTO_INCREMENT (esto es, un valor que se hubiese asignado a una transacción finalmente cancelada).
- Cuando una columna AUTO_INCREMENT sobrepasa el máximo valor que es capaz de almacenar, InnoDB coloca la columna en -9223372036854775808 (si es BIGINT) o en 1 (si es BIGINT UNSIGNED). Sin embargo, como los valores BIGINT tienen 64 bits, hay que notar que si se insertara

un millón de filas por segundo, se demoraría cerca de trescientos mil años en agotar los números disponibles. Con otros tipos de columnas enteros, ocurre un error de clave duplicada. Esto es similar al funcionamiento de MyISAM, ya que es en mayor medida el comportamiento general de MySQL y no pertenece a ningún motor de almacenamiento en particular.

- DELETE FROM *nom_tabla* no regenera la tabla sino que elimina todas sus filas, una por una.
- TRUNCATE *tbl_name* se implementa en InnoDB como DELETE FROM *tbl_name* y no inicializa el contador de AUTO_INCREMENT.
- SHOW TABLE STATUS no proporciona estadísticas precisas en tablas InnoDB, excepto para el tamaño físico reservado por la tabla. El recuento de filas es solamente una estimación utilizada para la optimización SQL.
- En MySQL 5.0, la operación LOCK TABLES establece dos bloqueos en cada tabla si innodb_table_locks=1, que es el valor por defecto. Adicionalmente al bloqueo de tabla en la capa MySQL, también se establece un bloqueo de tabla en InnoDB. En versiones antiguas de MySQL no se establecía el bloqueo en InnoDB, para volver a este comportamiento debe especificarse innodb_table_locks=0. Si no se establece el bloqueo InnoDB, LOCK TABLES se completa aún cuando algunos registros de las tablas estén bloqueados por otras transacciones.
- Todos los bloqueos InnoDB efectuados por una transacción se liberan cuando la transacción se confirma o se cancela. Por lo tanto, no tiene mucho sentido invocar LOCK TABLES en tablas InnoDB cuando se está en el modoAUTOCOMMIT=1, porque los bloqueos establecidos sobre una tabla InnoDB se liberarán inmediatamente.
- Algunas veces sería útil bloquear tablas extensas en el curso de una trasacción. Desafortunadamente, LOCK TABLES, en MySQL, emite implícitamente un COMMIT y un UNLOCK TABLES. Está planeada una variante para InnoDB de LOCK TABLES que puede ejecutarse dentro de una transacción.
- La sentencia LOAD TABLE FROM MASTER empleada para la replicación de servidores esclavos no funciona aún con tablas InnoDB. Una solución temporal es cambiar a MyISAMla tabla en el servidor amo (master), efectuar la carga, y volver a cambiar la tabla en el amo (master) a su motor original InnoDB.
- El tamaño por defecto de cada página de base de datos en InnoDB es de 16KB. Se puede establecer en valores entre 8KB y 64KB recompilando el código. Se deben modificar los valores de UNIV_PAGE_SIZE y UNIV_PAGE_SIZE_SHIFT en el fichero fuente univ.i.
- En MySQL 5.0, los disparadores (triggers) aún no son activados por modificaciones efectuadas en cascada a través de claves foráneas.

15.17. Resolver problemas relacionados con InnoDB

- Por regla general, cuando una operación falla o se tienen sospechas de un error, se debe inspeccionar el log de errores del servidor MySQL, que normalmente tiene un nombre como *nombre_host*.err, o posiblemente mysql.err en Windows.
- Durante la resolución de problemas, usualmente es mejor ejecutar el servidor MySQL desde la línea de comandos, en lugar de utilizar mysqld_safe o como servicio de Windows. Ejecutándolo como se indica, se podrán ver los mensajes que mysqld imprime en la pantalla, y hacerse una mejor idea de lo que está sucediendo. En Windows, el servidor debe iniciarse con la opción --console para que la salida se dirija a la ventana de DOS utilizada.
- Pueden utilizarse los Monitores InnoDB para obtener información sobre un problema. Si el problema está relacionado con el rendimiento, o el servidor parece estar congelado, se debería utilizar

innodb_monitor para ver información acerca del estado interno de InnoDB. Si el problema es con bloqueos, debe utilizarse innodb_lock_monitor. Si el problema es en la creación de tablas u otra operación del diccionario de datos, debe emplearse innodb_table_monitor para imprimir los contenidos del diccionario de datos interno de InnoDB.

- Si se sospecha que una tabla está corrupta, hay que ejecutar CHECK TABLE sobre ella.

15.17.1. Resolver problemas de las operaciones del diccionario de datos de InnoDB

Un problema específico de las tablas es que el servidor MySQL mantiene la información relativa al diccionario de datos dentro de ficheros .frm que guarda en los directorios de las bases de datos, en tanto que InnoDB también almacena la información dentro de su propio diccionario de datos, en el interior de los ficheros de espacio de tablas. Si se mueven los ficheros .frm o si el servidor sufre una caída durante una operación de diccionario de datos, los ficheros .frm pueden quedar con diferencias respecto al diccionario de datos interno de InnoDB.

Un síntoma de que ha ocurrido esto es si falla una sentencia CREATE TABLE. Si esto sucede, se debería observar el registro (log) de errores del servidor. Si el registro indica que la tabla ya existía dentro del diccionario de datos interno de InnoDB, se tiene una tabla que ha quedado únicamente dentro de los ficheros de espacio de tablas de InnoDB y que no tiene el correspondiente fichero .frm. El mensaje de error tiene este aspecto:

```
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.
```

Se puede eliminar la tabla que causa el conflicto siguiendo las instrucciones del mensaje de error. Esto es, crear una tabla InnoDB con el mismo nombre en otra base de datos y mover al directorio de la base de datos actual el fichero .frm resultante. MySQL asumirá que la tabla ya existe, y se podrá eliminar con DROP TABLE. creará.

Otro síntoma de un diccionario de datos desactualizado es que MySQL emite un mensaje de error donde dice que no puede abrir un fichero .InnoDB:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

En el registro de errores puede encontrarse un mensaje similar a este:

```
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?
```

Esto significa que hay un fichero .frm que no tiene la correspondiente tabla dentro de InnoDB. El fichero .frm puede ser borrado manualmente para solucionarlo.

Si MySQL cae durante una operación ALTER TABLE, puede aparecer una tabla temporal huérfana dentro del espacio de tablas InnoDB. Empleando innodb_table_monitor se verá listada una tabla cuyo

nombre es #sql-... . En MySQL 5.0, se pueden llevar a cabo sentencias SQL sobre tablas cuyo nombre contenga el caracter '#' si se encierra el nombre dentro de acentos graves (ASCII 96). De esa forma, se puede eliminar esta tabla huérfana del mismo modo que las mencionadas anteriormente. Hay que tener en cuenta que al copiar o renombrar un fichero en el shell de Unix, se necesitará colocar el nombre del fichero entre comillas dobles si éste contiene un caracter '#'.

Capítulo 16. MySQL Cluster

Tabla de contenidos

MySQL Cluster es una versión de alta disponibilidad, alta redundancia de MySQL adaptada para el entorno de computación distribuida. Usa el motor de almacenamiento **NDB Cluster** para permitir la ejecución de varios servidores MySQL en un cluster. Este motor de almacenamiento está disponible en las distribuciones binarias de MySQL 5.0 y en los RPMs compatibles con las distribuciones Linux más modernas. (Tenga en cuenta que tanto los RPMs `mysql-server` como `mysql-max` deben instalarse para tener la capacidad de MySQL Cluster .)

Lo sistemas operativos en que MySQL Cluster está disponible son Linux, Mac OS X, y Solaris. (Agunos usuarios han reportado éxito al ejecutar MySQL Cluster en FreeBSD, aunque no está soportada oficialmente por MySQL AB.) Estamos trabajando en hacer que Cluster se ejecute en todos los sistemas operativos soportados por MySQL, incluyendo Windows, y actualizaremos esta página en cuanto se soporten nuevas plataformas.

Este capítulo representa el trabajo en progreso, y sus contenidos son objeto de revisión mientras MySQL Cluster evoluciona. Información adicional acerca de MySQL Cluster puede encontrarse en la web de MySQL AB http://www.mysql.com/products/cluster/.

Puede usar dos recursos en línea adicionales proporcionados por MySQL AB:

- Lista de correo de MySQL Cluster.
- El área de Cluster del MySQL User Forums.

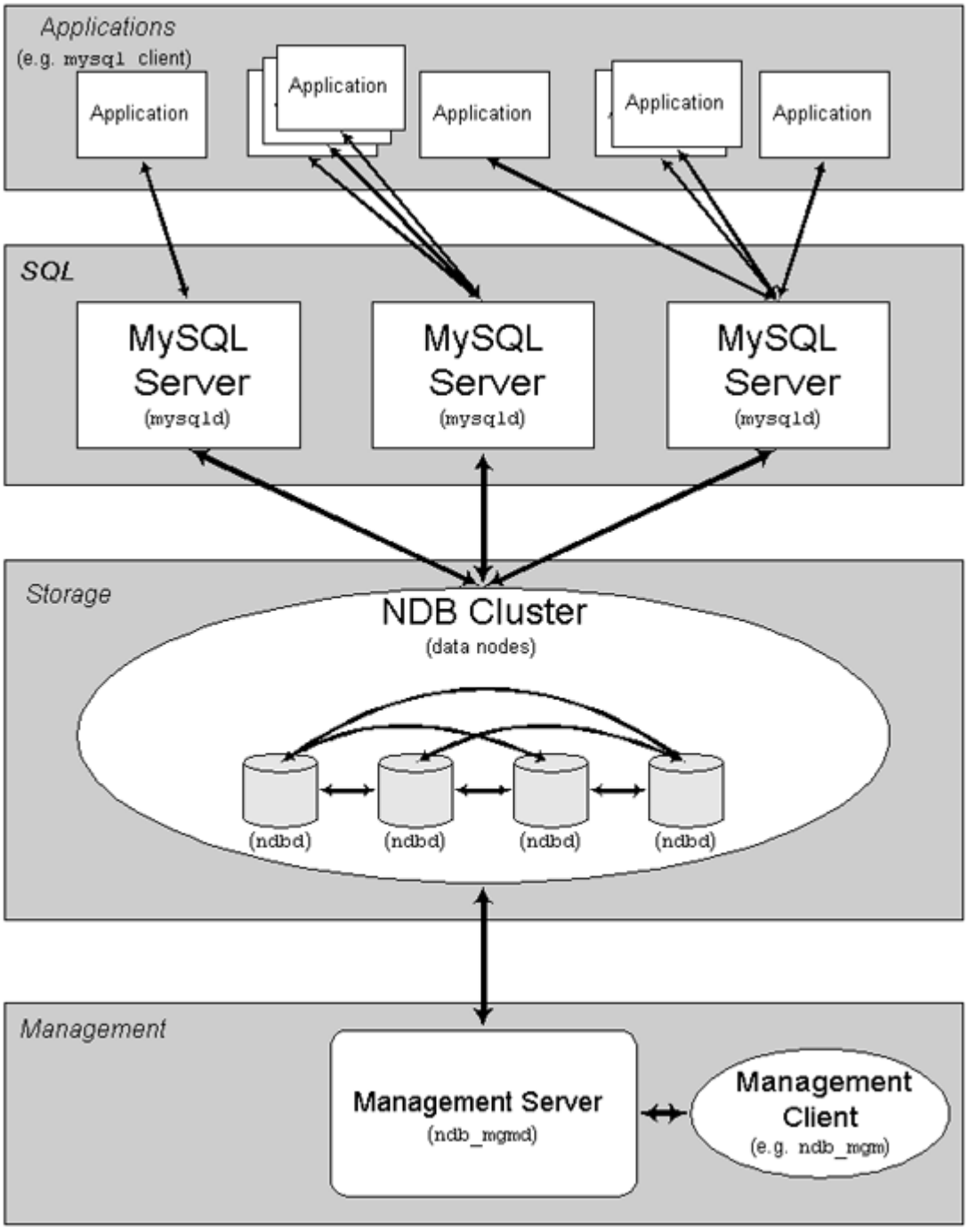
Puede encontrar respuestas a preguntas comunes acerca de Cluster en Sección 16.10, “Preguntas frecuentes sobre MySQL Cluster”. Si es novato en MySQL Cluster, puede encontrar útil el artículo de la Developer Zone How to set up a MySQL Cluster for two servers .

16.1. Panorámica de MySQL Cluster

MySQL Cluster es una tecnología que permite clustering de bases de datos en memoria en un entorno de no compartición. La arquitectura de no compartición permite que el sistema funcione con hardware barato, y sin ningún requerimiento especial de hardware o software. Tampoco tienen ningún punto único de fallo porque cada componente tiene su propia memoria y disco.

MySQL Cluster integra el servidor MySQL estándar con un motor de almacenamiento clusterizado en memoria llamado **NDB**. En nuestra documentación, el término **NDB** se refiere a la parte de la inicialización específica al motor de almacenamiento, mientras que **MySQL Cluster** se refiere a la combinación de MySQL y el nuevo motor de almacenamiento.

Un MySQL Cluster consiste en un conjunto de máquinas, cada una ejecutando un número de procesos incluyendo servidores MySQL , nodos de datos para NDB Cluster, servidores de administración, y (posiblemente) programas especializados de acceso a datos. La relación de estos componentes en un cluster se muestra aquí:



Todos estos programas funcionan juntos para formar un MySQL Cluster. Cuando se almacenan los datos en el motor NDB Cluster, las tablas se almacenan en los nodos de datos. Tales tablas son directamente accesibles desde todos los otros servidores MySQL en el cluster. Por lo tanto, en una aplicación de pago que almacene datos en un cluster, si una aplicación actualiza el salario de un empleado, todos los otros servidores MySQL que acceden a estos datos pueden ver el cambio inmediatamente.

Los datos almacenados en los nodos de datos de MySQL Cluster pueden replicarse: el cluster puede tratar fallos de nodos de datos individuales sin otro impacto a parte de abortar unas pocas transacciones debido a la pérdida de estado de transacción. Como las aplicaciones transaccionales se suponen que tratan fallos transaccionales, esto no debería ser un problema.

Al llevar MySQL Cluster al mundo Open Source , MySQL propociona tratamiento de datos clusterizado con alta disponibilidad, alto rendimiento, y escalabilidad disponible para todo el que lo necesite.

16.2. Conceptos básicos de Basic MySQL Cluster

NDB es un motor de almacenamiento en memoria que ofrece alta disponibilidad y caracterísitcas de persistencia de datos.

El motor NDB puede configurarse con un rango de opciones de fallo y balanceo de carga, pero es más sencillo arrancarlo con el motor de almacenamiento a nivel de cluster. El motor de MySQL Cluster NDB contiene un conjunto completo de datos, dependiente sólo de otros datos dentro del propio cluster.

Ahora describiremos cómo inicializar un MySQL Cluster consistente de un motor NDB y algunos servidores MySQL .

La porción de cluster del MySQL Cluster está configurada independientemente de los servidores MySQL . En MySQL Cluster, cada parte del cluster se considera como un **nod**.

Nota: En muchos contextos, el término "nodo" se usa para indicar una máquina, pero cuando se discute MySQL Cluster significa un *proceso*. Puede haber cualquier número de nodos en una máquina, para los que se usa el término **máquina cluster**.

Hay tres tipos de nodos cluster, y en una configuración MySQL Cluster mínima, al menos habrán tres nodos, uno de cada tipo:

- El nodo de administración (`MGM`) : El rol de este tipo de nodo es administrar los otros nodos dentro del MySQL Cluster, tal como proporcionar datos de configuración, iniciar y parar nodos, ejecutar copias de seguridad, y así. Como este tipo de nodo administra la configuración de otros nodos, un nodo de este tipo debe arrancarse primero, antes de cualquier otro nodo. Un nodo MGM se arranca con el comando `ndb_mgmd`.

- El **nodo de datos**: Este es el tipo de nodo que almacena los datos del cluster. Hay tantos nodos de datos como réplicas, multiplicado por el número de fragmentos. Por ejemplo, con dos réplicas, cada uno teniendo dos fragmentos, necesita cuatro nodos de datos. No es necesario tener más de una réplica. Un nodo de datos se arranca con el comando `ndbd`.

- El **nodo SQL**: Este es el nodo que accede a los datos del cluster. En el caso de MySQL Cluster, un nodo cliente es un servidor MySQL tradicional que usa el motor NDB Cluster . Un nodo SQL típicamente se arranca con el comando `mysqld --ndbcluster` o símplemente usando `mysqld` con `ndbcluster` añadido a `my.cnf`.

La configuración de un cluster implica configurar cada nodo individual en el cluster y inicializar los enlaces de comunicación individual entre los nodos. MySQL Cluster está diseñado con la intención que los nodos de almacenamiento son homogéneos en términos de procesador, espacio de memoria, y ancho de banda. Además, para proporcionar un punto único de configuración, todos los datos de configuración del cluster entero se guardan en un único fichero de configuración.

El servidor de administración (nodo MGM) administra el fichero de configuración del cluster y el log. Cada nodo en el cluster recibe los datos de configuración del servidor de administración, y necesita una forma de determinar dónde reside el servidor de administración. Cuando ocurren eventos interesantes en los nodos de datos, los nodos transfieren información acerca de estos eventos al servidor de administración, que guarda la información en el log del cluster.

Además, puede haber cualquier número de procesos clientes del cluster o aplicaciones. Hay de dos tipos:

- **Clientes MySQL estándar**: No son diferentes para MySQL Cluster que para cualquier MySQL (no cluster). En otras palabras,MySQL Cluster puede ser accedido para aplicaciones MySQL existentes escritas en PHP, Perl, C, C++, Java, Python, Ruby, y así.
- **Clientes de administración**: Estos clientes conectan al servidor de administración y proporcionan comandos para arrancar y parar nodos, arrancar y parar traceo de mensajes (sólo en versiones de depuración), mostrar versiones y estatus de nodos, arrancar y parar copias de seguridad, y así.

16.3. Cómo configurar varios ordenadores

Esta sección es un “Cómo” (“How-To”) en el que describimos las bases para planear, instalar, configurar, y ejecutar un MySQL Cluster. A diferencia del ejemplo de Sección 16.4, “Configuración de MySQL Cluster”, el resultado de las guías y procedimientos descritos a continuación deben ser utilizables para MySQL Cluster con unos requerimientos mínimos para disponibilidad y salvaguardia de los datos.

En esta sección, cubrimos requerimientos de hardware y software; red; instalación de MySQL Cluster; configuración; arrancar, parar y reiniciar el cluster; cargar una base de datos de ejemplo; y realizar consultas.

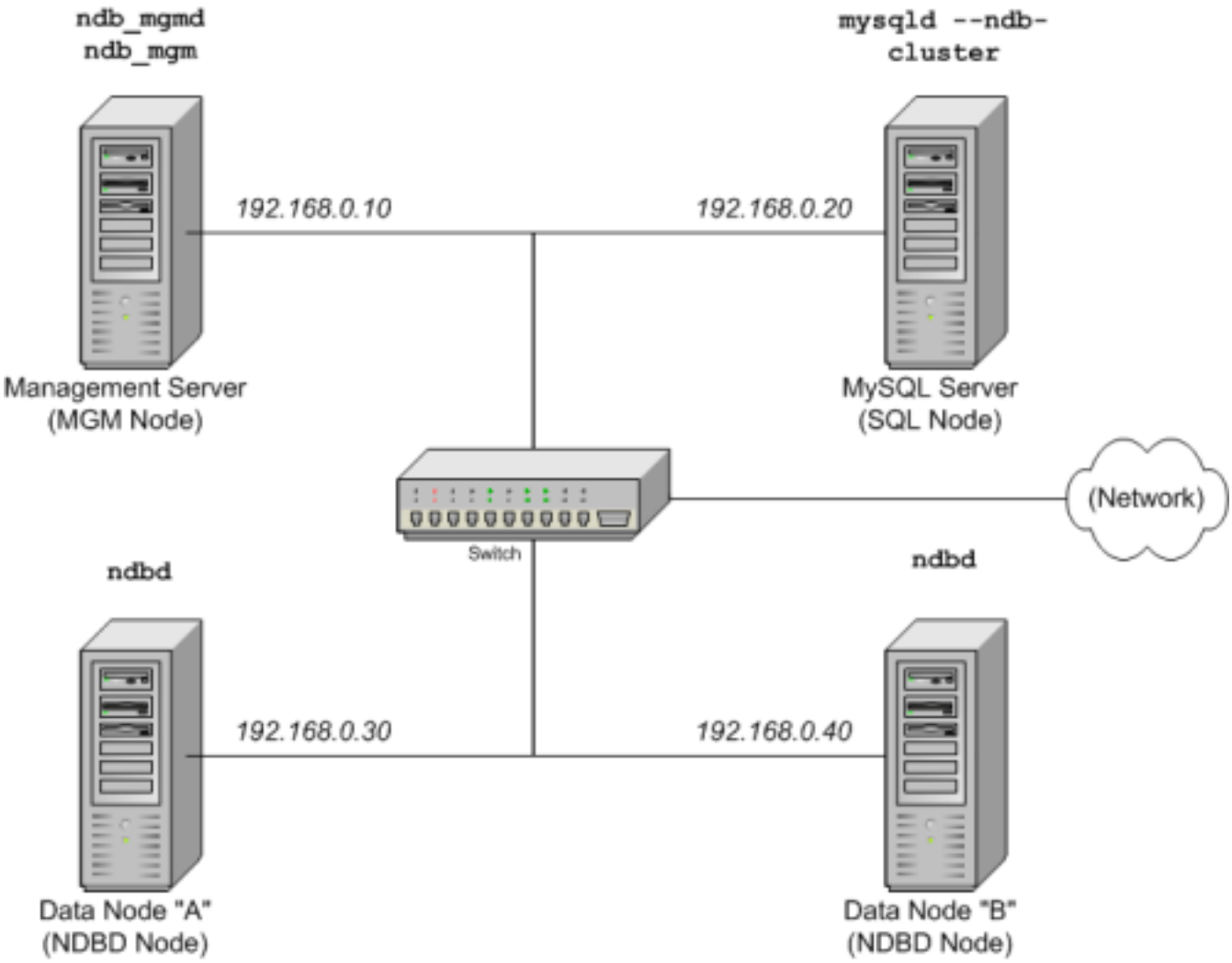
Suposiciones básicas

Este How-To hace las siguientes suposiciones:

1. Estamos preparando un cluster con 4 nodos, cada uno en máquinas separadas, y cada uno con dirección de red fija en una Ethernet como se muestra:

Nodo	Dirección IP
Nodo de administración (MGM)	192.168.0.10
Nodo MySQL server (SQL)	192.168.0.20
Nodo de datos (NDBD) "A"	192.168.0.30
Nodo de datos (NDBD) "B"	192.168.0.40

   Puede verse mejor en el siguiente diagrama:



Nota: Para mayor simplicidad, usamos sólo direcciones IP numéricas en este How-To. Sin embargo, si la resolución de DNS está disponible en su red, es posible usar nombres de equipos en lugar de direcciones IP al configurar el cluster. Alternativamente, puede usar el fichero `/etc/hosts` o el equivalente en su sistema operativo para proporcionar significado al hacer la búsqueda de equipos si está disponible.

2. Cada equipo en nuestro escenario es un PC de sobremesa basado en Intel con una distribución Linux genérica instalada en disco con una configuración estándar, y ejecutando sólo los servicios necesarios. El sistema operativo con un cliente de red TCP/IP estándar es suficiente. Para simplicidad, asumimos que el sistema de ficheros en todas las máquinas está configurado igual. En caso que no fuera así, necesita adaptar estas instrucciones.

3. Tarjetas 100 Mbps o 1 gigabit Ethernet están instaladas en cada máquina, junto con sus drivers, y las cuatro máquinas están conectadas via un elemento de red Ethernet como un switch. (Todas las máquinas deben usar tarjetas de red con la misma velocidad; esto es, las cuatro máquinas del cluster deben tener tarjetas 100 Mbps *o* las 4 máquinas deben tener tarjetas 1 Gbps.) MySQL Cluster funcionará en una red 100 Mbps ; sin embargo, gigabit Ethernet proporciona mejor rendimiento.

Tenga en cuenta que MySQL Cluster **no** está diseñado para una conectividad de red menor a 100 Mbps. Por esta razón (entre otras), intentar ejecutar un MySQL Cluster en una red pública como Internet no funcionará y no está recomendado.

4. Para nuestros datos de prueba, usaremos la base de datos `world` disponible para descarga en la web de MySQL AB. Como ocupa poco espacio, suponemos que cada máquina tiene 256 MB RAM, que debe ser suficiente para ejecutar el sistema operativo, proceso NDB del equipo y (para los nodos de datos) almacenar la base de datos.

Aunque nos referimos a Linux en este How-To, las instrucciones y procedimientos que proporcionamos aquí pueden adaptarse tanto a Solaris como a Mac OS X. Suponemos que sabe realizar una instalación mínima y configurar el sistema operativo con capacidad de red, o que puede obtener asistencia cuando lo necesite.

Discutimos los requerimientos hardware ,software, y de red de MySQL Cluster con más detalle en la siguiente sección . (Consulte Sección 16.3.1, “Hardware, software y redes”.)

16.3.1. Hardware, software y redes

Una de las ventajas de MySQL Cluster es que puede ejecutarse en hardware normal sin ningún requerimiento especial a parte de grandes cantidades de RAM, debido al hecho que todos los datos se almacenan en memoria. (Tenga en cuenta que esto puede cambiar y que queremos implementar almacenamiento en disco en versiones futuras.) Naturalmente, CPUs múltiples y más rápidas mejoran el rendimiento. Los requerimientos de memoria para procesos cluster son relativamente pequeños.

Los requerimientos de software para Cluster son modestos. Los sistemas operativos de las máquinas no requieren ningún modulo no usual, servicios, aplicaciones o configuración extraña para soportar MySQL Cluster. Para Mac OS X o Solaris, la instalación estándar es suficiente. Para Linux, una instalación estándar debe ser todo lo necesario. Los requerimientos del software MySQL son simples: todo lo necesario es una versión de producción de MySQL-max 5.0; debe usar la versión `-max` de MySQL 5.0 para tener soporte de cluster. No es necesario compilar MySQL para usar cluster. En este How-To, asumimos que está usando el `-max` binario apropiado para Linux. Solaris, o Mac OS X disponible en la página de descargas de MySQL http://dev.mysql.com/downloads.

Para comunicación entre nodos, el cluster soporta red TCP/IP en cualquier topología estándar, y como mínimo se espera una red 100 Mbps Ethernet , más un switch, hub, o router para proporcionar conectividad de red al cluster entero. Recomendamos que MySQL Cluster se ejecute en su subred que no está compartida con máquinas no-cluster por las siguientes razones:

- **Seguridad**: La comunicación entre nodos del cluster no están cifradas. La única forma de proteger transmisiones dentro de un MySQL Cluster es ejecutar su cluster en una red protegida. Si trata de usar MySQL Cluster para aplicaciones Web , el cluster debe residir detrás de un firewall y no en su DMZ (DMZ) o en otro lugar.

- **Eficiencia**: Inicializar un MySQL Cluster en una red privada o protegida permite que el cluster haga uso exclusivo del ancho de banda entre máquinas del cluster. Usar un switch esparado para su MySQL Cluster no sólo ayuda a protegerse de accesos no autorizados a los datos del clsuter, también asegura que los nodos del cluster están protegidos de interferencias causadas por transmisiones entre otras máquinas en la red. Para mayor confianza puede usar switches duales y tarjetas duales para eliminar la red como punto único de fallo; varios dispositivos soportan fallos para estos enlaces de comunicación.

Es posible usar la Scalable Coherent Interface (SCI) con MySQL Cluster, pero no es un requerimiento. Consulte Sección 16.7, “Usar interconexiones de alta velocidad con MySQL Cluster” para más información.

16.3.2. Instalación

Cada máquina MySQL Cluster ejecutando nodos de almacenamiento o SQL deben tener insalados el binario MySQL-max . Para nodos de almacenamiento, no es necesario tener el binario MySQL server instalado, pero tiene que instalar el demonio del servidor MGM y los binarios de clientes (`ndb_mgmd` y `ndb_mgm`, respectivamente). En esta sección, cubrimos los pasos necesarios para instalar los binarios correctos para cada tipo de nodo del cluster.

MySQL AB proporciona binarios precompilados que soportan cluster y que no tiene que compilar. (Si necesita un binario personalizado, consulte Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”.) Por lo tanto, el primer paso del proceso de instalación para cada máquina del cluster es bajar la versión más reciente de su plataforma desde MySQL downloads area. Asumimos que los guarda en el directorio `/var/tmp` de cada máquina.

Hay RPMs disponibles para plataformas Linux 32-bit y 64-bit; los binarios `-max` instalados por los RPMs soportan el motor `NDBCluster` . Si elige usarlos en lugar de los binarios, tenga en cuenta que debe instalar **ambos** el paquete `-server` y el `-max` en todas las máquinas que tendrán nodos del cluster. (Consulte Sección 2.4, “Instalar MySQL en Linux” para más información acerca de instalar MySQL usando RPMs.) Tras instalar de los RPM, todavía tiene que configurar el cluster como se discute en Sección 16.3.3, “Configuración”.

Nota: Tras completar la instalación no arranque los binarios. Le mostraremos como hacerlo siguiendo la configuración de todos los nodos.

Instalación de nodos de almacenamiento y SQL

En cada una de las 3 máquinas designadas para tener los nodos de almacenamiento o SQL, realice los siguientes pasos como root del sistema:

1. Compruebe los ficheros `/etc/passwd` y `/etc/group` (o use cualquier herramienta proporcionada por el sistema operativo para tratar usuarios y grupos) para ver si hay un grupo `mysql` y usuario `mysql` en el sistema, como algunas de las distribuciones de sistema operativo los crean como parte del proceso de instalación. Si no están presentes, cree un nuevo grupo `mysql` y un usuario `mysql` para este grupo:

   ```
   groupadd mysql
   useradd -g mysql mysql
   ```

2. Cambie al directorio que contiene el fichero descargado; desempaquételo; cree un symlink al ejecutable mysql-max. Tenga en cuenta que el nombre del fichero y difectorio cambia en función del número de versión de MySQL .

   ```
   cd /var/tmp
   tar -xzvf -C /usr/local/bin mysql-max-5.0.10-pc-linux-gnu-i686.tar.gz
   ln -s /usr/local/bin/mysql-max-5.0.10-pc-linux-gnu-i686 mysql
   ```

3. Cambie al directorio `mysql` , y ejecute el script proporcionado para crear las bases de datos del sistema:

   ```
   cd mysql
   scripts/mysql_install_db --user=mysql
   ```

4. Configure los permisos necesarios para los directorios de MySQL server y de datos:

   ```
   chown -R root .
   ```

```
chown -R mysql data
chgrp -R mysql .
```

Tenga en cuenta que el directorio de datos en cada máquina con un nodo de datos es `/usr/local/mysql/data`. Haremos uso de esta información al configurar el nodo de administración . (Consulte Sección 16.3.3, “Configuración”.)

5. Copie el script de arranque de MySQL en el directorio apropiado, hágalo ejecutable, y configúrelo para arrancar junto al sistema operativo:

```
cp support-files/mysql.server /etc/rc.d/init.d/
chmod +x /etc/rc.d/init.d/mysql.server
chkconfig --add mysql.server
```

Aquí usamos Red Hat `chkconfig` para crear enlaces a los scripts de arranque; use lo que sea apropiado para su sistema operativo / distribución, tal como `update-rc.d` en Debian.

Recuerde que los pasos listados anteriormente deben realizarse separadamente en cada máquina en que el nodo de almacenamiento o SQL residen.

Instalación del nodo de administración

Para el nodo MGM (administración), no es necesario instalar el ejecutable `mysqld`, sólo los binarios para el cliente y servidor MGM, que pueden encontrarse en el archivo `-max` descargado. De nuevo asumimos que ha guardado este fichero en `/var/tmp`. Como root del sistema (esto es, tras usar `sudo`, `su root`, o su equivalente en su sistema para asumir temporalmente los privilegios de administrador de sistema), realice los siguientes pasos para instalar `ndb_mgmd` y `ndb_mgm` en el nodo de administración del cluster:

1. Mueva el directorio `/var/tmp` , y extraiga `ndb_mgm` y `ndb_mgmd` del archivo a un directorio disponible como `/usr/local/bin`:

```
cd /var/tmp
tar -zxvf mysql-max-5.0.10-pc-linux-gnu-i686.tar.gz /usr/local/bin '*/bin/ndb_mgm*'
```

2. Vaya al directorio en que ha desempaquetado los ficheros, y hágalos ejecutables:

```
cd /usr/local/bin
chmod +x ndb_mgm*
```

En Sección 16.3.3, “Configuración”, crearemos y escribiremos los ficheros de configuración para todos los nodos del cluster de ejemplo.

16.3.3. Configuración

Para nuestro cluster de 4 nodos y 4 equipos, necesitamos escribir 4 ficheros de configuración, 1 por nodo/equipo.

- Cada nodo de datos o SQL necesita un fichero `my.cnf` que proporciona dos informaciones: un **connectstring** diciendo al nodo dónde encontrar el nodo MGM , y una línea diciendo al servidor MySQL en este equipo (la máquina del nodo de datos) que se ejecute en modo NDB.

  Para más información de connectstrings, consulte Sección 16.4.4.2, “El `connectstring` de MySQL Cluster”.

- El nodo de administración necesita un fichero `config.ini` que le diga cuántas replicas mantener, cuánta memoria reservar para datos e índices en cada nodo de datos, dónde encontrar los nodos de datos, dónde se guardarán los datos en cada nodo de datos, y dónde encontrar los nodos SQL.

Configuración de los nodos de almacenamiento y SQL

El fichero `my.cnf` necesitado por los nodos de datos es muy simple. El fichero de configuración debe estar localizado en el directorio `/etc` y puede editarse (y crearse en caso necesario) usando un editor de texto, por ejemplo:

```
vi /etc/my.cnf
```

Para cada nodo de datos y SQL en nuestra configuración de ejemplo, `my.cnf` debe tener este aspecto:

```
[MYSQLD]                      # Options for mysqld process:
ndbcluster                    # run NDB engine
ndb-connectstring=192.168.0.10  # location of MGM node

[MYSQL_CLUSTER]               # Options for ndbd process:
ndb-connectstring=192.168.0.10  # location of MGM node
```

Tras introducir lo anterior, guarde este fichero y salga del editor de texto. Hágalo pra las máquinas que guarden el nodo de datos "A", el "B" y el nodo SQL.

Configuración del nodo de administración

El primer paso al configurar el nodo MGM es crear el directorio en que puede encontrarse el fichero de configuración y crear el fichero própiamente dicho. Por ejemplo (ejecutando como root):

```
mkdir /var/lib/mysql-cluster
cd /var/lib/mysql-cluster
vi config.ini
```

Mostramos `vi` para crear el fichero, pero puede usar cualquier editor de textos.

Para nuestra inicialización representativa, el fichero `config.ini` debe leerse así:

```
[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 cluster
                  # Note: It is recommended beginning with MySQL 5.0 that
                  # you do not specify the portnumber 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 logfiles

[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 datafiles

[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 datafiles
```

```
[MYSQLD]                        # SQL node options:
hostname=192.168.0.20           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for SQL node's datafiles
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

(**NOTA**: La base de datos "world" puede descargarse desde http://dev.mysql.com/doc/ donde puede encontrarse en "Examples".)

Una vez que todos los ficheros de configuración se han creado y se han especificado estas opciones, está preparado para arrancar el cluster y verificar que todos los procesos están en ejecución. Se discute acreca de esto en Sección 16.3.4, “Arranque inicial”.

Para información más detallada acerca de los parámetros de configuración de MySQL Cluster y sus usos, consulte Sección 16.4.4, “Fichero de configuración” y Sección 16.4, “Configuración de MySQL Cluster”. Para configuración de MySQL Cluster para realizar copias de seguridad, consulte Sección 16.6.4.4, “Configuración para copias de seguridad de un nodo”.

Nota: El puerto por defecto para administración del cluster es 1186; el puerto por defecto para nodos de datos es 2202. A partir de MySQL 5.0.3, esta restricción se elimina, y el cluster reserva los puertos de los nodos de datos automáticamente de los que están libres.

16.3.4. Arranque inicial

Arrancar el cluster no es complicado una vez configurado. Cada proceso de los nodos del cluster debe arrancarse separadamente, y en la máquina en que reside. Mientras es posible arrancar los nodos en cualquier orden, seguidos por los nodos de almacenamiento y finalmente por los nodos SQL:

1. En la máquina de administración, realice el siguiente comando del shell del sistema para arrancar los procesos del nodo MGM:

   ```
   shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
   ```

   Tenga en cuenta que `ndb_mgmd` debe saber dónde encontrar su fichero de configuración , usando la opción `-f` o `--config-file` . (Consulte Sección 16.5.3, “El proceso del servidor de administración `ndb_mgmd`” para más detalles.)

2. En cada equipo de los nodos de datos, ejecute este comando para arrancar el proceso NDBD por primera vez:

   ```
   shell> ndbd --initial
   ```

   Tenga en cuenta que es muy imporante usar el parámetro `--initial` **sólo** al arrancar `ndbd` por primera vez, o tras reiniciar tras una copia de seguridad/restauración o cambio de configuración. Esto es debido a que este parámetro hará que el nodo borre cualquier fichero creado por instancias `ndbd` anteriormente necesarios para la restauración, incluyendo el fichero log de restauración.

3. En la máquina cluster donde reside el nodo SQL, ejecute un `mysqld` normal como se muestra:

   ```
   shell> mysqld &
   ```

Si todo ha ido bien, y el cluster se ha inicializado correctamente, el cluster debería ser operacional. Puede comprobarlo invocando el nodo cliente de administración `ndb_mgm` ; la salida debe parecerse a la que hay a continuación:

```
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: 5.0.11, Nodegroup: 0, Master)
id=3    @192.168.0.40  (Version: 5.0.11, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @192.168.0.10  (Version: 5.0.11)

[mysqld(SQL)]   1 node(s)
id=4   (Version: 5.0.11)
```

Puede encontrar ligeras diferencias en función de la versión exacta de MySQL que use.

Nota: Si usa una versión antigua de MySQL, puede ver el nodo SQL referenciado como '`[mysqld(API)]`'. Esto refleja un uso antiguo que ya está obsoleto.

Debería ser capaz de trabajar con bases de datos, tablas y datos en MySQL Cluster. Consulte Sección 16.3.5, “Cargar datos de ejemplo y realizar consultas” para una breve discusión.

16.3.5. Cargar datos de ejemplo y realizar consultas

Trabajar don datos en MySQL Cluster no es muy distinto que trabajar con MySQL sin Cluster. Hay que tener en cuenta dos puntos al hacerlo:

- Las tablas deben crearse con la opción `ENGINE=NDB` o `ENGINE=NDBCLUSTER` , o cambiarse (mediante `ALTER TABLE`) para usar el motor NDB CLuster para que puedan replicarse en el cluster. Si está importando tablas de una base de datos existente usando la salida de `mysqldump`, puede abrir el script SQL en un editor de texto y añadir está opción a cualquier comando de creación de tablas, o reemplazar cualquier opción `ENGINE` (o `TYPE`) existente con alguna de estas. Por ejemplo, suponga que tiene la base de datos de ejemplo `world` en otro MySQL server (que no soporta MySQL Cluster), y desea exportar la definición de la tabla `City` :

  ```
  shell> mysqldump --add-drop-table world City > city_table.sql
  ```

  El fichero `city_table.sql` resultante contendrá el comando de creación de la tabla (y el comando `INSERT` necesario para importar los datos de la tabla):

  ```
  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)
  ) TYPE=MyISAM;

  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)
  ```

Necesitará asegurarse que MySQL usa el motor NDB para esta tabla. Hay dos formas de hacerlo. Una es, **antes** de importar la tabla en la base de datos del cluster, modificar su definición para que lea (usando `City` como ejemplo):

```
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;

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);
# (etc.)
```

Esto debe hacerse para la definición de cada tabla que será parte de la base de datos clusterizada. La forma más fácil de hacerlo es símplemente hacer un buscar y reemplazar en el fichero `world.sql` y reemplazar todas las instancias de `TYPE=MyISAM` con `ENGINE=NDBCLUSTER`. Si no quiere modificar el fichero, puede usar `ALTER TABLE`; consulte a continuación las particularidades.

Asumiendo que ha creado la base de datos llamada `world` en el nodo SQL del cluster, puede usar el cliente de línea de comandos `mysql` para leer `city_table.sql`, y crear y llenar de datos la tabla correspondiente de la forma usual:

```
shell> mysql world < city_table.sql
```

Es muy importante recordar que el comando anterior debe ejecutarse en la máquina en que corre el nodo SQL -- en este caso, la máquina con dirección IP **192.168.0.20**.

Para crear una copia de la base de datos `world` en el nodo SQL, guarde el fichero en `/usr/local/mysql/data`, luego ejecute

```
shell> cd /usr/local/mysql/data
shell> mysql world < world.sql
```

Por supuesto, el script SQL debe ser leíble por el usuario `mysql` . Si guarda el fichero en un lugar distinto, ajuste lo anterior correctamente.

Es importante tener en cuenta que NDB Cluster en MySQL 5.0 no soporta descubrimiento automático de bases de datos. (Consulte Sección 16.8, “Limitaciones conocidas de MySQL Cluster”.) Esto significa que , una vez que la base de datos `world` y sus tablas se han creado en un nodo de datos, necesitará ejecutar el comando `CREATE DATABASE world;` (a partir de MySQL 5.0.2, puede usar `CREATE SCHEMA world;` en su lugar), seguido por `FLUSH TABLES;` en cada nodo de datos del cluster. Esto hará que el nodo reconozca la base de datos y lea sus definiciones de tablas.

Ejecutar consultas `SELECT` en el nodo SQL no es distinto a ejecutarlas en cualquier otra instancia de un MySQL server. Para ejecutar consultas de línea de comandos, primero necesita registrarse en el MySQL Monitor normalmente:

```
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: 4.1.9-max

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

mysql>
```

Si no modifica las cláusulas `ENGINE=` en las definiciones de tabla antes de importar el script SQL, debe ejecutar los siguientes comandos en este punto:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Tenga en cuenta que símplemente usamos la cuenta `root` por defecto del sistema con una contraseña vacía. Por supuesto, en un sistema de producción, debe **siempre** seguir las precauciones de seguridad para instalar un MySQL server, incluyendo una contraseña de root y la creación de una cuenta de usuario con sólo los permisos necesarios para realizar las tareas necesarias para ese usuario. Para más información acerca de esto, consulte Sección 5.6, “El sistema de privilegios de acceso de MySQL”.

Vale la pena tener en cuenta que los nodos del cluster no utilizan el sistema de permisos de MySQL al acceder el uno al otro, y preparar o cambiar las cuentas de usuario de MySQL (incluyendo la cuenta `root`) no tiene efecto en la interacción entre nodos, sólo en aplicaciones accediendo al nodo SQL.

Seleccionar una base de datos y ejecutar una consulta `SELECT` contra una tabla en la base de datos se realiza de la forma normal, como salir del 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>
```

Las aplicaciones usando MySQL pueden usar APIs estándar. Es importante recordar que sus aplicaciones deben acceder al nodo SQL, y no al nodo MGM o a los de almacenamiento. Este breve ejemplo muestra cómo puede ejecutar la misma consulta que la anterior usando la extensión de PHP 5 `mysqli` ejecutando un servidor web en cualquier punto de la red:

```
<!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', '', '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>
```

Suponemos que el proceso ejecutándose en el servidor web puede alcanzar la IP del nodo SQL.

De forma parecida, puede usar la MySQL C API, Perl-DBI, Python-mysql, o los conectores propios de MySQL AB para realizar las tareas de definición y manipulación de datos como haría normalmente con MySQL.

- Recuerde que *cada tabla NDB debe tener una clave primaria*. Si no se define clave primaria por el usuario cuando se crea la tabla, el motor `NDB Cluster` creará una oculta automáticamente. (**Nota**: esta clave oculta ocupa espacio como cualquier otro índice de tabla. No es raro encontrar problemas debido a espacio insuficiente en memoria para guardar estos índices creados automáticamente.)

16.3.6. Apagado y encendido seguros

Para parar el cluster símplemente introduzca lo siguiente en una shell en la máquina con el nodo MGM :

```
shell> ndb_mgm -e shutdown
```

Esto hará que ndb_mgm, ndb_mgmd, y cualquier proceso ndbd termine normalmente. Cualquier nodo SQL puede terminarse usando mysqladmin shutdown y otros medios. Tenga en cuenta que la opción -e aquí se usa para pasar un comando al cliente ndb_mgm desde el shell. Consulte Sección 4.3.1, “Usar opciones en la línea de comandos”.

Para reiniciar el cluster, símplemente ejecute estos comandos:

- En el equipo de administración (192.168.0.10 en nuestra configuración):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- En cada uno de los nodos de datos (192.168.0.30 y 192.168.0.40):

```
shell> ndbd
```

  Recuerde de **no** invocar este comando con la opción --initial cuando reinicie el nodo NDBD normalmente.

- Y en el equipo SQL (192.168.0.20):

```
shell> mysqld &
```

Para inforamción de hacer copias de seguridad de clusters, consulte Sección 16.6.4.2, “Usar el servidor de administración para crear una copia de seguridad”.

Para restaurar un cluster de una copia de seguridad requiere el uso del comando ndb_restore . Esto se trata en Sección 16.6.4.3, “Cómo restablecer una copia de seguridad de un nodo”.

Más información sobre la configuración del cluster MySQL puede encontrarse en Sección 16.4, “Configuración de MySQL Cluster”.

16.4. Configuración de MySQL Cluster

Un MySQL server que es parte de un MySQL Cluster difiere sólo en un aspecto de un servidor MySQL normal (no cluster), en que emplea el motor NDB Cluster . Este motor también se conoce símplemente como NDB, y las dos formas del nombre son sinónimas.

Para evitar reserva innecesaria de recursos, el servidor se configura por defecto con el motor NDB desactivado. Para activar NDB, necesitará configurar el fichero de configuración my.cnf del servidor con la opción --ndbcluster .

Desde que MySQL server es parte del cluster, necesita datos de configuración que sepa cómo acceder al nodo MGM para obtener datos de configuración del cluster. El comportamiento por defecto es buscar el nodo MGM en localhost. Sin embargo, puede necesitar especificar su localización donde se encuentre, esto puede hacerse en my.cnf o en la línea de comandos del servidor MySQL. Antes de poderse usar el NDB, al menos un nodo MGM debe ser operacional, así como los nodos de datos deseados.

16.4.1. Generar MySQL Cluster desde el código fuente

NDB, el motor del cluster, está disponible en distribución binaria para Linux, Mac OS X, y Solaris. No está disponible en Windows, pero queremos que lo esté para win32 y otras plataformas en un futuro próximo.

Si elige compilar desde un paquete fuente o desde el MySQL 5.0 BitKeeper tree, asegúrese de usar la opción --with-ndbcluster cuando ejecute configure. Puede usar el script BUILD/compile-

`pentium-max` . Tenga en cuenta que este script incluye OpenSSL,así que debe tener u obtener OpenSSL para compilar con éxito; de otro modo, necesitará modificar `compile-pentium-max` para excluir este requerimiento. Por supuesto, puede seguir las instrucciones estándar para compilar sus propios binarios, luego realizar los tests usuales y procedimiento de instalación. Consulte Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”.

16.4.2. Instalar el software

En las siguientes secciones, asumimos que conoce el proceso de instalación de MySQL,y cubrimos sólo las diferencias entre configurar MySQL Cluster y configurar MySQL sin clustering. (Consulte Capítulo 2, *Instalar MySQL* si necesita más información sobre lo último.)

Encontrará la configuración del cluster más sencilla si ya tiene todos los nodos de administración y datos en ejecución; esta es la parte más costosa de tiempo de la configuración. Editar el fichero `my.cnf` es sencillo y esta sección cubre sólo las diferencias de configurar MySQL sin clustering.

16.4.3. Rápido montaje de prueba de MySQL Cluster

Para familiarizarse con los conceptos básicos, describimos la configuración más sencilla para un MySQL Cluster. Después, debería poder diseñar su inicialización deseada a partir de la información proporcionada en las otras secciones de este capítulo.

Primero, necesita crear un directorio de configuración tal como `/var/lib/mysql-cluster`, ejecutando el siguiente comando como root del sistema:

```
shell> mkdir /var/lib/mysql-cluster
```

En este directorio, cree un fichero llamado `config.ini` con la siguiente información, substituyendo los valores apropiados por `HostName` y `DataDir` como sea necesario para su sistema.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't 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 and API are 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]
```

Ahora puede arrancar el servidor de administración:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Arranque un nodo DB sencillo usando `ndbd`. Al arrancar `ndbd` para un nodod DB dado por primera vez, debe usar la opción `--initial` :

```
shell> ndbd --initial
```

Para arrancar `ndbd` de nuevo,normalmente **no** necesitará usar esta opción:

```
shell> ndbd
```

Esto es porque la opción `--initial` borra todos los ficheros de datos y log existentes (así como todos los metadatos de tablas) para este nodo de datos y crea nuevos. Una excepción a esta regla es al reiniciar el cluster y restaurar desde una copia de seguridad tras añadir nuevos nodos de datos.

Por defecto, `ndbd` buscará el servidor de administración en `localhost` en el puerto 1186.

Nota: Si ha instalado MySQL desde un tarball binario, necesitará especificar la ruta de los servidores `ndb_mgmd` y `ndbd` explícitamente. (Normalemnte, se encuentran en `/usr/local/mysql/bin`.)

Finalmente, vaya al directorio de datos de MySQL (usualmente `/var/lib/mysql` o `/usr/local/mysql/data`), y asegúrese que el fichero `my.cnf` contiene la opción necesaria para permitir el motor NDB :

```
[mysqld]
ndbcluster
```

Ahora puede arrancar el servidor MySQL normalmente:

```
shell> mysqld_safe --user=mysql &
```

Espere un momento para asegurarse que el servidor MySQL está corriendo apropiadamente. Si ve `mysql ended`, chequee el fichero `.err` del servidor para averiguar qué ha fallado.

Si todo ha ido bien, puede arrancar usando el cluster:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.10-Max

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

mysql> SHOW ENGINES;
+------------+---------+----------------------------------------------------------------+
| Engine     | Support | Comment                                                        |
+------------+---------+----------------------------------------------------------------+
...
| NDBCLUSTER | DEFAULT | Clustered, fault-tolerant, memory-based tables                 |
| NDB        | YES     | Alias for NDBCLUSTER                                           |
...

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)
```

Para chequear que sus nodos se han inicializado correctamente, arranque el cliente de administración como se muestra:

```
shell> ndb_mgm
```

Puede usar el comando `SHOW` desde el cliente de administración para obtener un reporte del estado del cluster:

```
NDB> 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)
```

En este punto, ha inicializado correctamente un cluster MySQL. Ya puede almacenar datos en el cluster usando cualquier tabla creada con `ENGINE=NDBCLUSTER` o su alias `ENGINE=NDB`.

16.4.4. Fichero de configuración

Configurar MySQL Cluster requiere trabajar con dos ficheros:

- `my.cnf`: Especifica opciones para todos los ejecutables del MySQL Cluster . Este fichero, con el que debe familiarizarse de trabajar anteriormente con MySQL, debe ser accesible por cada ejecutable del cluster.
- `config.ini`: Este fichero es de sólo lectura por el servidor de administración del MySQL Cluster , que distribuye la información contenida en el mismo a todos los procesos participantes en el cluster. `config.ini` contiene una descripción de cada nodo del cluster. Esto incluye los parámetros de configuración para nodos de datos y parámetros de configuración para conexiones entre todos los nodos del cluster.

Mejoramos la configuración del cluster contínuamente y tratamos de simplicar el proceso. Mientras tratamos de mantener compatibilidad con versiones anteriores, puede ser que a veces hagamos cambios incompatibles. Si encuentra uno de estos cambios no documentados, use nuestra Bugs Database para reportarlo.

16.4.4.1. Ejemplo de configuración para MySQL Cluster

Para soportar MySQL Cluster, necesita actualizar `my.cnf` como se muestra en el siguiente ejemplo. Tenga en cuenta que las opciones mostradas aquí no deben confundirse con las de los ficheros `config.ini` . Puede especificar estos parámetros al invocar los ejecutables desde línea de comandos

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid in MySQL 5.0)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com


# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring 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
```

(Para más información acerca de los connectstrings, consulte Sección 16.4.4.2, “El `connectstring` de MySQL Cluster”.)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Puede usar una sección separada `[mysql_cluster]` en el cluster `my.cnf` para configuración que deba ser leída y afecte a todos los ejecutables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Actualmente el fichero de configuración está en formato INI, y se llama `config.ini` por defecto . Lo lee `ndb_mgmd` al arrancar y puede situarse en cualquier sitio. Su localización y nombre se especifican usando `--config-file=[<path>]<filename>` en la línea de comandos con `ndb_mgmd`. Si el fichero de configuración no se especifica, `ndb_mgmd` trata por defecto de leer el fichero `config.ini` localizado en el directorio de trabajo actual.

Los valores por defecto se definen para la mayoría de parámetros, y pueden especificarse en `config.ini`. Para crear una sección de valores por defecto, añada la palabra `DEFAULT` al nombre de sección. Por ejemplo, los nodos de datos se configuran usando las secciones `[NDBD]`. Si todos los nodos de datos usan el mismo tamaño de memoria de datos, y este no es le mismo que el tamaño por defecto, cree una sección `[NDBD DEFAULT]` que contenga una línea con `DataMemory` para especificar el tamaño por defecto de la memoria de datos para todos los nodos de datos.

El formato INI consiste en secciones precedidas por cabeceras de secciones (rodeados por corchetes), segidos por los nombres y valores apropiados de parámetros. Una desviación del formato estándar es que el nombre y valor del parámetro puede separarse por un punto y coma (':') así como el signo de igualdad

('='); otra es que las secciones no se identifican únicamente por el nombre. En su lugar, las entradas únicas (tales como dos nodos distintos del mismo tipo) se identifican por un ID único.

Como mínimo, el fichero de configuración debe definir las máquinas y nodos involucrados en el cluster y en qué máquinas están estos nodos. Como ejemplo de un fichero de configuración simple para un cluster con un servidor de administración, dos nodos de datos y dos servidores MySQL se muestra a continuación:

```
# 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 MySQL Server 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
```

Hay seis secciones distintas en este fichero de configuración:

- `[COMPUTER]`: Define las máquinas del cluster.
- `[NDBD]`: Define los nodos de datos del cluster.
- `[MYSQLD]`: Define los nodos MySQL del cluster.
- `[MGM]` o `[NDB_MGMD]`: Define el nodo de administración del cluster.
- `[TCP]`: Define conexiones TCP/IP entre nodos en el cluster, siendo TCP/IP el protocolo de conexión por defecto.
- `[SHM]`: Define conexiones de memoria compartida entre nodos. Antiguamente, este tipo de conexión estaba disponible sólo en binarios compilados con la opción `--with-ndb-shm` . En MySQL 5.0-Max, está activado por defecto, pero debe considerarse experimental.

Tenga en cuenta que cada nodo tiene su propia sección en `config.ini`. Por ejemplo, desde que el cluster tiene dos nodos de datos, el fichero de configuración contiene dos secciones definiendo estos nodos.

Puede definir valores `DEFAULT` para cada sección. En MySQL 5.0, todos los nombres de parámetros no son sensibles a mayúsculas.

16.4.4.2. El `connectstring` de MySQL Cluster

Con excepción del servidor de administración MySQL Cluster (`ndb_mgmd`), cada nodo de un MySQL Cluster requiere de un *connectstring* que apunte a la localización del servidor de administración. Se usa

para establecer una conexión al servidor de administración así como realizar otras tareas en función del rol del nodo en el cluster. La sintaxis para el connectstring es la siguiente:

```
<connectstring> :=
[<nodeid-specification>,]<host-specification>[,<host-specification>]
<nodeid-specification> := nodeid=<id>
<host-specification> := <host>[:<port>]
```

`<id>` es un entero mayor que 1 que identifica un nodo en config.ini `<port>` es un entero que se refiere a un puerto unix regular `<host>` es una cadena que tiene una dirección de equipo de internet válida

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):   "myhost1"
```

Todos los nodos usan `localhost:1186` como valor connectstring por defecto si no se especifica otro. Si se omite `<port>` del connectstring, el puerto por defecto es 1186. Este puerto debe estar siempre disponible en la red, ya que se ha asignado por la IANA para este propósito (consulte http://www.iana.org/assignments/port-numbers para más detalles).

Listando múltiples valores `<host-specification>` , es posible diseñar varios servidores de administración redundantes. Un nodo del cluster tratará de contactar con administradores sucesivamente en cada equipo en el orden especificado, hasta que se establezca una conexión.

Hay un número de distintos modos de especificar el connectstring:

- Cada ejecutable tiene su propia opción de línea de comandos que permite especificar el servidor de administración al arrancar. (Consulte la documentación para los respectivos ejecutables.)
- Es posible en MySQL 5.0 Cluster inicializar el connectstring para todos los nodos en el cluster a la vez poniéndolo en la sección `[mysql_cluster]` en el fichero del servidor de administración `my.cnf`.
- Para compatibilidad con versiones anteriores, hay dos otras opciones disponibles, usando la misma sintaxis:

  1. Inicialice la variable de entorno `NDB_CONNECTSTRING` para que contenga el connectstring.
  2. Escriba el connectstring para cada ejecutable en un fichero de texto llamado `Ndb.cfg` y guarde este fichero en el directorio de arranque del ejecutable.

  Sin embargo, esto está ahora obsoleto y no debería usarse en nuevas instalaciones.

El método recomendado para especificar el connectstring es inicializarlo en la línea de comandos o en el fichero `my.cnf` para cada ejecutable.

16.4.4.3. Definir los equipos que forman un cluster MySQL

La sección `[COMPUTER]` no tiene otro significado a parte de servir como forma de eliminar la necesidad de definir nombres de equipos para cada nodo en el sistema. Todos los parámetros mencionados aquí son necesarios.

- `[COMPUTER]Id`

  Este valor es un entero, usado para referirse la máquina equipo en cualquier punto del fichero de configuración.

- `[COMPUTER]HostName`

  Este es el nombre de equipo o dirección IP de la máquina.

16.4.4.4. Definición del servidor de administración de MySQL Cluster

La sección `[NDB_MGMD]` (o su alias `[MGM]`) se usa para configurar el comportamiento del servidor de administración. Todos los parámetros de la siguiente lista puede omitirse y, si así es, se asumen sus valores por defecto. **Nota**: Si ni el parámetro `ExecuteOnComputer` ni `HostName` están presentes, se asume para ambos el valor por defecto `localhost` .

- `[NDB_MGMD]Id`

  Cada nodo en el cluster tiene un identificador único, que está representado por un valor entero en el rango de 1 a 63 inclusive. Este ID se usa por todos los mensajes de cluster internos para dirigirse al nodo.

- `[NDB_MGMD]ExecuteOnComputer`

  Esto se refiere a una de las máquinas definidas en la sección `[COMPUTER]` .

- `[NDB_MGMD]PortNumber`

  Este es el número de puerto en que escucha el servidor de administración para peticiones de configuración o comandos de administración.

- `[NDB_MGMD]LogDestination`

  Este parámetro especifica dónde enviar la información de registro del cluster. Hay tres opciones al respecto: `CONSOLE`, `SYSLOG`, y `FILE`:

  - `CONSOLE` envía el log a `stdout`:

    ```
    CONSOLE
    ```

  - `SYSLOG` envía el log a un `syslog` , siendo los valores posibles uno de `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, o `local7`.

    Nota: No todos los dispositivos son soportadas necesariamente por cada sistema operativo.

    ```
    SYSLOG:facility=syslog
    ```

  - `FILE` envía por un pipe la salida del log del cluster a un fichero normal de la misma máquina. Pueden especificarse los siguientes valores:

    - `filename`: Nombre del fichero de log.
    - `maxsize`: Tamaño máximo al que puede crecer el fichero antes de pasar el log a un nuevo fichero. Cuando esto ocurre, el antiguo fichero de log se renombra añadiendo `.x` al nombre del fichero, donde `x` es el siguiente número no usado todavía por este nombre.
    - `maxfiles`: Número máximo de ficheros de log.

    ```
    FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
    ```

    Es posible especificar múltiples destinaciones de logs como se muestra aquí, usando una cadena delimitada por puntos y comas:

    ```
    ```

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

El valor por defecto para el parámetro `FILE` es `FILE:filename=ndb_<id>_cluster.log,maxsize=1000000,maxfiles=6`, donde `<id>` es el ID del nodo.

- `[NDB_MGMD]ArbitrationRank`

  Este parámetro se usa para definir qué nodos pueden actuar como árbitros. Sólo los nodos MGM y SQL pueden serlo. `ArbitrationRank` puede tener uno de los siguientes valores:

  - `0`: El nodo nunca se usará como árbitro.
  - `1`: El nodo tiene alta prioridad, esto es, será preferido como árbitro sobre los nodos con baja prioridad.
  - `2`: Indica que un nodo de baja prioridad será usado como árbitro sólo si un nodo con una prioridad alta no está disponible para este propósito.

  Normalmente, el servidor de administración debe configurarse como árbitro poniendo `ArbitrationRank` a 1 (el valor por defecto) y todos los nodos SQL a 0.

- `[NDB_MGMD]ArbitrationDelay`

  Un valor entero hace que las respuestas a peticiones de árbitro al servidor de administración se retrasen el número de milisegundos indicado. Por defecto, este valor es 0, normalmente no es necesario cambiarlo.

- `[NDB_MGMD]DataDir`

  Cambia el directorio donde se guardan los ficheros de salida del servidor de administración. Estos ficheros incluyen ficheros de log del cluster, ficheros de salida de procesos, y los ficheros pid de los demonios. (Para ficheros de log, esto puede sobreescribirse poniendo el parámetro `FILE` para `[NDB_MGMD]LogDestination` como se discute préviamente en esta sección.)

16.4.4.5. Definir los nodos de MySQL Cluster

La sección `[NDBD]` se usa para configurar el comportamiento de los nodos de datos del cluster. Hay varios parámetros que controlan los tamaños de buffer, de pool, timeouts, y así. Los únicos parámetros obligatorios son:

- O `ExecuteOnComputer` o `HostName`.
- El parámetro `NoOfReplicas`

Se tienen que definir en la sección `[NDBD DEFAULT]` .

La mayoría de parámetors de los nodos de datos se inicializan en la sección `[NDBD DEFAULT]` . Sólo los parámetros marcados como capaces de cambiar valores locales se permiten cambiar en la sección `[NDBD]` . `HostName`, `Id` y `ExecuteOnComputer` *deben* definirse en la sección `[NDBD]` local.

Identificar nodos de datos

El valor `Id` (esto es, el identificador del nodo de datos) puede cambiarse en la línea de comandos cuando se arranca el nodo o en el fichero de configuración.

Para cada parámetro es posible usar `k`, `M`, o `G` como sufijo para indicar unidades de 1024, 1024*1024, o 1024*1024*1024. (Por ejemplo, `100k` significa 100 * 1024 = 102400.) Los parámetros y valores son sensibles a mayúsculas.

- `[NBDB]Id`

  Este es el ID del nodo usado como dirección del nodo para todos los mensajes internos del cluster. Este es un entero entre 1 y 63. Cada nodo en el cluster tiene una identidad única.

- `[NDBD]ExecuteOnComputer`

  Se refiere a uno de las máquinas (equipos) definidos en la sección `COMPUTER` .

- `[NDBD]HostName`

  Especificar este parámetro tiene un efecto similar a especificar `ExecuteOnComputer`. Define el nombre de equipo de la máquina en que reside el nodo de almacenamiento. Este parámetro o `ExecuteOnComputer` es necesario para especificar un nombre de equipo distinto a `localhost`.

- (*OBSOLETO*) `[NDBD]ServerPort`

  Cada nodo en el cluster usa un puerto para conectarse a otros nodos. Este puerto se usa también para transporte distinto a TCP en la fase de establecimiento de la conexión. Ahora, el puerto por defecto se reserva dinámicamente de forma que se asegura que dos nodos en la misma máquina reciban el mismo número de puerto, no debe ser necesario indicar un valor para este parámetro.

- `[NDBD]NoOfReplicas`

  Este parámetro global sólo puede cambiarse en la sección `[NDBD DEFAULT]` , y define el número de replicas para cada tabla almacenada en el cluster. Este parámetro también especifica el tamaño de los grupos de nodos. Un grupo de nodos es un conjunto de nodos que almacenan todos la misma información.

  Los grupos de nodos se forman implítamente. El primer grupo de nodos se forma por el conjunto de nodos de datos con los IDs de nodo más bajos, el siguiente grupo de nodos se conforma con el conjunto del siguiente conjunto de identidades de nodos más baja, y así. Como ejemplo, asuma que tenemos 4 nodos de datos y que `NoOfReplicas` es 2. Los cuatro nodos de datos tienen los IDs 2, 3, 4 y 5. El primer grupo de nodos está formado con los nodos 2 y 3, y el segundo grupo con los nodos 4 y 5. Es importante configurar el cluster de forma que los nodos en el mismo grupo no se guarden en la misma máquina, ya que en esta situación un fallo de hardware haría que fallare el cluster entero.

  Si no hay IDs de nodos el orden de los nodos de datos es el factor determinante para el grupo de nodos. Se haga o no una asignación explícita, puede verse en la salida del comando del cliente administrador `SHOW` .

  No hay valor por defecto para `NoOfReplicas`; el valor máximo posible es 4.

- `[NDBD]DataDir`

  Este parámetro especifica el directorio donde los ficheros de traceo, ficheros de log, ficheros pid y logs de errores se guardan.

- `[NDBD]FileSystemPath`

  Este parámetro especifica el directorio donde se guardan todos los ficheros para metadatos, logs de REDO, logs de UNDO y ficheros de datos. Por defecto es el directorio especificado por `DataDir`. **Nota**: Este directorio debe existir antes de inciar el proceso `ndbd`.

  La jerarquía de directorio recomendada para MySQL Cluster incluye `/var/lib/mysql-cluster`, bajo qué directorio se crea un sistema de ficheros de un nodo. Este subdirectorio contiene el ID del nodo. Por ejemplo, si el ID del nodo es 2, el subdirectorio se llama `ndb_2_fs`.

- `[NDBD]BackupDataDir`

  Es posible especificar el directorio en que se guardan las copias de seguridad. Por defecto, este directorio es *`FileSystemPath`*`/BACKUP`. (Consulte secciones anteriores.)

Memoria de datos e índice

`DataMemory` y `IndexMemory` son parámetros especificando el tamaño de los segmentos de memoria usados para almacenar los registros actuales y sus índices. Al inicializar los valores de los mismos, es importante entender cómo se usan `DataMemory` y `IndexMemory` , ya que usualmente necesitan actualizarse para reflejar el uso real del cluster:

- `[NDBD]DataMemory`

  Este parámetro define la cantidad de espacio disponible para almacenar registros de base de datos. La cantidad entera se guardará en memoria, así que es extremadamente importante que la máquina tenga sufuciente memoria física para guardar este valor.

  La memoria reservada por `DataMemory` se usa para almacenar los registros y los índices. Cada registro tiene una longitud fija . (Incluso columnas `VARCHAR` se almacenan como columnas de longitud fija.) Hay una sobrecarga de 16-byte para cada registro; una cantidad adicional para cada registro se necesita porque se almacena en páginas de 32KB con sobrecarga de 128 byte por página (consulte a continuación). También hay una pequeña cantidad gastada por página debido al hecho que cada registro se almacena en sólo una página. El tamaño de registro máximo actualmente es de 8052 bytes.

  El espacio de memoria definido por `DataMemory` se usa para almacenar índices ordenados, que usa acerca de 10 bytes por registor. Cada registro de tabla se representa en el índice ordenado. Un error común entre usuarios es asumir que todos los índices se almacenan en la memoria reservada por `IndexMemory`, pero no es así: sólo claves primarias e índices hash únicos usan esta memoria; los índices ordenados usan la memoria almacenada por `DataMemory`. Sin embargo, crear una clave primar única o índice hash único también crea un índice ordenado en la misma clave, a no ser que especifique `USING HASH` en el comando de creación del índice. Esto puede verificarse ejecutando `ndb_desc -d` *`db_name table_name`* en el cliente de administración.

  El espacio de memoria reservado por `DataMemory` consiste en 32KB páginas, que se reservan en fragmentos de tabla. Cada tabla se particiona normalmente en el mismo número de fragmentos como hay nodos de datos en el cluster. Por lo tanto, por cada nodo, hay el mismo número de fragmentos así como se especifica en `NoOfReplicas`. Una vez que una página se ha reservado, no es posible retornarla al pool de páginas libres, excepto borrando la tabla. Ejecutar el modo de recuperación también comprime la partición ya que todos los registros se insertan en particiones vacías por parte de los otros nodos vivos.

  El espacio de memoria `DataMemory` también contiene información UNDO: Para cada actualización, se reserva una copia de los registros no alterados en `DataMemory`. También hay una referencia de cada copia en los índices de tablas ordenadas. Los índices hash únicos se modifican sólo cuando las columnas de índice único se actualizan, en tal caso se inserta una nueva entrada en la tabla de índices y la antigua entrada se borra al hacer un commit. Por esta razón, es necesario reservar espacio necesario para tratar la transacción más grande realizada por aplicaciones usando el cluster. En cualquier caso, realizar unas cuantas transacciones muy grandes no tiene ventaja sobre usar muchas pequeñas, por las siguientes razones:

  - Las transacciones grandes no son más rápidas que las pequeñas.
  - Las transacciones grandes incrementan el número de operaciones que se pierden y deben repetirse en caso de fallo de transacción

- Las transacciones grandes usan más memorias

El valor por defecto de `DataMemory` es 80MB; el mínimo es 1MB. No hay tamaño máximo, pero en realidad el tamaño máximo tiene que adaptarse para que el proceso no empiece ha hacer swapping cuando llega al límite. Este límite se determina por la cantidad de RAM física disponible en la máquina y por la cantidad de memoria que el sistema operativo puede asignar a un proceso. Sistemas operativos 32-bit están normalmente limitados a 2-4GB por proceso; los sistemas opertivos de 64-bit pueden usar más. Para bases de datos grandes, puede ser preferible usar un sistema operativo 64-bit por esta razón. Además, es posible ejecutar más de un proceso `ndbd` por máquina, y esto puede ser ventajoso en máquinas con múltiples CPUs.

- `[NDBD]IndexMemory`

Este parámetro controla la cantidad de almacenamiento usado por índices hash en MySQL Cluster. Los índices hash siempre son usados por índices de clave primaria, índices únicos y restricciones únicas. Tenga en cuenta que cuando define una clave primaria y un índice único, se crean dos índices, uno de los cuales es un índice hash usado por todos los accesos a tuplas así como tratamiento de bloqueos. También se usa para forzar restricciones únicas.

El tamaño del índice hash es de 25 bytes por registro, más el tamaño de la clave primaria. Para claves primarias mayores a 32 bytes se añaden otros 8 bytes .

Considere una tabla definida por

```
CREATE TABLE example
(
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

Hay una sobrecarga de 12 bytes (tener columnas no nulas ahorra 4 bytes de sobrecarga) más 12 bytes de datos por registro. Además tenemos dos índices ordenados en las columnas **a** y **b** consumiendo cerca de 10 bytes cada uno por registro. Hay un índice hash de clave primaria en la tabla base con unos 29 bytes por registro. La restricción única se implementa por tablas separadas con **b** como clave primaria y **a** como columna. Esta tabla consumirá otros 29 bytes de memoria de índice por registro en la tabla **ejemplo** así como 12 bytes de sobrecarga, más 8 bytes de datos de registros.

Así, para un millón de registros, necesitamos 58 MB para memoria de índices para tratar los índices hash para la clave primaria y la restricción única. También necesitamos 64 MB para los registros de la tabla base y la tabla de índice único, más las dos tablas con índice ordenado.

Puede ver que los índices hash tienen una buena cantidad de espacio de memoria; sin embargo, proporcionan acceso muy rápido a los datos retornados. También se usan en MySQL Cluster para tratar restricciones únicas.

Actualmente el único algoritmo de partición es hashing y los índices ordenados son locales para cada nodo. Por lo tanto los índices ordenados no pueden usarse para tratar restricciones únicas en caso general.

Un punto importante para`IndexMemory` y `DataMemory` es que el tamaño de base de datos total es la suma de toda la memoria de datos y toda la memoria de índice para cada grupo de nodos. Cada grupo de nodos se usa para guardar información replicada, así que si hay cuatro nodos con 2 réplicas, habrá

dos nodos de grupos. Por lo tanto, la cantidad total de memoria disponible es 2*`DataMemory` para cada nodo de datos.

Se recomienda que `DataMemory` y `IndexMemory` se inicialice con el mismo valor para todos los nodos. Desde que los datos se distribuyen eventualmente sobre todos los nodos en el cluster la cantidad de espacio máxima disponible no es mejor que la del menor nodo del cluster.

`DataMemory` y `IndexMemory` pueden cambiarse, pero decrementar cualquiera de ellos puede ser arriesgado; al hacerlo puede hacer que un nodo o incluso el cluster entero no sea capaz de reiniciar debido a que hay memoria insuficiente. Incrementar estos valores está bien, pero se recomienda que estas actualizaciones se realicen de la misma manera que una actualización de software, comenzando con una actualización del fichero de configuración, y luego reiniciando el servidor de administración seguido del reinicio de cada nodo de datos.

Las actualizaciones no incrementan la cantidad de memoria índice usada. Las inserciones tienen efecto inmediatamente; sin embargo, los registros no se borran realmente hasta que la transacción hace un commit.

El valor por defecto para `IndexMemory` es 18MB. El mínimo es 1MB.

Parámetros de transacción

Los siguientes tres parámetros que discutimos son importanets porque afectan al número de transacciones paralelas y los tamaños de transacciones que pueden tratarse por el sistema. `MaxNoOfConcurrentTransactions` es el número de transacciones posibles en un nodo; `MaxNoOfConcurrentOperations` es el número de registros que pueden estar en fase de actualización o bloqueados simultáneamente.

Ambos parámetros (especialmente `MaxNoOfConcurrentOperations`) son objetivos probables para que los usuarios especifiquen valores específicos y no usen el valor por defecto. El valor por defecto es para sistemas con transacciones pequeñas, para asegurar que no usan excesiva cantidad de memoria.

- `[NDBD]MaxNoOfConcurrentTransactions`

  Para cada transacción activa en el cluster debe haber un registro en uno de los nodos del cluster. La tarea de coordinar transacciones se envía entre los nodos: el número total de registros de transacciones en el cluster es el número de transacciones en cualquier nodo por el número de nodos en el cluster.

  Los registros de transacciones se guardan en servidores MySQL individuales. Normalmente al menos hay un registro de transacción por conexión que usa cualquier tabla en el cluster. Por esta razón, debe asegurarse que hay más registros de transacciones en el cluster que conexiones concurrentes en todos los servidores MySQL del cluster.

  Este parámetro debe inicializarse al mismo valor para todos los nodos del cluster.

  Cambiar este parámetro nunca es seguro y puede causar que un cluster falle. Cuando un nodo falla uno de los nodos (el nodo superviviente más antiguo) levanta el estado de transacción de todas las transacciones que haya en funcionamiento en todos los nodos que han fallado cuando ha ocurrido el fallo. Por lo tanto es importante que este nodo tenga tantos registros de transacción como el nodo que ha fallado.

  El valor por defecto para este parámetro es 4096.

- `[NDBD]MaxNoOfConcurrentOperations`

  Es una buena idea ajustar el valor de este parámetro según el tamaño y número de transacciones. Cuando se realizan transacciones de sólo unas cuantas operaciones y que no involucren muchos

registros, no hay necesidad de inicializar este parámetro con un valor muy alto. Cuando realiza transacciones grandes involucrando muchos registros necesita cambiar a un valor mayor este parámetro.

Los registros se mantienen para cada transacción que actualice los datos del cluster, tanto en el coordinador de la transacción como en los nodos donde se hacen las actualizaciones. Estos registros contienen información de estado necesaria para encontrar registros UNDO para deshacer cambios, bloquear colas y otros propósitos.

Este parámetro debe inicializarse al número de registros a actualizar simultáneamente en transacciones, dividido por el número de nodos de datos del cluster. Por ejemplo, en un cluster que tenga 4 nodos de datos y del que se espera que trate 1,000,000 de actualizaciones concurrentes usando transacciones debe inicializar este valor a 1000000 / 4 = 250000.

Leer consultas que hacen bloqueos hace que se creen registros. Se reserva un espacio extra en los nodos individuales para los casos en que la distibución no es perfecta entre los nodos.

Cuando las consultas usan el índice hash único, hay dos registros de operación usados para registrar la transacción. El primer registro representa la lectura en la tabla índice y el segundo trata la operación en la tabla base.

El valor por defecto para este parámetro es 32768.

Este parámetro trata dos valores que se pueden configurar separadamente. El primero de ellos especifica cuántos registros de operación se tienen que situar en el coordinador de la transacción. La segunda parte especifica cuántos registros de operación serán locales en la base de datos.

Una transacción muy grande realizada en un cluster de 8 nodos requiere tantos registros de operación en el coordinador de transacción como lecturas, actualizaciones y operaciones de borrado involucradas en la transacción. Sin embargo, los registros de operación se envían a los 8 nodos. Por lo tanto, si es necesario configurar el sistema para una transacción muy grande, es buena idea configurar las dos partes por separado. `MaxNoOfConcurrentOperations` se usará siempre para calcular el número de registros de operación en la porción del coordinador de transacción del nodo.

Es importante tener una idea de los requerimientos de memoria para registros de operaciones. Actualmente es aproximadamente 1KB por registro.

- `[NDBD]MaxNoOfLocalOperations`

  Por defecto, este parámetro se calcula como 1.1 * `MaxNoOfConcurrentOperations` que encaja en sistemas con muchas transacciones simultáneas, ninguna de ellas demasiado grande. Si hay una necesidad de tratar una transacción muy grande de golpe y hay muchos nodos, es buena idea sobreescribir el valor por defecto especificando explícitamente este parámetro.

Almacenamiento temporal de transacciones

El siguiente conjunto de parámetros se usa para determinar almacenamiento temporal al ejecutar una consulta que es parte de una transacción de Cluster. Todos los registros se liberan al completar la consulta y el cluster espera para un commit o rollback.

Los valores por defecto de estos parámetros son adecuados para la mayoría de situaciones. Sin embargo, los usuarios con necesidad de soportar transacciones que involucren gran cantidad de registros u operaciones pueden necesitar incrementarlos para activar un mejor paralelismo en el sistema, mientras que los usuarios cuyas aplicaciones necesitan transacciones relativamente pequeñas pueden decrementar los valores para ahorrar memoria.

- `[NDBD]MaxNoOfConcurrentIndexOperations`

Para consultas usando un índice hash único, el conjunto temporal de registros de operaciones se usa durante la fase de ejecución de una consulta. Este parámetro delimita el tamaño del conjunto de registros. Así este registro sólo se reserva mientras se ejecuta una parte de una consulta, en cuanto ha acabado la ejecución se libera el registro. El estado necesario para tratar abortos y commits se trata mediante los registros operacionales normales donde el tamaño del pool se asigna por el parámetro `MaxNoOfConcurrentOperations`.

El valor por defecto de este parámetro es 8192. Sólo en casos raros de paralelismo extremadamente alto usando índices hash únicos debería ser necesario incrementar este valor. Usar un valor menor es posible y puede ahorrar memoria si el DBA está seguro que un cierto grado de paralelismo no es necesario en el cluster.

- `[NDBD]MaxNoOfFiredTriggers`

  El valor por defecto de `MaxNoOfFiredTriggers` es 4000, que es suficiente para la mayoría de situaciones. En algunos casos puede decrementarse si el DBA está seguro que el cluster no necesita un paralelismo alto.

  Se crea un registro cuando se realiza una operación que afecta un índice hash único. Insertar o borrar un registro en una tabla con índices hash únicos o actualizar una columna que es parte de un índice hash único provoca una inserción o borrado en la tabla índice. El registro resultante se usa para representar esta operación de la tabla índice mientras se espera a que la operación original se complete. Esta operación tiene una vida corta pero puede requerir un gran número de registros en su pool para situaciones en que muchas operaciones de escritura paralelas en una tabla base que contenga un conjunto de índices hash únicos.

- `[NDBD]TransactionBufferMemory`

  La memoria afectada por este parámetro se usa para tracear operaciones disparadas al actualizar tablas índice y leer índices únicos. Esta memoria se usa para almacenar la información de clave y columna para estas opraciones. Es muy raro que el valor de este parámetro necesite alterarse.

  Las operaciones normales de lectura y escritura usan un búffer similar, cuyo uso es incluso de vida más corta. El parámetro de tiempo de compilación `ZATTRBUF_FILESIZE` (que se encuentra en `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) es 4000*128 bytes (500KB). Un buffer similar para información de claves, `ZDATABUF_FILESIZE` (también en `Dbtc.hpp`) contiene 4000*16 = 62.5KB de tamaño de búffer. `Dbtc` es el módulo que trata la coordinación de transacciones.

Escaneos y búffering

Hay parámtros adicionales en el módulo `Dblqh` (en `ndb/src/kernel/blocks/Dblqh/Dblqh.hpp`) que afecta a lecturas y actualizaciones. Incluyen `ZATTRINBUF_FILESIZE`, por defecto 10000*128 bytes (1250KB) y `ZDATABUF_FILE_SIZE`, por defecto 10000*16 bytes (unos 156KB) de tamaño de buffer. Hasta hoy, no ha habido ningún reporte de usuarios ni resultados de nuestros tests que sugieran que deba incrementarse estos límites en tiempo de compilación.

El valor por defecto de `TransactionBufferMemory` es 1MB.

- `[NDBD]MaxNoOfConcurrentScans`

  Este parámetro se usa para controlar el número de escaneos paralelos que pueden realizarse en el cluster. Cada coordinador de transacción puede tratar el número de escaneos paralelos definidos por este parámetro. Cada consulta de escaneo se realiza escaneando todas las particiones en paralelo. Cada escaneo de partición usa un registro de escaneo en el nodo donde se encuentra la partición, siendo el número de registros el valor de este parámetro por el número de nodos. El clúster debe ser

capaz de sostener `MaxNoOfConcurrentScans` escaneos concurrentes de todos los nodos en el cluster.

Los escaneos se realizan en dos casos. El primero cuando no existe un índice hash u ordenado para tratar la consulta, en cuyo caso la consulta se ejecuta realizando un escaneo de la tabla completa. El segundo caso se encuentra cuando no hay índice hash para soportar la consulta pero hay un índice ordenado. Usar el índice ordenado significa ejecutar un escaneo de rango paralelo. Como el orden se mantiene en las particiones locales sólo, es necesario realizar el escaneo de índice en todas las particiones.

El valor por defecto de `MaxNoOfConcurrentScans` es 256. El valor máximo es 500.

Este parámetro especifica el número de escaneos posibles en el coordinador de transacción. Si el número de registros de escaneos locales no se proporciona, se calcula como el producto de `MaxNoOfConcurrentScans` y el número de nodos de datos en el sistema.

- `[NDBD]MaxNoOfLocalScans`

  Especifica el número de registros de escaneo locales si varios escaneos no están paralelizados completamente.

- `[NDBD]BatchSizePerLocalScan`

  Este parámetro se usa para calcular el número de registros bloqueados que necesitan estar ahí para tratar varias operaciones de escaneo concurrentes.

  El valor por defecto es 64; este valor tiene una fuerte conexión con `ScanBatchSize` definido en los nodos SQL.

- `[NDBD]LongMessageBuffer`

  Este es un buffer interno usado para pasar mensajes entre nodos individuales y entre nodos. Aunque es muy improbable que este valor deba cambiarse, es configurable. Por defecto es 1MB.

Registro y establecer Checkpoins

- `[NDBD]NoOfFragmentLogFiles`

  Este parámetro especifica el tamaño del fichero de log REDO del nodo. Los ficheros de log REDO se organizan en un anillo. Es muy importante que el fichero primero y último del log (a veces conocidos como "cabeza" y "cola", respectivamente) no se encuentren; cuando se aproximan el uno al otro demasiado, el nodo empieza a abortar todas las transacciones que realizan actualizaciones debido a la falta de espacio para nuevos registros de log.

  Un registro de log REDO no se elimina hasta que se han completado tres checkpoints locales desde la inserción del último registro del log. Establecer checkpoints frecuentemente viene determinado por su propio conjunto de parámetros de configuración discutidos en este capítulo.

  El valor por defecto del parámetro es 8, que significa 8 conjuntos de 4 ficheros de 16MB para un total de 512MB. En otras palabras, el espacio de log REDO debe reservarse en bloques de 64MB. En escenarios que necesitan muchas actualizaciones, el valor de `NoOfFragmentLogFiles` puede necesitar ser tan grande como 300 o incluso mayor para proporcionar suficiente espacio para logs REDO.

  Si se hacen pocos checkpoings y hay tantas escrituras en la base de datos que los ficheros de log se llenan y la cola de logs no puede cortarse sin recuperación jeapo rdising , se abortan todas las

transacciones con un código de error interno 410, o `Out of log file space temporarily`. Esta condición prevalecerá hasta que se complete un checkpoint y se mueva la cola de log.

- `[NDBD]MaxNoOfSavedMessages`

  Este parámetro especifica el máximo número de ficheros de traceo que se mantienen antes de sobreescribir los antiguos. Los ficheros de traceo se generan cuando, por alguna razón, el nodo cae.

  Por defecto son 25 ficheros de traceo.

Objetos de metadatos

El siguiente conjunto de parámetros definen tamaños de pool para objetos de metadatos. Es necesario definir el máximo número de atributos, tablas, índices y disparadores usados por índices, eventos y replicaciones entre clusters.

- `[NDBD]MaxNoOfAttributes`

  Define el número de atributos que pueden definirse en el cluster.

  El valor por defecto para este parámetro es 1000, cuyo valor mínimo posible es 32. No hay máximo. Cada atributo consume acerca de 200 bytes de almacenamiento por nodo debido a que todos los metadatos están replicados en los servidores.

- `[NDBD]MaxNoOfTables`

  Se reserva un objeto tabla para cada tabla, índice hash único e índice ordenado. Este parámetro especifica el máximo número de objetos tabla para el cluster en total.

  Para cada atributo que tiene un tipo de datos `BLOB` se usa una tabla extra para almacenar la mayoría de datos `BLOB` . Estas tablas deben tenerse en cuenta al definir el número total de tablas.

  El valor por defecto es 128. El mínimo es 8 y el máximo es 1600. Cada objeto tabla consume aproximadamente 20KB por nodo.

- `[NDBD]MaxNoOfOrderedIndexes`

  Para cada índice ordenado en este cluster, un objeto se reserva que describe lo que se indexa y sus segmentos de almacenamiento. Por defecto cada índice definido así también define un índice ordenado. Cada índice único y clave primaria tiene un índice ordenado e índice hash.

  El valor por defecto es 128. Cada objeto consume unos 10KB de datos por nodo.

- `[NDBD]MaxNoOfUniqueHashIndexes`

  Para cada índice único que no sea clave primaria, se reserva una tabla especial que mapea la clave única con la clave primaria de la tabla indexada. Por defecto hay un índice ordenado que se define para cada índice único. Para evitarlo, debe usar la opción `USING HASH` al definir el índice único.

  El valor por defecto es 64. Cada índice consume aproximadamente 15KB por nodo.

- `[NDBD]MaxNoOfTriggers`

  Los disparadores internos de actualización, inserción y borrado se reservan para cada índice hash único. (Esto significa que se crean 3 disparadores para cada índice hash.) Sin embargo, un índice *ordenado* necesita sólo un objeto disparador. Las copias de seguridad también usan tres objetos disparadores para cada tabla normal en el cluster.

Nota: Cuando se soporta replicación entre clusters, esta hará uso de disparadores internos.

Este parámetro inicializa el número máximo de objetos disparadores en el cluster.

El valor por defecto es 768.

- `[NDBD]MaxNoOfIndexes`

  Este parámetro está obsoleto en MySQL 5.0; debe usar `MaxNoOfOrderedIndexes` y `MaxNoOfUniqueHashIndexes` en su lugar.

  Este parámetro se usa sólo en índices hash únicos. Debe haber un registro en este pool para cada índice hash único definido en el cluster.

  El valor por defecto es 128.

Parámetros booleanos

El comportamiento de los nodos de datos se ve afectado por un conjunto de parámetros booleanos. Estos parámetros pueden especificarse como `TRUE` poniéndolos a `1` o `Y`, y como `FALSE` poniéndolos a `0` o `N`.

- `[NDBD]LockPagesInMainMemory`

  Para algunos sistemas operativos, incluyendo Solaris y Linux, es posible bloquear el proceso en memoria y evitar el swapping de disco. Esto puede usarse para ayudar a garantizar las características de tiempo real del cluster.

  Por defecto, esta característica está desactivada.

- `[NDBD]StopOnError`

  Este parámetro especifica si un proceso NDBD debe salir o reiniciarse automáticamente cuando se encuentra una condición de error.

  Esta característica está activada por defecto.

- `[NDBD]Diskless`

  Es posible especificar tablas cluster como **diskless**, que significa que no se hacen checkpoints en disco y que no se registran. Estas tablas sólo existen en memoria. Una consecuencia de usar estas tablas es que no se pueden mantener tras una caida. Sin embargo, cuando el modo diskless está operativo se puede ejecutar `ndbd` en una máquina sin disco.

  Nota: Actualmente, esta característica hace que el cluster entero funcione en modo diskless.

  Actualmente, cuando está característica está activa, las copias de seguridad se realizan pero los datos no se almacenan. En futuras versiones esperamos hacer la copia de seguridad diskless un parámetro separado y configurable.

  Esta característica está activada mediante poner `Diskless` a `1` o `Y`. `Diskless` está desactivado por defecto.

- `[NDBD]RestartOnErrorInsert`

  Esta característica es accesible sólo al compilar la versión de debug cuando es posible insertar errores en la ejecución de bloques de código individuales como parte del testeo.

  Por defecto, esta característica está desactivada.

Controlar timeouts, intervalos y paginación de disco

Hay un número de parámetros que especifican timeouts e intervalos entre varias acciones en nodos de datos del culster. La mayoría de los timeouts se especifican en milisegundos. Cualquier excepción se menciona a continuación.

- `[NDBD]TimeBetweenWatchDogCheck`

  Para evitar que el flujo principal quede en un bucle sin fin en algún punto, un “watchdog” chequea el flujo principal. Este parámetro especifica el número de milisegundos entre chequeos. Si el proceso sigue siendo el mismo tras tres chequeos, lo mata el flujo watchdog.

  Este parámetro puede cambiarse fácilmente para propósitos de experimentación o para adaptar las condiciones locales. Puede especificarse por nodo aunque parece que no hay grandes razones para hacerlo.

  El timeout por defecto es de 4000 milisegundos (4 segundos).

- `[NDBD]StartPartialTimeout`

  Este parámetro especifica lo que esperará el cluster para que arranquen los nodos de almacenamiento cuando se invoca la rutina de inicialización . Este timeout se usa para evitar un arranque parcial del cluster cuando es posible.

  El valor por defecto es de 30000 milisegundos (30 segundos). `0` significa timeout eterno; en otras palabras, el cluster puede arrancar sólo si todos los nodos están disponibles.

- `[NDBD]StartPartitionedTimeout`

  Si el cluster está preparado para arrancar tras esperar durante `StartPartialTimeout` milisegundos pero todavía es posible en un estado particionado, el cluster espera hasta que este timeout pasa.

  El timeout por defecto es 60000 milisegundos (60 segundos).

- `[NDBD]StartFailureTimeout`

  Si un nodo de datos no ha completado su secuencia de arranque en el tiempo especificado por este parámetro, el arranque del nodo falla. Poner este parámetro a `0` significa que no se aplica ningún timeout para el nodo de datos.

  El valor por defecto es 60000 milisegundos (60 segundos). Para los nodos de datos con cantidades de datos extremadamente grandes, este parámetro debe incrementarse. Por ejemplo, en el caso de un nodo de almacenamiento que contenga varios gigabytes de datos, un perido de 10-15 minutos (esto es, 600,000 a 1,000,000 milisegundos) puede ser necesario para realizar una reinicialización de nodo.

- `[NDBD]HeartbeatIntervalDbDb`

  Uno de los métodos primarios de descubrimiento de nodos fallidos es el uso de heartbeats. Este parámetro refleja la frecuencia con que las señales heartbeat se envían y con que frecuencia se espera su recepción. Tras perder tres intervalos heartbeat seguidos, el nodo se declara muerto. Así el máximo tiempo para descubrir un fallo a través del mecanismo heartbeat es cuatro veces el intervalo heartbeat.

  El intervalo de heartbeat por defecto es de 1500 milisegundos (1.5 segundos). Este parámetro no debe cambiarse drásticamente y no debe variar demasiado entre nodos. Si un nodo usa 5000 milisegundos y el nodo observador usa 1000 milisegundos entonces óbviamente se declarará muerto rápidamente. Este parámetro puede cambiarse durande una actualización de software en línea pero sólo en pequeños incrementos.

- [NDBD]HeartbeatIntervalDbApi

  Cada nodo de datos envía señales heartbeat a cada MySQL server (nodo SQL) para segurar que permanece en contacto. Si un MySQL server falla para enviar un heartbeat a tiempo se declara “muerto”, en cuyo caso todas las transacciones activas se completan y todos los recursos se liberan. El nodo SQL no puede reconectar hasta que todas las actividades iniciadas por la instancia MySQL prévia se ha completado. El criterio de tres heartbeats para esta discriminación es la misma que se describe en HeartbeatIntervalDbDb.

  El intervalo por defecto es de 1500 milisegundos (1.5 segundos). Este intervalo puede variar entre nodos de datos individuales ya que cada nodo de almacenamiento vigila los servidores MySQL conectados a ellos, independientemente de todos los otros nodos de datos..

- [NDBD]TimeBetweenLocalCheckpoints

  Este parámetro es una excepción en que no especifica un tiempo para esperar antes de arrancar un nuevo checkpoint local; en lugar de esto, se usa para asegurarse que los checkpoints locales no se realizan en un cluster donde tienen lugar relativamente pocas actualizaciones. En la mayoría de clusters con altos ratios de actualización, es probable que se arranquen checkpoints locales inmediatamente tras que se hayan completado los previos.

  El tamaño de todas las operaciones de escritura ejecutadas desde el arranque de los chekpoints locales prévios. Este parámetro es excepcional ya que se especifica como el logaritmo en base 2 del número de palabras de 4 bytes, así que el valor por defecto 20 significa 4MB ($4 * 2^{20}$) de operaciones de escritura, 21 significaría 8MB, y así hasta un máximo de 31, que son 8GB de operaciones de escritura.

  Todas las operaciones de escritura en el cluster se añaden juntas. Poner TimeBetweenLocalCheckpoints a 6 o menos significa que los checkpoints locales se ejecutarán contínuamente sin pausa, independientemente de la carga de trabajo del cluster.

- [NDBD]TimeBetweenGlobalCheckpoints

  Cuando se hace un commit de una transacción, se realiza en memoria principal en todos los nodos en que los datos se replican. Sin embargo, los registros del log de transacción no se vuelcan en disco como parte del commit. El razonamiento de este comportamiento es que tener la transacción con un commit en al menos dos máquinas autónomas debe cumplir estándars razonables para durabilidad.

  También es importante asegurarse que incluso el peor caso — una caída completa del cluster — se trata apropiadamente . Para garantizar que esto ocurre, todas las transacciones que tienen lugar dentro de un intervalo dado se ponen en un checkpoint global, que puede entenderse como un conjunto de transacciones volcadas en disco. En otras palabras, como parte del proceso de commit, una transacción se guarda en el grupo de checkpoint global; posteriormente, este registro de log de grupo se vuelca en disco, y luego se guarda en disco el completo grupo de transacciones en todas las máquinas del cluster.

  Este parámetro define el intervalo entre checkpoints globales. Por defecto son 2000 milisegundos.

- [NDBD]TimeBetweenInactiveTransactionAbortCheck

  El tratamiento de time-outs se realiza chequeando un timer en cada transacción una vez por cada intervalo especificado por este parámetro. Por lo tanto, si este parámetro se pone a 1000 milisegundos, cada transacción se chequea para hacer un time out una vez por segundo.

  El valor por defecto para este parámetro es 1000 milisegundos (1 segundo).

- [NDBD]TransactionInactiveTimeout

Si la transacción no está realizando ninguna consulta pero esta esperando más datos de entrada del usuario, este parámetro indica el tiempo máximo que el usuario puede esperar entes de abortar la transacción.

Por defecto este parámetro es cero (no timeout). Para una base de datos en tiempo real que necesita asegurarse que las transacciones no mantienen bloqueos demasiado tiempo se debe inicializar con un valor mucho más pequeño. La unidad es milisegundos.

- `[NDBD]TransactionDeadlockDetectionTimeout`

  Cuando un nodo ejecuta una consulta que implique una transacción, el nodo espera para que respondan los otros nodos en el cluster antes de continuar. Puede que haya un fallo al responder por cualquiera de las siguientes razones:

  1. El nodo está “muerto”
  2. La operación ha entrado en una cola de bloqueo
  3. El nodo con la petición de realizar la acción puede estar muy sobrecargado.

  Este parámetro de timeout indica cuánto espera el coordinador de la transacción para la ejecución de la consulta por otro nodo antes de abortar la transacción, y es importante para tratamiento de fallo de nodo y detección de deadlocks. Ponerlo a un valor muy alto puede causar comportamiento no deseables en situaciones que impliquen deadlocks y fallos de nodos.

  El timeout por defecto es 1200 milisegundos (1.2 segundos).

- `[NDBD]NoOfDiskPagesToDiskAfterRestartTUP`

  Al ejecutar un checkpoint local el algoritmo vuelca todas las páginas de datos a disco. Hacer esto tan rápidamente como sea posible sin ninguna moderación es posible que imponga demasiada carga de procesador, red y disco. Para controlar la velocidad de escritura, este parámetro especifica cuántas paginas se escriben en 100 milisegundos. En este contexto, una “página” se define como 8KB; por lo tanto, este parámetro se especifica en unidades de 80KB por segundo. Por lo tanto, poner `NoOfDiskPagesToDiskAfterRestartTUP` a un valor de 20 significa escribir 1.6MB de páginas de datos en disco cada segundo durante checkpoints locales. Este valor incluye escribir registros del log UNDO para páginas de datos; esto es, este parámetro trata la limitación de escrituras de memoria de datos. Los registros del log UNDO para páginas de índice se tratan mediante los parámetros `NoOfDiskPagesToDiskAfterRestartACC`. (Consulte la entrada para `IndexMemory` para información acerca de páginas de índice.)

  Resumiendo, este parámetro especifique con qué velocidad se realizan los checkpoints locales, y opera en conjunción con `NoOfFragmentLogFiles`, `DataMemory`, y `IndexMemory`.

  El valor por defecto es 40 (3.2MB de páginas de datos por segundo).

- `[NDBD]NoOfDiskPagesToDiskAfterRestartACC`

  Este parámetro usa las mismas unidades que `NoOfDiskPagesToDiskAfterRestartTUP` y actúa de forma similar, pero limita la velocidad de escritura de páginas índices de memoria índice.

  El valor por defecto de este parámetro es 20 páginas índice de memoria por segundo (1.6MB por segundo).

- `[NDBD]NoOfDiskPagesToDiskDuringRestartTUP`

Este parámero parecido a `NoOfDiskPagesToDiskAfterRestartTUP` y `NoOfDiskPagesToDiskAfterRestartACC`, sólo lo hace en función de los checkpoints locales ejecutados en el nodo donde se reinicia un nodo. Como parte de todo reinicio de nodo siempre se realiza un checkpoint. Durante el reinicio de un nodo es posible escribir en disco a una velocidad superior que otras veces, ya que se realizan menos actividades en el nodo.

Este parámetro cubre páginas escritas de memoria de datos.

El valor por defecto es 40 (3.2MB por segundo).

- `[NDBD]NoOfDiskPagesToDiskDuringRestartACC`

  Controla el número de páginas de memoria índice que pueden escribirse en disco durante la fase de checkpoint local del reinicio de un nodo.

  Como con `NoOfDiskPagesToDiskAfterRestartTUP` y `NoOfDiskPagesToDiskAfterRestartACC`, los valores para este parámetro se expresan en términos de 8KB páginas escritas en 100 milisegundos (80KB/segundo).

  El valor por defecto es 20 (1.6MB por segundo).

- `[NDBD]ArbitrationTimeout`

  Este parámetro especifica el tiempo que esperará el nodo de datos en respuesta al árbitro. Si se excede, se asume que la red se ha partido.

  El valor por defecto es de 1000 milisegundos (1 segundo).

Buffering y Logueo

Varios parámetros de configuración se corresponden a antiguos parámetros en tiempo de compilación que todavía están disponibles. Esto permite al usuario avanzado tener más control sobre los recursos usados por los procesos nodo y para ajustar varios tamaños de buffer según sea necesario.

Estos buffers se usan como front ends para el sistema de ficheros cuando se escriben los registros de log en disco. Si el nodo está ejecutándose en modo diskless, y estos parámetros pueden cambiarse a sus valores mínimos sin penalización debido al hecho que las escrituras en disco se "falsean" por parte la capa de abstracción del sistema de ficheros del motor de almacenamiento NDB.

- `[NDBD]UndoIndexBuffer`

  Este buffer se usa durante los checkpoints locales. El motor NDB usa un esquema de recuperación basado en consistencia de checkpoints en conjunción con un log REDO operacional. Para producir un checkpoint consistente con bloquear el sistema entero para escrituras, el logueo de UNDO se hace mientras se realiza el checkpoint local. El logueo UNDO se activa en un fragmento de tabla único cada vez. Esta optimización es posible porque las tablas se almacenan completamente en memoria.

  El buffer UNDO índice se usa para las actualizaciones en el índice de clave hash primaria. Las inserciones y borrados modifican el índice hash; el motor NDB escribe registros en el log UNDO que mapean todos los cambios físicos en una página índice de forma que pueden deshacerse al reiniciar al sistema. También loguea todas las operaciones de inserciones activas para cada fragmento al arranque del checkpoint local.

  Lee y actualiza el conjunto de bits de bloquea y actualiza una cabecera en la entrada del índice hash. Estos cambios los trata el algoritmo de escritura de páginas para asegurar que estas operaciones no necesitan logueo UNDO.

Este búffer es de 2MB por defecto. El valor mínimo es de 1MB, y para la mayoría de aplicaciones el mínimo es suficiente. Para aplicaciones que realicen un número de inserciones y borrados muy alto junto con transacciones muy grandes y claves primaria grandes, puede ser necesario incrementar el tamaño de este buffer. Si el buffer es muy pequeño, el motor NDB realiza un código de error interno 677 `Index UNDO buffers overloaded`.

- `[NDBD]UndoDataBuffer`

  El buffer de datos UNDO juega el mismo rol que el buffer de indice UNDO, excepto que se usa a con la memoria de datos en lugar de la de índice. Este buffer se usa durante la fase de checkpoint local de un fragmento para inserciones, borrados y actualizaciones.

  Como UNDO las entradas de log tienden a crecer mayores mientras más operaciones se loguean, este buffer también es mayor que su contraparte de memoria índice, con un valor por defecto de 16MB.

  Para algunas aplicaciones esta cantidad de memoria puede ser innecesariamente grande. En tales casos es posible decrementar este tamaño a un mínimo de 1MB.

  Ráramente es necesario incrementar el tamaño de este buffer. Si hay tal necesidad, es una buena idea chequear si el disco puede tratar la carga causada por la actividad de actualización de la base de datos. Una falta de espacio de disco no puede evitarse incrementando el tamaño de este buffer.

  Si este buffer es demasiado pequeño y se congestiona, el motor NDB realiza un código de error interno 891 `Data UNDO buffers overloaded`.

- `[NDBD]RedoBuffer`

  Todas las actividades de actualización necesitan loguearse. Este log hace posible rehacer estas actualizaciones cuando el sistema se reinicia. El algoritmo de recuperación NDB usa un checkpoint “fuzzy” de los datos junto con el log UNDO, luego aplica el log REDO para aplicar todos los cambios hasta el punto de restauración.

  Este buffer es de 8MB por defecto. El valor mínimo es 1MB.

  Si este buffer es demasiado pequeño, el motor NDB realiza un código de error 1221 `REDO log buffers overloaded`.

Al administrar el cluster, es muy importante ser capaz de controlar el número de mensajes de log enviados por distintos tipos de eventos a `stdout`. Hay 16 niveles posibles de evento (numerados de 0 a 15). Realizar un reporte de evento para una categoría de evento dada a nivel 15 significa que todos los reportes de evento en esta categoría se envían a `stdout`; ponerlo a 0 significa que no habrá reportes de eventos en esta categoría.

Por defecto, sólo el mensaje de arranque se envía `stdout`, con los valores por defecto de los niveles de reporte pendientes puestos a 0 . La razón para esto es que estos mensajes también se envían al log de administración del cluster.

Un conjunto de niveles análogo puede ponerse para el cliente de administración para determinar qué niveles de evento registrar en el log de cluster.

- `[NDBD]LogLevelStartup`

  Reportar niveles para eventos generados durante el arranque del proceso.

  El nivel por defecto es 1.

- `[NDBD]LogLevelShutdown`

El nivel de reporte para eventos generados como parte de un cierre de un nodo.

El nivel por defecto es 0.

- `[NDBD]LogLevelStatistic`

  El nivel de reporte para eventos estadísticos tales como número de lecturas de clave primaria, número de actualizaciones, número de inserciones, información acerca del uso de búffer, y así.

  El nivel por defecto es 0.

- `[NDBD]LogLevelCheckpoint`

  Nivel de reporte para eventos generados por los checkpoints locales y globales.

  El nivel por defecto es 0.

- `[NDBD]LogLevelNodeRestart`

  Nivel de reporte para eventos generados durante reinicio de nodo.

  El nivel por defecto es 0.

- `[NDBD]LogLevelConnection`

  El nivel de reporte para eventos generados por conexiones entre nodos del cluster.

  El nivel por defecto es 0.

- `[NDBD]LogLevelError`

  Nivel de reporte para eventos generados por errores y advertencias por el cluster global. Estos errores no causan ningún fallo de nodo pero se considera que vale la pena de reportar.

  El nivel por defecto es 0.

- `[NDBD]LogLevelInfo`

  Nivel de reporte para eventos generados por información acerca del estado general del cluster.

  El nivel por defecto es 0.

Parámetros de copia de seguridad

Los parámetros de esta sección definen buffers de memoria para ejecución de copias de seguridad en línea.

- `[NDBD]BackupDataBufferSize`

  Al crear una copia de seguridad, hay dos búffers usados para enviar data a disco. El buffer de copia de seguridad de datos se usa para rellenar con datos registrados al escanear las tablas de un nodo. Una vez que este buffer se ha rellenado al nivel especificado por `BackupWriteSize` (vea a continuación), las páginas se envían a disco. Al volcar los datos a disco, el proceso de copia de seguridad puede continuar rellenando este buffer hasta que se queda sin espacio. Cuando ocurre, el proceso de copia de seguridad pausa el escaneo y espera hasta que se completan algunas escrituras en disco y han liberado memoria de forma que pueda continuar el escaneo.

  El valor por defecto de este paráemtro es 2MB.

- `[NDBD]BackupLogBufferSize`

  El buffer de log de copia de seguridad tiene un rol similar al jugado por el buffer de copia de seguridad de datos , excepto que se usa para generar un log de todas las escrituras de tabla realizadas durante la ejecución de una copia de seguridad. Los mismos principios se aplican para escribir estas páginas como con el buffer de datos de copia de seguridad, excepto que cuando no hay más espacio en el buffer de log de copia de seguridad, la copia falla. Por esta razón, el tamaño del buffer de log de copia de seguridad debe ser lo bastante grande para tratar la carga causada por las actividades de escritura mientras se realiza la copia de seguridad.

  El valor por defecto de este parámetro debe ser suficiente para la mayoría de aplicaciones. De hecho, es más fácil que un fallo de copia de seguridad sea causado por velocidad de escritura en disco insuficiente que por que el buffer de log esté lleno. Si el subsistema de disco no está configurado para la carga de escritura causada por las aplicaciones, el cluster posiblemente será capaz de realizar las operaciones deseadas.

  Es preferible configurar nodos del cluster de forma que el procesador sea el cuello de botella en lugar que los discos o las conexiones de red.

  El valor por defecto es 2MB.

- `[NDBD]BackupMemory`

  Este parámetro es la suma de `BackupDataBufferSize` y `BackupLogBufferSize`.

  El valor por defecto es 2MB + 2MB = 4MB.

- `[NDBD]BackupWriteSize`

  Este parámetro especifica el tamaño de los mensajes escritos en disco por los buffers de log y datos de copia de seguridad.

  El valor por defecto es 32KB.

16.4.4.6. Definir los servidores MySQL (nodos sql) en un MySQL Cluster

La sección `[MYSQLD]` del fichero `config.ini` define el comportamiento de los servidores MySQL (nodos SQL) usados para acceder a los datos del cluster. Ninguno de los parámetros mostrados es necesario. Si no se proporciona máquina ni nombre de equipo, entonces cualquier equipo puede usar este nodo SQL.

- `[MYSQLD]Id`

  Este valor se usa como dirección del nodo por todos los mensajes internos del cluster, y debe ser un entero entre 1 y 63. Cada nodo del cluster debe tener una identidad única dentro del cluster.

- `[MYSQLD]ExecuteOnComputer`

  Se refiere a una de las máquinas definidas en la sección `[COMPUTER]` del fichero de configuración.

- `[MYSQLD]ArbitrationRank`

  Este parámetro se usa para definir qué nodos pueden actuar como árbitros. Pueden serlo los nodos MGM y SQL. Un valor de 0 significa que el nodo dado nunca se usará como árbitro, un valor de 1 le da al nodo alta prioridad como árbitro, y un valor de 2 le da baja prioridad. Una configuración normal usa el servidor de administración como árbitro, poniendo se `ArbitrationRank` a 1 (por defecto) y los nodos SQL a 0.

- `[MYSQLD]ArbitrationDelay`

Poner este parámetro con cualquier valor distinto a 0 (valor por defecto) significa que las respuestas al árbitro se retrasan el número indicado de milisegundos. Usualmente no es necesario cambiar este valor.

- `[MYSQLD]BatchByteSize`

  Para consultas traducidas en escaneos completos de tabla o escaneo de rangos de índices, es importante pare un mejor rendimiento tratar registros en batchs de tamaño apropiado. Es posible poner el tamaño adecuado en términos de número de registros (`BatchSize`) y en términos de bytes (`BatchByteSize`). El tamaño batch real viene limitado por ambos parámetros.

  La velocidad a la que se realizan las consultas puede variar más de un 50$ en función de cómo se ajusta este parámetro. En versiones futuras, el servidor MySQL realizará conjeturas sobre cómo ajustar estos parámetros según el tamaño batch basado en el tipo de consulta.

  Este parámetro se mide en bytes y por defecto es igual a 32KB.

- `[MYSQLD]BatchSize`

  Este parámetro se mide en número de registros y por defecto es 64. El tamaño máximo es 992.

- `[MYSQLD]MaxScanBatchSize`

  El tamaño batch es el tamaño de cada batch envíado por cada nodo de datos. La mayoría de escaneos se realizan en paralelo para proteger que el servidor MySQL reciba demasiados datos de demasiados nodos en paralelo; este parámetro ajusta el límite del tamaño total batch en todos los nodos.

  El valor por defecto de este parámetro se ajusta a 256KB. Su tamaño máximo es 16MB.

16.4.4.7. Conexiones TCP/IP de MySQL Cluster

TCP/IP es el mecanismo de transporte por defecto para establecer conexiones en MySQL Cluster. Normalmente no es necesario definir conexiones ya que el cluster automáticamente crea una conexión entre todos los nodos de datos, entre cada nodo de datos y los nodos MySQL server y entre cada nodo y el servidor de administración. (Para una excepción a esta regla consulte Sección 16.4.4.8, “Conexiones TCP/IP de MySQL Cluster usando conexiones directas”.)

Sólo es necesario definir una conexión para sobreescribir los parámetros por defecto de conexión. En tal caso es necesario definir al menos *`NodeId1`*, *`NodeId2`*, y los parámetros a cambiar.

Es posible cambiar los valores por defecto para estos parámetros cambiándolos en la sección `[TCP DEFAULT]` .

- `[TCP]`*`NodeId1`* , `[TCP]`*`NodeId2`*

  Para identificar una conexión entre dos nodos es necesario proporcionar los identificadores de nodo para ambos en la sección `[TCP]` del fichero de configuración.

- `[TCP]SendBufferMemory`

  TCP usa un buffer para almacenar todos los mensajes antes de realizar la llamada de envío del sistema operativo. Cuando este buffer llega a 64KB sus contenidos se envían; también se envían cuando se ha ejecutado una ronda de mensajes. Para tratar situaciones de sobrecarga temporal es posible definir un buffer de envio mayor. El tamaño por defecto del buffer de envío es 256KB.

- `[TCP]SendSignalId`

Para poder trazar un diagrama de mensaje distribuído es necesario identificar cada mensaje. Cuando este parámetro es `Y`, los IDs de mensaje se transportan por la red. Esta característica está desactivada por defecto.

- `[TCP]Checksum`

  Este parámetro es booleano (`Y`/`N` o `1`/`0`) , y está desactivado por defecto. Cuando está activado, los checksums de todos los mensajes se calculan antes de ponerlos en el buffer de envío. Esta característica se asegura que los mensajes no se corrompan mientras esperan en el buffer de envío, o por el mecanismo de transporte.

- `[TCP]PortNumber`

  (**OBSOLETO**.) Antiguamente especificada el número de puerto a usar para escuchar las conexiones de otros nodos. Este paráemtro no debe usarse más.

- `[TCP]ReceiveBufferMemory`

  Especifica el tamaño del buffer usado al recibir datos del socket TCP/IP . No hay necesidad de cambiar el valor por defecto de este parámetro de 64KB, excepto para ahorrar memoria.

16.4.4.8. Conexiones TCP/IP de MySQL Cluster usando conexiones directas

Ajustar el cluster usando conexiones directas entre nodos de datos requiere curzar explícitamente las direcciones IP de los nodos de datos conectados así en la sección `[TCP]` del fichero `config.ini` del cluster.

En el siguiente ejemplo, tenemos un cluster con al menos 4 equipos, uno para un servidor de administración, un nodo SQL, y dos nodos de datos. El cluster reside en la subred `172.23.72.*` de una LAN. Además de las conexiones de red usuales, los dos nodos de datos se conectan directamente usando un cable cruzado, y comunican entre ellos directamente usando direcciones IP en el rango IP `1.1.0.*` como se muestra:

```
# 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
```

El uso de conexiones directas entre nodos de datos puede mejorar la eficiencia global del cluster permitiendo a los nodos de datos cortocircuitar un equipo de Ethernet como un switch, hub o router, disminuyendo la latencia del cluster. Es importante tener en cuenta que para obtener el mayor rendimiento de las conexiones directas con más de 2 nodos de datos, necesitará tener una conexión directa entre cada datos de nodos y cada uno de los nodos de datos en el mismo grupo de nodos.

16.4.4.9. Conexiones de MySQL Cluster que comparten memoria

En MySQL 5.0, MySQL Cluster tata de usar transporte a través de memoria compartida y configurarla automáticamente cuando sea posible, principalmente donde más de un nodo se ejecuta concurrentemente en el mismo equipo del cluser. (En versiones anteriores de MySQL Cluster, los segmentos de memoria compartida se soportan sólo cuando el binario `-max` se compila usando `--with-ndb-shm`.) Cuando se define la memoria compartida explícitamente como método de conexión, es necesario definir como mínimo `NodeId1`, `NodeId2` y `ShmKey`. Todos los otros parámetros tienen valores por defecto que funciona bien en la mayoría de casos.

Nota: El soporte SHM debe considerarse experimental.

- `[SHM]NodeId1`, `[SHM]NodeId2`

  Para identificar una conexión entre dos nodos es necesario proporcionar identificadores para cada uno de ellos como `NodeId1` y `NodeId2`.

- `[SHM]ShmKey`

  Cuando se inicializan segmentos de memoria compartido, un ID de nodo se identifica para identificar unívocamente el segmento de memoria compartida para usar para la comunicación. Se expresa como un entero que no tiene valor por defecto.

- `[SHM]ShmSize`

  Cada conexión SHM tiene un segmento de memoria compartida dónde los nodos entre mensajes se guardan por parte del que envía y donde lo lee el receptor. El tamaño de este segmento lo define `ShmSize`. El valor por defecto es 1MB.

- `[SHM]SendSignalId`

  Para obtener la traza de la ruta de un mensaje distribuído, es necesario proporcionar un identificador único a cada mensaje. Poner este parámetro a `Y` hace que los IDs de mensajes se transporten sobre la red también. Esta característica está desactivada por defecto.

- `[SHM]Checksum`

  Este parámetro es `Y`/`N` , y está desactivado por defecto. Cuando se activa, se calculan los checksums para todos los mensajes y se guardan en el buffer de envío.

  Esta característica evita que los mensajes se corrompan mientras esperan en el buffer de envío. También sirve como chequeo para que no se corrompan los datos durante el transporte.

16.4.4.10. Conexiones de transporte SCI en MySQL Cluster

Se soporta el uso de transporte SCI en MySQL Cluster sólo cuando los binarios MySQL-Max se compilan usando `--with-ndb-sci=`*`/your/path/to/SCI`*. El *`path`* debe apuntar a un directorio que contiene como mínimo los directorios `lib` y `include` con las bibliotecas SISCI y ficheros de cabecera.

Además, SCI necesita hardware especializado.

Es muy recomendable usar SCI Transporters sólo para comunicación entre procesos `ndbd` . Además, usar SCI Transporters significa que los procesos `ndbd` nunca duermen. Por esta razón, los SCI Transporters deben usarse sólo en máquinas con al menos 2 CPUs dedicadas para usar con procesos `ndbd` . Al menos debe haber 1 CPU por proceso `ndbd` , con al menos 1 CPU en reserva para tratar actividades del sistema operativo.

- `[SCI]NodeId1`, `[SCI]NodeId2`

  Para identificar una conexión entre dos nodos es necesario proporcionar identificadores de nodo para cada uno de ellos, como `NodeId1` y `NodeId2`.

- `[SCI]Host1SciId0`

  Esto identifica el nodo ID SCI en el primer nodo del Cluster (identificado por `NodeId1`).

- `[SCI]Host1SciId1`

  Es posible preparar SCI Transporters para tratar fallos entre dos tarjetas SCI que deban usar redes separadas entre los nodos. Esto identifica el ID del nodo y la segunda tarjeta SCI a usar en el primer nodo.

- `[SCI]Host2SciId0`

  Esto identifica el ID del nodo SCI en el segundo nodo del Cluster (identificado por `NodeId2`).

- `[SCI]Host2SciId1`

  Cuando dos tarjetas SCI para proporcionar tolerancia a fallos, este parámetro identifica la segunda tarjeta SCI a usar en el segundo nodo.

- `[SCI]SharedBufferSize`

  Cada SCI transporter tiene un segmento de memoria compartido usado para la comunicación entre los dos nodos. Especificar el segmento de este valor por defecto de 1 MB debería ser suficiente para la mayoría de aplicaciones. Usar un valor menor puede llevar a problemas al realizar muchas inserciones paralelas; si un buffer compartido es demasiado pequeño, puede resultar en una caída del proceso `ndbd` .

- `[SCI]SendLimit`

  Un buffer pequeño en frente del medio SCI almacena mensajes antes de transmitirlos en la red SCI. Por defecto se pone 8kB. Nuestras pruebas muestras que el rendimiento es mejor con 64 kB pero 16kB alcanza un pequeño porcentaje de esto, y hay poca ventaja si se incrementa más allá de 8kB.

- `[SCI]SendSignalId`

  Para tracear un mensaje distribuído es necesario identificar cada mensaje únicamente. Cuando este parámetro se pone a `Y`, los IDs de mensaje se transportan sobre la red. Esta característica se desactiva por defecto.

- `[SCI]Checksum`

  Este parámetro es un valor booleano, y está desactivado por defecto. Cuando `Checksum` está activado, los checksums se calculan para todos los mensajes antes de ponerlos en el buffer de envío. Esta característica evita que los mensajes se corrompan mientras esperan en el buffer de salida durante el transporte.

16.5. Gestión de procesos en MySQL Cluster

Entender cómo administrar MySQL Cluster requiere un conocimiento de cuatro procesos esenciales. En las siguientes secciones de este capítulo, cubrimos los roles jugados por estos procesos en un cluster, cómo usarlos, y qué opciones de arranque están disponibles para cada uno de ellos.

16.5.1. El uso del proceso del servidor MySQL para MySQL Cluster

`mysqld` es el proceso de MySQL server tradicional. `mysqld` debe compilarse con soporte para el motor NDB Cluster , como lo está en los binarios `-max` precompilados disponibles en mysql.com.

Si el binario `mysqld` se ha compilado de esta forma, el motor NDB Cluster todavía está desactivado por defecto. Tiene dos opciones para activar el motor NDB Cluster :

Use `--ndbcluster` como opción de arranque para `mysqld`

- Inserte una línea con `ndbcluster` en la sección `[mysqld]` en el fichero `my.cnf` .

Una forma sencilla de verificar que su servidor está ejecutando el motor `NDB Cluster` es ejecutar el comando `SHOW ENGINES` en MySQL Monitor (`mysql`). Debe ver el valor `YES` en el registro `NDBCLUSTER`; si ve `NO` en este registro (o si no se muestra este registro en la salida), no está ejecutando la versión `NDB`-activada de `mysqld`. Si ve `DISABLED` en este registro, entonces su servidor es capaz de usar el motor `NDBCLUSTER` , pero debe activarse mediante algunos de los métodos ya descritos.

Para leer los datos de configuración del cluster, el MySQL server necesita como mínimo 3 informaciones:

- El ID del nodo del servidor MySQL.
- El nombre de equipo o dirección IP para el servidor de administración (nodo MGM).
- El puerto en el que puede conectarse al servidor de administración.

Los ID de nodo pueden registrarse dinámicamente en MySQL 5.0, así que no es estrictamente necesario especificarlo explícitamente.

El parámetro de `mysqld` `ndb-connectstring` se usa para especificar el connectstring en la línea de comandos al arrancar `mysqld` o en `my.cnf`. El connectstring contiene el nombre de equipo o direcciones IP así como el puerto donde puede encontrarse el servidor de administración.

En el siguiente ejemplo, `ndb_mgmd.mysql.com` es el equipo en que reside el servidor de administración, y el servidor de administracion escucha los mensajes del cluster en el puerto 1186:

```
shell> mysqld --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Consulte Sección 16.4.4.2, “El `connectstring` de MySQL Cluster” para más información sobre los connectstrings.

Dada esta información el servidor MySQL será un participante activo del cluster. (En ocasiones nos referimos a un proceso `mysqld` que se ejecute de este modo como nodo SQL.) Será consciente de todos los nodos de datos del cluster así como de su estado, y establecerá conexiones con todos los nodos de datos. En este caso, es capaz de usar cualquier nodo de datos como un coordinador de transacción y para acceder nodos de datos para leer y actualizar.

16.5.2. `ndbd`, el proceso del nodo de motor de almacenamiento

`ndbd` es el proceso que se usa para tratar todos los datos en las tablas usando el motor de almacenamiento NDB Cluster. Este proceso fuerza un nodo de almacenamiento a que cumpla el

tratamiento de transacciones distribuídas, recuperación de nodos, realización de checkpoints, copia de seguridad en línea, y tareas relacionadas.

En MySQL Cluster, un conjunto de procesos ndbd cooperan para tratar datos. Estos procesos pueden ejecutarse en la misma máquina (equipo) o en máquinas distintas. Las correspondencias entre nodos de datos y máquinas cluster son completamente configurables.

En MySQL 5.0, ndbd genera un conjunto de ficheros de log que se guardan en el directorio especificado por DataDir en el fichero de configuración. Estos ficheros de log se listan a continuación. Tenga en cuenta que <NodeID> representa el identificador único del nodo. Por ejemplo, ndb_2_error.log es el log de error generado por el nodo de almacenamiento cuyo ID de nodo es **2**.

- ndb_*<NodeID>*_error.log es un fichero que contiene registros de todas las caídas que ha encontrado el proceso referenciado ndbd. Cada registro en este fichero contiene un breve mensaje de error y una referencia a un fichero de traceo para este fallo. Una entrada típica en este fichero puede ser parecida a:

  ```
  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***
  ```

- ndb_*<NodeID>*_trace.log.*<TraceID>* es un fichero de traceo que describe exactamente qué ha ocurrido jusnto antes de que ocurra el error. Esta información es útil para su análisis por parte del equipo de desarrollo de MySQL Cluster .

  Es posible configurar el número de estos ficheros de traceo que se crean antes que los ficheros antiguos se sobreescriban. *<TraceID>* es un número que se incrementa para cada fichero de traceo sucesivo.

- ndb_*<NodeID>*_trace.log.next es el fichero que se encarga del siguiente número de traceo para fichero a asignar.

- ndb_*<NodeID>*_out.log es un fichero que contiene cualquier salida de datos del proceso ndbd . Este fichero se crea sólo si ndbd se arranca como demonio.

- ndb_*<NodeID>*.pid es un fichero que contiene el ID de proceso del proceso ndbd cuando se arranca como demonio. También funciona como fichero de bloqueo para evitar arrancar los nodos con el mismo identificador.

- ndb_*<NodeID>*_signal.log es un fichero usado sólo en versiones de depuración de ndbd, donde es posible tracear toda la entrada, salida, y mensajes internos con sus datos en el proceso ndbd .

Se recomiendo no usar un directorio montado mediante NFS porque en algunos entornos puede causar problemas donde el bloqueo del fichero pid sigue en efecto tras la finalización del proceso.

Al reiniciar ndbd puede ser necesario especificar un nombre de equipo del servidor de administración y el puerto en que está escuchando. (Opcionalmente, se puede especificar el ID de nodo que usará el proceso.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

Consulte Sección 16.4.4.2, “El connectstring de MySQL Cluster” para más información sobre este tema.

Cuando ndbd arranca, inicia dos procesos. El primero de ellos se llama el proceso “angel process”; su trabajo es descubrir cuándo se completa el proceso de ejecución, y luego reinicia el proceso ndbd si se configura para hacerlo. Por lo tanto, is trata de matar ndbd via comando Unix kill , es necesario matar a ambos procesos. Una forma más adecuada de matar un proceso ndbd es usar el cliente de administración y parar el proceso desde allí.

El proceso en ejecución usa un flujo para lectura, escritura y escaneo de datos, así como otras actividades. Este flujo se implementa asíncronamente para que pueda tratar fácilmente miles de actividades concurrentes. Además, un flujo watch-dog supervisa el flujo de ejecución para asegurarse que no se cuelga en un bucle infinito. Un pool de flujos trata ficheros de entrada/salida, siendo cada flujo capaz de tratar un fichero abierto. Los flujos pueden usarse por las conexiones en el proceso de transporet de ndbd . En un sistema que realice un gran número de operaciones, incluyendo actualizaciones, el proceso ndbd consume hasta 2 CPUs si se permite hacerlo. Para máquinas con varias CPUs se recomienda usar varios procesos ndbd que pertenecen a diferentes grupos de nodos.

16.5.3. El proceso del servidor de administración ndb_mgmd

El servidor de administración es el proceso que lee el fichero de configuración del cluster y distribuye esta información a todos los nodos en el cluster que la piden. También mantiene un log de las actividades del cluster. Los clientes de administración pueden conectar al servidor de administración y chequear el estado del cluster.

No es estrictamente necesario especificar un connectstring al arrancar un servidor de administración. Sin embargo, si está usando más de un servidor de administración, debe proporcionarse un connectstring y cada nodo en el cluster debe especificar su ID de nodo explícitamente.

Los siguientes ficheros se crean o usan por ndb_mgmd en su directorio de arranque. En MySQL 5.0 Cluster, los ficheros de log y PID se guardan en el DataDir como se especifica en el fichero de configuración. En la lista a continuación, *<NodeID>* es el único identificador de nodo.

- config.ini es el fichero de configuración para el cluster como un todo. Este fichero se crea por el usuario y lo lee el servidor de administración. La configuración de este fichero se discute en Sección 16.4, “Configuración de MySQL Cluster”.
- ndb_*<NodeID>*_cluster.log es el ficheor de log de eventos del cluster. Ejemplos de tales eventos incluyen inicio y finalización de checkpoints, eventos de arranque de nodos, fallos de nodos, y niveles de uso de memoria. Una lista completa de eventos de cluster con descripciones puede encontrarse en Sección 16.6, “Administración de MySQL Cluster”.

  Cuando el tamaño del log del cluster alcanza un millón de bytes el fichero se renombra a ndb_*<NodeID>*_cluster.log.*SeqID*, donde *SeqID* es el número de secuencia del fichero de log del cluster . (Por ejemplo: si 1,2, y 3 ya existen, el siguiente fichero de log se llamará con el número 4.)
- ndb_*<NodeID>*_out.log es el fichero usado por stdout y stderr cuando se ejecuta el servidor de administración como demonio.
- ndb_*<NodeID>*.pid es el fichero PID usado cuando se ejecuta el servidor de administración como demonio.

16.5.4. El proceso de cliente de administración ndb_mgm

El proceso de cliente de administración no es necesario que se ejecute en el cluster. Su validez estriba en proporcionar un conjunto de comandos para chequear el estado del cluster, arrancar copias de seguridad,

y realizar otras funciones administrativas. El cliente de administración accede al servidor de administración usando la API C que pueden emplear los usuarios avanzados para programación dedicada a procesos de administración que pueden realizar tareas similares a las realizadas por `ndb_mgm`.

Al arrancar el cliente de administración, es necesario proporcionar el nombre de equipo y puerto del servidor de administración como en el ejemplo posterior. Los valores por defecto en MySQL 5.0 son `localhost` y `1186`.

```
shell> ndb_mgm localhost 1186
```

Más información puede acerca de `ndb_mgm` puede encontrarse en Sección 16.5.5.4, “Opciones de comando para ndb_mgm” y Sección 16.6.1, “Comandos del cliente de administración”.

16.5.5. Opciones de comando para procesos de MySQL Cluster

Todos los ejecutables de MySQL Cluster (excepto `mysqld`) tienen las siguientes opciones. Los usuarios de versiones anteriores de MySQL Cluster deben tener en cuenta que algunas de estas opciones han cambiado desde MySQL 4.1 Cluster para hacerlos consistentes entre ellos así como con `mysqld`. Puede usar la opción `-?` para ver un listado de las opciones soportadas.

- `-?, --usage, --help`

  Muestra una pequeña lista con descripciones de las opciones de comandos disponibles.

- `-V, --version`

  Muestra el número de versión del proceso `ndbd`. El número de versión es el número de MySQL Cluster. El número de versión es relevante porque no todas las versiones pueden usarse juntas, y al arrancar el proceso MySQL Cluster verifica que las versoines de los binarios usados pueden coexistir en el mismo cluster. Esto es importante al realizar una actualización de software en línea de MySQL Cluster (consulte `Software Upgrade of MySQL Cluster`).

- `-c` *`connect_string`*`, --connect-string`

  `connect_string` Cambia la cadena de conexión del servidor de administración como opción de comando.

  ```
  shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
  ```

- `--debug[=`*`options`*`]`

  Esta opción sólo puede usarse para opciones compiladas con depuración activada. Se usa para permitir la salida de las llamadas de depuración de la misma forma que para el proceso `mysqld` .

- `-e`, `--execute`

  Puede usarse para enviar un comando al ejecutable cluster desde el shell de sistema. Por ejemplo, cualquiera de los siguientes comandos:

  ```
  shell> ndb_mgm -e show
  ```

  o

  ```
  shell> ndb_mgm --execute="SHOW"
  ```

  es equivalente a

```
NDB> SHOW;
```

Esto es análogo a cómo la opción `-e` funciona con el cliente de línea de comandos `mysql` . Consulte Sección 4.3.1, “Usar opciones en la línea de comandos”.

16.5.5.1. Opciones de `mysqld` relacionadas con MySQL Cluster

- `--ndbcluster`

  Si el binario incluye soporte para el motor `NDB Cluster` puede evitarse la desactivación por defecto del motor `NDB` mediante el uso de esta opción. El uso del motor `NDB Cluster` es necesario para MySQL Cluster.

- `--skip-ndbcluster`

  Desactiva el motor `NDB Cluster`. Esta opción está activa por defecto para binarios donde se incluye; en otras palabras, el motor `NDB Cluster` se desactiva hasta que se activa via `--ndbcluster`. Esta opción se activa sólo si el servidor se compiló con soporte para el motor `NDB Cluster` .

- `--ndb-connectstring=`*`connect_string`*

  Al usar el motor `NDB`, es posible mediante esta opción especificar el servidor de almacenamiento que distribuye los datos de configuración del cluster.

16.5.5.2. Opciones de comando para `ndbd`

Para opciones comunes consulte Sección 16.5.5, “Opciones de comando para procesos de MySQL Cluster”.

- `-d`, `--daemon`

  Le dice a `ndbd` que se ejecute como un proceso demonio. En MySQL 5.0, es el comportamiento por defecto.

- `--nodaemon`

  Le dice a `ndbd` que no arranque como proceso demonio. Es útil cuando `ndbd` se depura y se quiere que la salida se redirija a la pantalla.

- `--initial`

  Le dice a `ndbd` que realice un escaneo inicial. Un arranque inicial borra cualquier fichero creado con motivos de recuperación de instancias prévias de `ndbd`. También recrea ficheros log de recuperación. Tenga en cuenta que en algunos sistemas operativos este proceso puede llevar una cantidad de tiempo substancial.

  Un arranque `--initial` se usa sólo la primera vez que se arranca el proceso `ndbd` , y borra todos los ficheros del sistema de ficheros del cluster y recrea todos los ficheros de log REDO. Las excepciones a esta regla son:

  - Cuando se realiza una actualización de software que cambia los contenidos de cualquier fichero.
  - Al reiniciar el nodo con una nueva versión de `ndbd`.
  - Como medida de última hora cuando por alguna razón el reinicio repetitivo del nodo o sistema fallan. En este caso el nodo no puede seguir usándose para restaurar datos debido a la destrucción de los ficheros de datos.

Esta opción no afecta ningún fichero de copia de seguridad que se ha creado por el nodo afectado.

- `--nostart`

  Le dice a `ndbd` que no arranque automáticamente. Cuando se usa esta opción, `ndbd` conecta el servidor de administración , obtiene datos de configuración del mismo, e inicia los objetos de comunicación. Sin embargo, no arranca realmente el motor hasta que se le pida que lo haga explícitamente por parte del servidor de administración. Esto puede realizarse ejecutando el comando apropiado por parte del cliente de administración.

16.5.5.3. Opciones del comando `ndb_mgmd`

Para algunas opciones comunes consulte Sección 16.5.5, “Opciones de comando para procesos de MySQL Cluster”.

- `-f filename, --config-file=filename`, (**OBSOLETE**): `-c filename`

  Le dice al servidor de administración qué fichero debe usar como fichero de configuración. Esta opción debe especificarse. El fichero por defecto es `config.ini`. **Nota**: La abreviación `-c` es obsoleta en MySQL 5.0 Cluster y no debe usarse en nuevas instalaciones.

- `-d`, `--daemon`

  Le dice a `ndb_mgmd` que arranque como demonio. Este es el comportamiento por defecto.

- `-nodaemon`

  Le dice al servidor de administración que no arranque como demonio.

16.5.5.4. Opciones de comando para `ndb_mgm`

Para algunas opciones comunes consulte Sección 16.5.5, “Opciones de comando para procesos de MySQL Cluster”.

- [*host_name* [*port_num*]]

  Para arrancar el cliente de administración es necesario especificar dónde reside el servidor de administración, que significa especificar el nombre de equipo y puerto. El equipo por defecto es `localhost`; el puerto por defecto es 1186.

- `--try-reconnect=number`

  Si la conexión al servidor de administración se rompe, entonces el nodo trata de reconectar al mismo cada 5 segundos hasta que tiene éxito. Usando esta opción, es posible limitar el número de reintentos con *number* antes de abandonar y reportar un error en su lugar.

16.6. Administración de MySQL Cluster

Administar un MySQL Cluster implica un número de tareas, la primera de ellas es configurar y arrancar el MySQL Cluster. Esto se cubre en Sección 16.4, “Configuración de MySQL Cluster” y Sección 16.5, “Gestión de procesos en MySQL Cluster”.

Las siguientes secciones cubren la administración de un MySQL Cluster en ejecución.

Esencialmente hay dos métodos de administrar activamente un MySQL Cluster en ejecución. El primero de ellos es mediante el uso de comandos entrados en el cliente de administración donde

puede chequearse el estado del cluster, cambiar los niveles de log, arrancar y parar copias de seguridad y parar y arrancar nodos. El segundo método implica estudiar el contenido del log del cluster `ndb_<NodeID>_cluster.log` en el directorio del servidor de adminisración `DataDir` . (Recuerde que `<NodeID>` representa el identificador único del nodo cuya actividad se está logueando.) El log del cluster contiene reportes de eventos generados por `ndbd`. Es posible también enviar entradas del log del cluster a un log de sistema Unix.

16.6.1. Comandos del cliente de administración

Además del fichero de configuración central, un cluster puede ser controlado mediante una interfaz de línea de comandos disponible a través del cliente de administración `ndb_mgm`. Esta es la interfaz administrativa primaria para un cluster en ejecución.

El cliente de administración tiene los siguientes comandos básicos. En la lista que sigue, `<id>` dentoa un ID de nodo de base de datos o la palabra clave `ALL`, que indica que el comando debe aplicarse a todos los nodos de datos en el cluster.

- `HELP`

  Muestra información de todos los comandos disponibles.

- `SHOW`

  Muestra informaicón del estado del cluster.

- `<id> START`

  Arranca el nodo de datos identificado por `<id>` (o todos los nodos de datos).

- `<id> STOP`

  Para el nodo de datos identificado por `<id>` (o todos los nodos de datos).

- `<id> RESTART [-N] [-I]`

  Reinicia el nodo de datos identificado por `<id>` (o todos los nodos de datos).

- `<id> STATUS`

  Muestra información de estado para el nodo de datos identificado por `<id>` (o todos los nodos de datos).

- `ENTER SINGLE USER MODE <id>`

  Entra en modo de usuario único, donde sólo el servidor MySQL identificado por el ID del nodo `<id>` tiene permiso para acceder a la base de datos.

- `EXIT SINGLE USER MODE`

  Abandona el modo de usuario único permitiendo a todos los nodos SQL (esto es, todos los procesos `mysqld` en ejecución) para acceder a la base de datos.

- `QUIT`

  Termina el cliente de administración.

- `SHUTDOWN`

Para todos los nodos del cluster, excepto los nodos SQL, y sale.

Los comandos para los logs de eventos se dan en la siguiente sección; los comandos para creación de copias de seguridad y restaurar de una copia de seguridad se proporcionan en una sección separada de estos temas.

16.6.2. Informes de eventos generados por MySQL Cluster

En esta sección dicutimos los tipos de logs de eventos proporcionados por MySQL Cluster, y los tipos de eventos que se loguean.

MySQL Cluster proporciona dos tipos de log de evento. Son **cluster log**, que incluye eventos generados por todos los nodos del cluster, y **node logs**, que son locales en cada nodo de datos.

La salida generada por el logueo de eventos del cluster puede tener varias destinaciones incluyendo un fichero, la consola del servidor de administración o `syslog`. La salida generada por el logueo de eventos de un nodo se escribe en la ventana de consola de datos del nodo.

Ambos tipos de logueo de eventos pueden configurarse para loguear distintos subconjuntos de eventos.

Nota: El log del cluster es el log recomendado para la mayoría de usos, ya que proporciona información de logueo para un cluster entero en un único fichero . Los logs de nodos están pensados para ser usados sólo durante desarrollo de aplicaciones, o para depurar código de aplicación.

Cada evento reportable puede distinguirse según tres criterios distintos:

- *Categoría*: Puede tener uno de los siguientes valores: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, o `INFO`.
- *Prioridad*: Representada por un número de 1 a 15 incluidos, donde 1 indica “más importante” y 15 “menos importante”.
- *Nivel de severidad*: Puede ser uno de los siguientes valores: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, o `DEBUG`.

El log de cluster y el log de nodo pueden filtrarse con estas propiedades.

16.6.2.1. Registrar comandos de control

Los siguientes comandos de administación están relacionados con el log de cluster:

- `CLUSTERLOG ON`

  Activa el log del cluster.

- `CLUSTERLOG OFF`

  Desactiva el log del cluster

- `CLUSTERLOG INFO`

  Información acerca de la configuración del log del cluster.

- `<id> CLUSTERLOG <category>=<threshold>`

  Loguea eventos de *`category`* con prioridad menor o igual a *`threshold`* en el log del cluster.

- `CLUSTERLOG FILTER <severity>`

  Cambia a loguear eventos del cluster de la *`severity`* especificada.

La siguiente tabla describe la configuración por defecto (para todos los nodos de datos) del umbral de categoría del log del cluster. Si un evento tiene una prioridad con un valor menor o igual al umbral de prioridad, se reporta en el log del cluster.

Tenga en cuenta que los eventos se reportan por nodo de datos, y que el umbral puede configurarse con distintos valores en nodos diferentes

Categoría	Umbral por defecto (todos los nodos de datos)
`STARTUP`	**7**
`SHUTDOWN`	**7**
`STATISTICS`	**7**
`CHECKPOINT`	**7**
`NODERESTART`	**7**
`CONNECTION`	**7**
`ERROR`	**15**
`INFO`	**7**

Los umbrales se usan para filtrar eventos dentro de cada categoría. Por ejemplo, un evento `STARTUP` con una prioridad de 3 no se loguea a no ser que el umbral para `STARTUP` se cambie a 3 o menos. Sólo los eventos con prioridad de 3 o menor se envían si el umbral es 3.

Los niveles de seguridad de los eventos se muestran a continuación. (**Nota**: Se corresponde a niveles `syslog` Unix, excepto para `LOG_EMERG` y `LOG_NOTICE`, que no se usan o mapean.)

1	`ALERT`	Condición que debe corregirse inmediatamente, tal como una base de datos de sistema corrupta
2	`CRITICAL`	Condiciones críticas, tales como errores de dispositivos o recursos insuficientes
3	`ERROR`	Condiciones que deben corregirse, tales como errores de configuración
4	`WARNING`	Condiciones que no son errores, pero que pueden requerir tratamiento especial
5	`INFO`	Mensajes de información
6	`DEBUG`	Mensajes de depuración usados para desarrollo `NDB Cluster`

Los niveles de severidad de eventos pueden activarse o desactivarse. Si un invel de severidad se activa, todos los eventos con una prioridad menor o igual que los umbrales de categoría se loguan. Si el nivel de severidad se deactiva no se loguan los eventos pertenecientes a ese nivel.

16.6.2.2. Registro de eventos

Un reporte de evento se reporta en el log de eventos con el siguiente formato: *`<datetime>`* `[`*`<string>`*`]` *`<severity>`* `--` *`<message>`*. Por ejemplo:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

Todos los eventos reportables se discuten en esta sección, ordenados por categoría y nivel de severidad dentro de cada categoría.

`CONNECTION` Eventos

Hay eventos asociados con conexiones entre nodos del cluster.

Evento	Prioridad	Nivel de severidad	Descripción
nodos DB conectados	8	INFO	nodos de datos conectados
nodos DB desconectados	8	INFO	nodos de datos desconectados
comunicación cerrada	8	INFO	conexión con nodo SQL o de datos node cerrada
Comunicación abierta	8	INFO	conexión con nodo SQL o de datos node abierta

`CHECKPOINT` Eventos

Los mensajes de log que se muestran se asocian con checkpoints.

(**Nota**: GCP = Global Checkpoint, LCP = Local Checkpoint.)

Evento	Prioridad	Nivel de severidad	Descripción
LCP parado en el cálculo de mantenimiento de GCI	0	ALERT	LCP parado
Fragmento de checkpoint local completo	11	INFO	LCP en un fragmento se ha completado
Checkpoint global completo	10	INFO	GCP finalizado
Checkpoint global arrancado	9	INFO	Arranque de GCP: REDO log escrito en disco
Checkpoint local completado	8	INFO	LCP completado normalmente
Checkpoint local arrancado	7	INFO	Arranque de LCP: datos escritos en disco
Reporte del log de undo bloqueado	7	INFO	Logueo de UNDO bloqueado; buffer cerca de desbordarse

`STARTUP` Eventos

Los siguientes eventos se generan en respuesta al arranque de un nodo o del cluster y de su éxito o fallida. También proporciona información relacionada con el progreso del proceso de arranque, incluyendo información acerca de las actividades de logueo.

Evento	Prioridad	Nivel de severidad	Descripción
Señal de arranque interna recibida STTORRY	15	INFO	Bloques recibidos tras completar el reinicio

Registros Undo rejecutados	15	INFO	
Arrancado nuevo log REDO	10	INFO	GCI guarda *X*, los GCI restaurables más nuevos *Y*
Nuevo log arrancado	10	INFO	Parte *X* de log, arranca MB *Y*, para MB *Z*
El nodo se ha rehusado para incluir en el cluster	8	INFO	El nodo no puede incluirse en el cluster debido a fallos de configuración, inabilidad de establecer comunicación, u otros problemas
DB nodos vecinos	8	INFO	Muestra nodos de datos vecino
DB nodo arranca fase *X* completada	4	INFO	Un inicio de fase de nodo se ha completado
El nodo se ha incluido con éxito en el cluster	3	INFO	Muestra el nodo, nodo de administración, e ID dinámico
Fases de arranque de nodo DB iniciadas	1	INFO	Nodos de cluster NDB arrancando
Todas las fases de arranque del nodo DB completadas	1	INFO	Nodos NDB Cluster arrancados
Parada de nodo DB iniciada	1	INFO	Parada de nodos de datos comenzada
Parada de nodo DB abortada	1	INFO	Incapaz de cerrar el nodo de datos normalmente

`NODERESTART` Eventos

Los siguientes eventos se generan al reiniciar un nodo y están relacionados con el éxito o fracaso del proceso de reinicio del nodo.

Evento	Prioridad	Nivel de severidad	Descripción
Fase de fallo de nodo completa	8	ALERT	Rerporta la finalización de las fases de fallo de nodo
El nodo ha fallado, el estado del nodo era *X*	8	ALERT	Reporta que el nodo ha fallado
Reporta el resultado del árbitro	2	ALERT	Hay 8 resultados distintos posibles de arbitraciones: • Chequeo de arbitración fallido - quedan menos de 1/2 de los nodos. • Chequeo de arbitración exitoso - mayoría de nodos del grupo. • Chequeo de arbitración fallido - falta un grupo de nodos. • Partición de red - arbitración necesaria • Arbitración exitosa - respuesta afirmativa del nodo *X* • Arbitración fallida - respuesta negativa del nodo *X*

			• Partición de red - no hay árbitro disponible • Partición de red - no hay árbitro configurado
Completada la copia de fragmento	10	INFO	
Completada la copia de información de directorio	8	INFO	
Completada la copia de información de distribución	8	INFO	
Iniciada la copia de fragmentos	8	INFO	
Completada la copia de todos los fragmentos	8	INFO	
Control de GCP iniciado	7	INFO	
Control de GCP completado	7	INFO	
Control de LCP iniciado	7	INFO	
Control de LCP completado (estado = *X*)	7	INFO	
Reporta si se encuentra un árbitro o no	6	INFO	Hay 7 salidas distintas al buscar un árbitro: • El servidor de administración reinicia el flujo árbitro [estado=*X*] • Prepara nodo árbitro *X* [ticket=*Y*] • Recibe nodo árbitro *X* [ticket=*Y*] • Nodo árbitro iniciado *X* [ticket=*Y*] • Nodo árbitro perdido *X* - proceso fallido [estado=*Y*] • Lost arbitrator node *X* - process exit [state=*Y*] • Nodo árbitro perido *X* <error msg> [estado=*Y*]

`STATISTICS` Eventos

Los siguientes eventos son de naturaleza estadística. Proporcionan información tal como números de transacción y otras operaciones, cantidad de datos enviados o recibidos por nodos individuales, y uso de memoria.

Evento	Prioridad	Nivel de severidad	Descripción
Reporta estadísticas de planificación de trabajos	9	INFO	Estadísticas de planificación de trabajos internos
Número de bytes enviados	9	INFO	Número de bytes enviados al nodo*X*
Recibidos # bytes	9	INFO	Número de bytes recibidos del nodo *X*
Reporte de estadísticas de transacción	8	INFO	Número de: transacciones, commits, lecturas, lecturas simples, escrituras, operaciones

			concurrentes, información de atributos, y abortos
Operaciones de reporte	8	INFO	Número de operaciones
Creación de tabla de reportes	7	INFO	
Uso de memoria	5	INFO	Uso de datos e índices (80%, 90% y 100%)

`ERROR` Eventos

Estos eventos están relacionados con errores y advertencias del cluster; la presencia de uno o más de ellos indica generalmente que un fallo importante ha ocurrido.

Evento	Prioridad	Severidad	Descripción
Muerte debida a un heartbeat fallido	8	ALERT	Nodo *X* declarado “dead” debido a un heartbeat fallido
Errores de transporte	2	ERROR	
Advertencias de transporte	8	WARNING	
Heartbeats perdidos	8	WARNING	Nodo *X* ha perdido el heartbeat #*Y*
Eventos de advertencia generales	2	WARNING	

`INFO` Eventos

Estos eventos proporcionan información general sobre el estado del cluster y actividades asociadas con el mantenimiento del mismo, tales como logueo y transmisión de heartbeat .

Evento	Prioridad	Severidad	Descripción
Heartbeat enviado	12	INFO	Heartbeat enviado a nodo *X*
Bytes de log creado	11	INFO	Parte de log, fichero de log, MB
Eventos de información general	2	INFO	

16.6.3. Modo de usuario único

El modo de usuario único permite al administrador de la base de datos restringir el acceso al sistema de base de datos a un único servidor MySQL (nodo SQL): Al entrar en el modo de usuario único todas las conexiones a todos los otros servidores MySQL se cierran y todas las transacciones en ejecución se abortan. No se permiten nuevas transacciones.

Una vez que el cluster ha entrado en modo de usuario único, sólo el nodo SQL designado tiene acceso a la base de datos. Puede usar el comando `all status` para ver si el cluster ha entrado en este modo.

Ejemplo:

```
NDB> ENTER SINGLE USER MODE 5
```

Tras la ejecución de este comando y que el cluster entre en modo de usuario único, el nodo SQL cuyo ID de nodo es `5` pasa a ser el único usuario permitido del cluster.

El nodo especificado en el comando superior debe ser un nodo MySQL Server ; Un intento de especificar cualquier otro tipo de nodo se rehusará.

Nota: Cuando se invoca el comando anteior, todas las transacciones en ejecución en el nodo designado se abortan, la conexión se cierra y el servidor debe reiniciarse.

El comando `EXIT SINGLE USER MODE` cambia el estado de los nodos del datos del cluster de modo de usuario único a modo normal. Los servidores MySQL esperando conexiones (esto es, a que el cluster pase a estar preparado y disponible), pueden conectarse: El servidor MySQL denotado como el nodo de usuario único SQL continúa ejecutándose (si todavía está conectado) durante y tras el cambio de estado.

Ejemplo:

```
NDB> EXIT SINGLE USER MODE
```

La forma recomendada de tratar un fallo de nodo al ejecutar el modo de usuario único es una de las siguientes:

- 1. Terminar todas las transacciones de modo de usuario único.
  2. Ejecutar el comando `EXIT SINGLE USER MODE`
  3. Reiniciar los nodos de datos del cluster

or

- Reinicia los nodos de base de datos antes de entrar en modo de usuario único.

16.6.4. Copias de seguridad On-line para MySQL Cluster

Esta sección describe cómo crear una copia de seguridad y cómo restaurar la base de datos de una copia de seguridad posteriormente.

16.6.4.1. Conceptos de copias de seguridad de nodos

Una copia de seguridad es una fotografía de la base de datos en un momento dado. La copia de seguridad tiene tres partes principales:

- **Metadatos**: nombres y definiciones de todas las tablas de la base de datos.
- **Registros de tabla**: los datos guardados realmente en las tablas de la base de datos cuando se hizo la copia de seguridad
- **Log de transacción**: un registro secuencial diciendo cómo y cuándo se han almacenado los datos en la base de datos.

Cada uno de ellos se guarda en todos los nodos participantes en la copia de seguridad. Durante la copia de seguridad cada nodo guarda estas tres partes en tres ficheros en disco:

- `BACKUP-<BackupId>.<NodeId>.ctl`

  Un fichero de control con información de control y metadatos. Cada nodo guarda las mismas definiciones de tabla (para todas las tablas en el cluster) en su propia versión de este fichero.

- `BACKUP-<BackupId>-0.<NodeId>.data`

  Un fichero de datos conteniendo los registros de tabla, que se guardan en fragmentos; esto es, distintos nodos guardan distintos fragmentos durante la copia de seguridad. El fichero guardado por cada nodo

comienza con una cabecera que explica a qué tabla pertenencen los registros. A continuación de la lista de registros, hay un pie que contiene un checksum de los registros.

- `BACKUP-<BackupId>.<NodeId>.log`

  Un fichero de log con registros de transacciones finalizadas. Sólo las transacciones de tablas guardadas en la copia de seguridad se guardan en el log. Los nodos involucrados en la copia de seguridad guardan distintos registros, ya que distintos nodos guardan distintos fragmentos de base de datos.

En la tabla anterior `<BackupId>` es el identificador de la copia de seguridad y `<NodeId>` es el identificador único para el nodo que crea el fichero.

16.6.4.2. Usar el servidor de administración para crear una copia de seguridad

Antes de iniciar una copia de seguridad, asegúrese que el cluster está debidamente configurado para ello. (consulte Sección 16.6.4.4, “Configuración para copias de seguridad de un nodo”.)

Para crear una copia de seguridad usando el servidor de administración implica los siguientes pasos:

1. Arrancar el servidor de administración (`ndb_mgm`).
2. Ejecutar el comando `START BACKUP`.
3. El servidor de administración responderá con el mensaje “`Start of backup ordered`”. Esto significa que el servidor de administración ha enviado la petición al cluster, pero no ha recibido respuesta aun.
4. El servidor de administración responderá “`Backup <BackupId> started`”, donde `<BackupId>` es el identificador único para esta copia de seguridad. (Este ID se guardará en el log del cluster, si no ha sido configurado de otro modo.) Esto significa sólo que el cluster ha recibido y procesado la petición de copia de seguridad. Esto *no* significa que se haya completado.
5. El servidor de administración envía la señal que la copia ha finalizado con el mensaje “`Backup <BackupId> completed`”.

Es posible iniciar la copia de seguridad de la shell del sistema usando

```
shell> ndb_mgm -e "START BACKUP"
```

Para abortar una copia en marcha:

1. Arranque el servidor de administración.
2. Ejecute el comando '`ABORT BACKUP <BackupId>`'. El número `<BackupId>` es el identificador de la copia que se incluyó en la respuesta del servidor de administración cuando se inició la copia (en el mensaje '`"Backup <BackupId> started"`').
3. El servidor de administración reconocerá la petición de aborto con “`Abort of backup <BackupId> ordered`”; tenga en cuenta que no se ha recibido todavía respuesta a esta petición.
4. Una vez que la copia de seguridad se ha abortado, el servidor de administración reportará “`Backup <BackupId> has been aborted for reason msg`”. Esto significa que el cluster ha terminado la copia y que todos los ficheros relacionados con la misma se han eliminado del sistema de ficheros del cluster.

Es posible abortar una copia de seguridad en curso desde el shell de sistema usando este comando:

```
shell> ndb_mgm -e "ABORT BACKUP BackupId"
```

Nota: Si no hay una copia con ID *<BackupId>* en ejecución cuando se aborta, el servidor de administración no hace ninguna respuesta explícita. Sin embargo, el hecho que se envió un comando de aborto incorrecto se indica en el log del cluster.

16.6.4.3. Cómo restablecer una copia de seguridad de un nodo

El programa de restauración del cluster se implementa como utilidad de línea de comandos separada `ndb_restore`, que lee los ficheros creados por la copia de seguridad e inserta la información almacenada en la base de datos. El programa de restauración debe ejecutarse una vez para cada conjunto de ficheros de la copia, esto es, tantas veces como nodos había cuando se creo la copia.

La primera vez que ejecute el programa de restauración, necesita restaurar los metadatos; en otras palabras, debe recrear las tablas de la base de datos. (Tenga en cuenta que el cluster debe tener una base de datos vacía cuando empieza a restaurar una copia de seguridad.) El program de restauración actúa como una API contra el cluster y por lo tanto necesita una conexión libre para conectar al cluster. Esto puede verificarse con el comando `ndb_mgm SHOW` (o desde una shell de sistema usando `ndb_mgm -e SHOW`). La opción `-c <connectstring>` puede usarse para localizar el nodo MGM (consulte Sección 16.4.4.2, “El `connectstring` de MySQL Cluster” para información acerca de connectstrings). Los ficheros de la copia de seguridad deben estar presentes en el directorio dado como argumento al programa de restauración.

Es posible restaurar una copia de seguridad a una base de datos con una configuración distinta a la que tenía en el momento de su creación. Por ejemplo, suponga que una copia de seguridad con ID `12`,se creó en un cluster con dos nodos de base de datos con los IDs de nodo `2` y `3`, se restaura en un cluster con cuatro nodos. Entonces `ndb_restore` debe ejecutarse dos veces — una por cada nodo de base de datos en el cluster donde se hizo la copia de seguridad.

Nota: Para una restauración rápida, los datos pueden restaurarse en paralelo, dado que haya una cantidad de conexiones de cluster disponibles. Sin embargo, los ficheros de datos deben siempre aplicarse antes que los logs.

16.6.4.4. Configuración para copias de seguridad de un nodo

Hay cuatro parámetros de configuración esenciales para una copia de seguridad:

- `BackupDataBufferSize`

  Cantidad de memoria usada para buffer de los datos antes de escribir en disco.

- `BackupLogBufferSize`

  Cantidad de memoria usada para buffer de registros de lob antes de escribir en disco.

- `BackupMemory`

  Memoria total reservada en un nodo de base de datos para copia de seguridad. Debe ser la suma de memoria reservada para el buffer de datos y el de log.

- `BackupWriteSize`

  Tamaño de los blocks escritos en disco. Esto se aplica tanto para el buffer de datos como para el de log.

Puede encontrar información más detallada sobre estos parámetros en Sección 16.4, “Configuración de MySQL Cluster”.

16.6.4.5. Resolver problemas con copias de seguridad

Si se retorna un código de error al hacer una petición de copia de seguridad la causa más probable es memoria insuficiente o espacio de disco insuficiente. Debe chequear que hay suficiente memoria reservada para la copia de seguridad. También que hay espacio suficiente en la partición del disco duro.

En MySQL 5.0, `NDB` no soporta lecturas repetibles, que puede causar problemas en el proceso de restauración. Mientras que el proceso de copia de seguridad es“hot”, restaurar un MySQL Cluster desde una copia de seguridad no es un proceso 100% “hot” . Esto se debe a que, por la duración del proceso de restauración, las transacciones en ejecución obtienen lecturas no repetibles de los datos restaurados. Esto significa que el estado de los datos es incosistente mientras que la restauración está en marcha.

16.7. Usar interconexiones de alta velocidad con MySQL Cluster

Antes que comenzara el diseño de `NDB Cluster` en 1996, era evidente que uno de los mayores problemas que encontraríamos al construir bases de datos paralelas sería la comunicación entre los nodos de la red. Por esta razón, `NDB Cluster` se diseñó desde el comienzo para permitir el uso de un número de mecanismos de transporte distintos. En este manual, usamos el término *transporter* para ello.

Actualmente el código MySQL Cluster incluye soporte para 4 transportes distintos. La mayoría de usuarios usan TCP/IP sobre Ethernet ya que es ubicuo. Este también es el transporter más probado en MySQL Cluster.

Estamos trabajando para asegurar que la comunicación con el proceso `ndbd` se hace en “chucnks” tan grandes como sea posible ya que esto beneficia a todos los tipos de transimisión de datos.

Para los usuarios que lo deseen, también es posible usar interconexión de cluster para mejorar el rendimiento. Hay dos formas de hacerlo: puede diseñar un transporter a medida o puede usar implementaciones de socket que sobrepasen la pila TCP/IP de una forma u otra. Hemos expermientando con ambas técnicas usando la tecnología SCI (Scalable Coherent Interface) desarrollada por Dolphin.

16.7.1. Configurar MySQL Cluster para que utilice Sockets SCI

En esta secció mostraremos cómo se puede adaptar un cluster configurado para comunicación TCP/IP normal para que use SCI Sockets. Esta documentación se basa en SCI Sockets versión 2.3.0 de 01 Octubre 2004.

Prerequisitos

Cualquier máquina que quiera que use SCI Sockets necesita disponer de tarjetas SCI .

Es posible usar SCI Sockets con cualquier versión de MySQL Cluster. No se necesita ninguna compilación especial ya que usa llamadas de socket normales que están disponibles en MySQL Cluster. Sin embargo, SCI Sockets se soportan sólo en los kernels Linux 2.4 y 2.6 . SCI Transporters se ha probado con éxito en sistemas operativos adicionales aunque sólo lo hemos verificado con Linux 2.4 hasta ahora.

Esencialmente hay cuatro requerimientos para SCI Sockets:

1. Compilar las bibliotecas SCI Socket.
2. Instlara las bibliotecas del kernel SCI Socket.
3. Instalación de uno o dos ficheros de configuración.
4. La biblioteca de kernel SCI Socket debe estar activada para toda la máquina o para la shell en que se arranca el proceso MySQL Cluster.

Este proceso debe repetirse para cada máquina del cluster en que quiera usar SCI Sockets para comunicación entre nodos.

Necesita dos paquetes para que funcione SCI Sockets :

1. El paquete de código fuente con las bibliotecas de soporte DIS para las bibliotecas SCI Sockets .

2. El paquete de código fuente para las bibliotecas SCI Socket mismas.

Actualmente, estan disponible sólo en formato de código fuente. Las últimas versiones de estos paquetes al escribir esto están disponibles como (respectivamente) `DIS_GPL_2_5_0_SEP_10_2004.tar.gz` y `SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz`. Debería poder encontrar estas versiones (o posteriores) en http://www.dolphinics.no/support/downloads.html.

Instalación de paquetes

Una vez que tiene los paquetes con las bibliotecas, el siguiente paso es descomprimirlas en los directorios apropiados, con la biblioteca SCI Sockets en un directorio bajo el código DIS. Luego, necesita compilar las bibliotecas. Este ejemplo muestra los comandos usados en Linux/x86 para realizar esta tarea:

```
shell> tar xzf DIS_GPL_2_5_0_SEP_10_2004.tar.gz
shell> cd DIS_GPL_2_5_0_SEP_10_2004/src/
shell> tar xzf ../../SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz
shell> cd ../adm/bin/Linux_pkgs
shell> ./make_PSB_66_release
```

Es posible compilar estas bibliotecas para algunos procesadores 64-bit. Para compilar las bibliotecas para Opteron CPUs usando la extensión 64-bit , ejecute `make_PSB_66_X86_64_release` en lugar a `make_PSB_66_release`; si la compilación se hace en una máquina Itanium, debe usar `make_PSB_66_IA64_release`. La variante X86-64 debe funcionar para arquitecturas Intel EM64T pero no se ha probado.

Una vez que se completa el proceso de compilación ,las bibliotecas compiladas se encuentran en un fichero tar zipeado con un nombre entre las líneas de `DIS-<operating-system>-time-date`. Es hora de instalar el paquete en el lugar apropiado. En este ejemplo lo guardaremos en `/opt/DIS`. (**Nota**: Seguramente necesitará ejecutar lo siguiente como root del sistema.)

```
shell> cp DIS_Linux_2.4.20-8_181004.tar.gz /opt/
shell> cd /opt
shell> tar xzf DIS_Linux_2.4.20-8_181004.tar.gz
shell> mv DIS_Linux_2.4.20-8_181004 DIS
```

Configuración de red

Ahora que todos los binarios y bibliotecas están en su lugar adecuado, necesitamos asegurarnos que las tarjetas SCI tiene IDs de nodo adecuados en el espacio de direcciones SCI.

Es necesario decidir la estructura de red antes de seguir. Hay tres tipo de estructura de red que pueden usarse en este contexto:

- Anillo simple unidimensional
- Uno o más switches con un anilo para cada puerto de switch
- Un toro de dos o tres dimensiones.

Cada una de estas topologías tiene su propio método para proporcionar Ids de nodo. Discutimos cada una de las mismas brévemente.

Un anillo simple usa IDs de nodo que son múltipes de 4 distintos a cero: 4, 8, 12,...

La siguiente posibilidad usa swithces SCI. Un switch SCI tiene 8 puertos, cada uno de los cuales puede soportar un anillo. Es necesario asegurarse que distintos anillos usan distintos espacios de nombres. En una configuración típica, el primer puerto usa IDs de nodo por debajo de 64 (4 - 60), el siguiente puerto los siguientes 64 IDs (68 - 124) , y así, con IDs de nodo 452 - 508 siendo asignados al octavo puerto.

Las estructuras de red de toro bi o tri dimensional se tienen en cuenta donde cada nodo se localiza en cada dimensión, incrementando en 4 para cada nodo en la primera dimensión, en 63 cada nodo en la segunda dimensión, y (donde sea aplicable) en 1024 en la tercera dimensión. Consulte Dolphin's Web site para más documentación.

En nuestras pruebasa hemos usado switches, aunque las instalaciones de cluster más grandes usan toros bi o tri dimensionales. La ventaja proporcionada por switches es que, con tarjetas SCI duales y switches duales, es posible constuir con facilidad relativa una red redundante donde el tiempo de fallo medio en la red SCI es del orden de 100 microsegundos. Esto lo soporta el transporte SCI en MySQL Cluster y también está bajo desarrollo par la implementación SCI Socket .

Evitar fallos en el toro 2D/3D es posible pero requiere el envío de nuevos índices de enrutamiento a todos los nodos. Sin embargo, esto requiere sólo 100 milisegundos o así para completarse y debería ser aceptable para la mayoría de casos de alta disponibilidad.

Situar los nodos de datos del cluster adecuadamente en la arquitectura de switch hace que sea posible usar 2 switches para construir una arquitectura en que puedan interconectarse 16 máquinas y que un fallo único no pueda afectar a más de una. Con 32 máquinas y 2 switches se puede configurar el cluster de forma que un fallo único no pueda causar la pérdida de más de dos nodos; en este caso es posible saber qué par de nodos está afectado. Por lo tanto, poner los dos nodos en grupos de nodos separados hace que la instalación de MySQL Cluster sea “segura” .

Para poner el ID de nodo en una tarjeta SCI use el siguiente comando en el directorio `/opt/DIS/sbin`. En este ejemplo `-c 1` se refiere al número de la tarjeta SCI (siempre será 1 si sólo hay 1 tarjeta en la máquina) , `-a 0` se refiere al adaptador 0, y `68` es el ID del nodo:

```
shell> ./sciconfig -c 1 -a 0 -n 68
```

Si tiene varias tarjetas SCI en la misma máquina, puede determinar qué tarjeta tiene cada slot ejecutando el siguiente comando (de nuevo asumimos que el directorio de trabajo actual es `/opt/DIS/sbin`):

```
shell> ./sciconfig -c 1 -gsn
```

Esto le retorna el número de serie de la tarjeta SCI. Luego repita este procedimiento con `-c 2`, y siga con cada tarjeta en la máquina. Una vez que ha relacionado cada tarjeta con cada slot, puede poner los IDs de nodo para todas las tarjetas.

Una vez instaladas la bibliotecas y binarios necesarios, y configurado el ID de nodo SCI, el siguiente paso es configurar el mapeo de nombres de equipo (o direcciones IP) con el ID del nodo SCI. Esto se realiza en el fichero de configuración de los sockets SCI, que debería encontrarse en `/etc/sci/scisock.conf`. En este fichero, cada nombre de máquina o dirección IP se mapea con el ID de nodo SCI. Aquí hay un ejemplo sencillo de un fichero de configuración:

```
#host           #nodeId
alpha           8
beta            12
192.168.10.20   16
```

Es posible limitar la configuración para que se aplique sólo a un subconjunto de los puertos disponibles para estos equipos. Un fichero de configuración adicional `/etc/sci/scisock_opt.conf` puede usarse para esto, como se muestra aquí:

```
#-key                        -type        -values
EnablePortsByDefault                yes
EnablePort                  tcp          2200
DisablePort                 tcp          2201
EnablePortRange             tcp          2202 2219
DisablePortRange            tcp          2220 2231
```

Instalación de drivers

Con los ficheros de configuración en su lugar, se pueden instalar los drivers.

Primero los drivers de bajo nivel y luego el driver del socket SCI:

```
shell> cd DIS/sbin/
shell> ./drv-install add PSB66
shell> ./scisocket-install add
```

Si se desea, la instalación puede chequearse invocando un script que verifica que todos los nodos en el los ficheros de configuración del socket SCI son accesibles:

```
shell> cd /opt/DIS/sbin/
shell> ./status.sh
```

Si descubre un error y necesita cambiar la configuración del socket SCI, es necesario usar `ksocketconfig` para ello:

```
shell> cd /opt/DIS/util
shell> ./ksocketconfig -f
```

Testing y Setup

Para asegurar que los sockets SCI están siendo usados, puede emplear el programa de testeo `latency_bench` . Usando este componente del servidor de utilidades, los clientes pueden conectar con el servidor para testear la latencia de la conexión; determinar si el SCI está activado o no debe ser muy simple observando la latencia. (**Nota**: Antes de usar `latency_bench`, es necesario incializar la variable de entorno `LD_PRELOAD` como se muestra posteriormente.)

Para inicializar un servidor, use:

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -server
```

Para ejecutar un cliente, use `latency_bench` de nuevo, excepto que esta vez con la opción `-client` :

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -client server_hostname
```

La configuración del socket SCI debería estar completa ahora y MySQL Cluster preparado para usar SCI Sockets y SCI transporter (consulte Sección 16.4.4.10, “Conexiones de transporte SCI en MySQL Cluster”).

Arrancar el Cluster

El siguiente paso en el proceso es arrancar MySQL Cluster. Para permitir el uso de SCI Sockets es necesario inicializar la variable de entorno `LD_PRELOAD` antes de arrancar `ndbd`, `mysqld`, y `ndb_mgmd`. Esta variable debe apuntar a la biblioteca kernel para SCI Sockets.

Para arrancar `ndbd` en un bash shell, haga:

```
bash-shell> export LD_PRELOAD=/opt/DIS/lib/libkscisock.so
bash-shell> ndbd
```

En un entorno tcsh se puede hacer lo mismo con:

```
tcsh-shell> setenv LD_PRELOAD=/opt/DIS/lib/libkscisock.so
tcsh-shell> ndbd
```

Nota: MySQL Cluster sólo puede usar la variante de kernel de SCI Sockets.

16.7.2. Entender el impacto de interconexiones de nodos

El proceso `ndbd` tiene un número de constructores simples que se usan para acceder a los datos en MySQL Cluster. Hemos creado un benchmark sencillo para comprobar el rendimiento de cada uno de ellos y los efectos en el rendimiento de varias interconexiones.

Hay cuatro métodos de acceso:

- **Acceso por clave primaria**

  Este es el acceso simple a un registro a través de su clave primaria. Es el caso más sencillo en que sólo se accede a un registro a la vez, que significa que el coste total de preparar un número de mensajes TCP/IP y un número de costes para cambio de contexto se envían mediante esta simple petición.
  En el caaso en que se envían accesos a varias claves primarias en un proceso batch estos accesos comparten el coste de preparar los mensajes TPC/IP necesarios y los cambios de contexto. Si los mensajes TCP/IP son para destinaciones distintas, se debe preparar mensajes adicionales.

- **Acceso por clave única**

  Son similares a los anteriores, con la excepción que los accesos por clave única se ejecutan como lecturas en una tabla índice seguidos por su acceso por clave primaria a la tabla. Sin embargo, sólo se envía una petición desde el servidor MySQL, y la lectura de la tabla índice es tratada por `ndbd`. Estas peticiones también se benefician del uso de batch.

- **Escaneo de toda la tabla**

  Cuando no existen índices para la búsqueda en una tabla, se realiza un escaneo completo. Se envía como petición úncia al proceso `ndbd` , que divide el escaneo de tabla en un conjunto de escaneos paralelos en todos los procesos `ndbd` del proceso. En futuras versiones de MySQL Cluster, un nodo SQL será capaz de filtrar algunos de estos escaneos.

- **Escaneo de rangos usando índices ordenados**

  Cuando se usa un índice ordenado, realiza un escaneo igual que lo hace un escaneo de tabla completo, excepto que escanea sólo los registros en el rango usados por la consulta transmitidas por el MySQL server (nodo SQL).

Para chequear el rendimiento base de estos métodos de acceso hemos desarrollado un conjunto de benchmarks. Uno de ellos `testReadPerf`, testea accesos simple y en batch por clave primaria y única. Este benchmark también mide el coste de preparar los escaneos de rango ejecutando escaneos que

retornan un único registro. Hay una variante de este benchmark que usa un escaneo de rango para recibir un batch de registros.

De este modo, podemos determinar el coste de accesos de clave única y accesos de escaneo de registro simple, así como medir el impacto del medio de comunicación usdo, en métodos de acceso base.

En nuestros tests, ejecutamos los benchmarks para transporters normales usando sockets TCP/IP sockets y una configuración similar usando sockets SCI. Las figuras posteriores son pequeños accesos de 20 registros por acceso. Las diferencias entre accesos seriales y mediante batch se decrementan en un factor de 3 a 4 usando registros de 2 kB. SCI Sockets no se testearon con registros de 2 kB. Los tests se realizaron en un cluster con 2 nodos de datos en 2 máquinas de CPU dual equipadas con procesadores AMD MP1900+ .

```
Tipo de acceso:          TCP/IP sockets           SCI Socket
Acceso Serial clave primaria:   400 microsegundos         160 microsegundos
Acceso Batched clave primaria    28 microsegundos          22 microsegundos
Acceso Serial indice unico:     500 microsegundos         250 microsegundos
Acceso Batched indice unico:     70 microsegundos          36 microsegundos
Acceso Indexado igualdad:      1250 microsegundos         750 microsegundos
Índice rango:                    24 microsegundos          12 microsegundos
```

También realizamos otro conjunto de tests para chequear el rendimiento de los SCI Sockets vis-à-vis contra el SCI transporter, y ambos comparados con el TCP/IP transporter. Todos estos tests usaron acceso por clave primaria de forma serial o multi-flujo, o multi-flujo y batch.

Los tests mostraron que los sockets SCI eran 100% más rápido que TCP/IP. El transporter SCI era más rápido en la mayoría de casos comparado con los sockets SCI. Un caso notable ocurrión con muchos flujos en el programa de test, que mostró que el tramporter SCI no funcionaba muy bien usado por el proceso `mysqld`.

Nuestra conclusión global fue que, para la mayoría de benchmarks, usando sockets SCI mejoraba el rendimiento aproximadamente 100% sobre TCP/IP, excepto en raros casos cuando el rendimiento de la configuración no es un tema importante. Esto puede ocurrir cuando los filtros de escaneo toman la mayoría del tiempo de proceso o cuando los procesos batchs muy grandes de accesos por clave primaria se realizan. En tal caso, el proceso de la CPU en los procesos `ndbd` es la mayor parte del proceso.

Usar el transporter SCI en lugar de SCI Sockets sólo es de interés en comunicación entre procesos `ndbd`. Usar el SCI transporter también es sólo de interés si una CPU puede dedicarse al proceso `ndbd` ya que el SCI transporter se asegura que su proceso nunca estará en espera. Es importante asegurar que la prioridad del proceso `ndbd` está configurada de tal modo que el proceso no pierde prioridad debido a ejecutarse durante un periodo de tiempo extendido, como puede pasar por bloqueo de procesos por la CPU en Linux 2.6. Si tal configuración es posible, entonces el proceso `ndbd` se beneficiará de un 10-70% comparado con usar SCI sockets. (Las diferencias más grandes se verán al realizar actualizaciones y probablemente en escaneos paralelos también

Hay otras implementaciones de socket para clusters de máquinas, incluyendo Myrinet, Gigabit Ethernet, Infiniband y la interfaz VIA . Hemos testeado MySQL Cluster hasta ahora sólo con SCI sockets.También incluímos documentación acerca de cómo preparar SCI sockets usando TCP/IP ordinario para MySQL Cluster.

16.8. Limitaciones conocidas de MySQL Cluster

En esta sección, proporcionamos un listado de las limitaciones conocidas en versiones de MySQL Cluster en las series 4.1.x comparadas con características disponibles al usar los motores MyISAM e InnoDB. Actualmente, no hay planes para arreglarlas en próximas versiones 4.1; sin embargo, intentaremos proporcionar soluciones para éstas en MySQL 5.0 y versiones posteriores. Si chequea la categoría de

Cluster en la base de datos de bug de MySQL en http://bugs.mysql.com, encontrará bugs conocidos que (si están marcados como 4.1) intentaremos arreglar en próximas versiones de MySQL 4.1.

- **No cumplimiento de la sintaxis** (resultando en errores al ejecutar aplicaciones existentes):
  - No se soportan todos los conjuntos de caracteres y colaciones.
  - No hay índices prefijo; sólo los campos completos pueden indexarse.
  - Los índices de texto no están soportados.
  - Tipos de datos geométricos (WKT y WKB) no soportados.
- **No cumplimiento con límites/comportamiento** (puede resultar en errores al ejecutar aplicaciones existentes):
  - No hay rollback parcial de transacciones. Una clave duplicada o error similar resultará en un rollback de la transacción completa.
  - Existen un número de límites de hard que son configurables, pero la memoria disponible en el cluster delimite al límite. Consulte la lista completa de parámetros de configuración en Sección 16.4.4, “Fichero de configuración”. La mayoría de parámetros de configuración pueden actualizarse en línea. Estos límites de hard incluyen:
    - Tamaño de memoria de base de datos y de memoria índice (`DataMemory` y `IndexMemory`, respectivamente).
    - El máximo número de transacciones que pueden ejecutarse se delimita mediante el parámetro `MaxNoOfConcurrentOperations`. Tenga en cuenta que la cantidad de carga , `TRUNCATE TABLE`, y `ALTER TABLE` se tratan como casos especiales mediante ejecución de transacciones múltiples, y por lo tanto no están sujetas a esta limitación.
    - Los distintos límites relacionados con tablas e índices. Por ejemplo, el número máximo de índices ordenados por tabla lo determina `MaxNoOfOrderedIndexes`.
  - Los nombres de bases de datos, tablas y atributos no pueden ser tan largos en NDB como con otros motores. Los nombres de atributos se truncan a 31 caracteres, y si no son únicos tras truncarse da lugar a error. Los nombres de base de datos y de tabla pueden tener en total un máximo de 122 caracteres. (Esto es, la longitud máxima para un nombre de tabla en un cluster NDB es 122 caracteres menos el número de caracteres en el nombre de la base de datos de la tabla.)
  - En MySQL 4.1 y 5.0, todos los registros de tabla de Cluster son de longitud fija. Esto significa (por ejemplo) que si una tabla tiene uno o más campos `VARCHAR` conteniendo sólo valores pequeños, serán necesarios más memoria y espacio de disco al usar el motor NDB que el que sería para la misma tabla y datos usando el motor MyISAM. Estamos trabajando para rectificar este punto en MySQL 5.1.
  - El máximo número de objetos de metadatos está limitado a 1600, incluyendo tablas de base de datos, tablas de sistema, índices y BLOBs. Estamos trabajando para incrementar esto a aproximadamente 20k en MySQL 5.0.
  - El máximo número de atributos por tabla está limitado a 128.
  - El tamaño máximo permitido de registro es 8k, sin incluir datos almacenados en columnas BLOB. Esperamos incrementar esto a aproximadamente 32k en MySQL 5.1.
  - El máximo número de atributos por clave es 32.

- **Características no soportadas** (no causan errores, pero no están soportadas o forzadas):
  - El constructor de clave primaria se ignora, como en tablas MyISAM .
  - Los puntos de chequeo y rollbacks se ignoran como en MyISAM.
- **Rendimiento y temas relacionados con limitaciones** :
  - La caché de consulta está desactivada, ya que no está invalidad si ocurre una actualización en distintos servidores MySQL.
  - Hay temas de rendimiento de consulta debido a acceso secuencial al motor NDB; también es relativamente más costoso hacer varios escaneos de rango que con MyISAM o InnoDB.
  - La estadística `Records in range` no está soportada, resultando en planes de consulta no óptimos en algunos casos. Use `USE INDEX` o `FORCE INDEX` como solución.
  - Índices hash únicos creados con `USING HASH` no pueden usarse para acceder a una tabla si `NULL` se da como parte de la clave.
- **Caracterísitcas no incluídas**:
  - El único nivel de isolación soportado es `READ_COMMITTED`. (InnoDB soporta `READ_COMMITTED`, `REPEATABLE_READ`, y `SERIALIZABLE`.) Consulte Sección 16.6.4.5, “Resolver problemas con copias de seguridad” para información sobre cómo puede afectar a las copias de seguridad y restauración de bases de datos Cluster.
  - Commits no durables en disco. Los commits se replican, pero no hay garantía que los logs se vuelquen a disco en un commit.
- **Problemas relacionados con múltiples MySQL servers** (no relacionados con MyISAM o InnoDB):
  - `ALTER TABLE` no es completamente bloqueante cuando se ejecutan múltiples MySQL servers (no hay bloqueo de tabla distribuído).
  - La replicación MySQL no funcionará correctamente si las acutalizaciones se hacen en múltiples MySQL servers. Sin embargo, si el esquema de particionado de la base de datos es en nivel de aplicación, y no hay transacciones entre estas particiones, la replicación puede funcionar.
  - El autodescubrimiento de bases de datos no se soporta para múltiples MySQL servers accediendo al mismo MySQL Cluster. Sin embargo, el autodescubrimiento de tablas se soporta en estos casos. Lo que significa que si tras una base de datos llamada *`db_name`* se crea o importa usando un MySQL server, debe ejecutar un `CREATE DATABASE` *`db_name`*`;` en cada MySQL server adicional que accede al mismo MySQL Cluster. (Desde MySQL 5.0.2 puede usar `CREATE SCHEMA` *`db_name`*`;`.) Una vez hecho esto para un MySQL server dado, el servidor debería ser capaz de detectar las tablas de la base de datos sin error.
- **Temas exclusivos de MySQL Cluster** (no relacionados con MyISAM o InnoDB):
  - Todas las máquinsa usadas en el cluster deben tener la misma arquitectura; esto es, todas las máquinas con nodos deben ser o big-endian o little-endian, y no puede usar una mezcla. Por ejemplo, no puede tener un nodo de administración ejecutándose en un PPC con un nodo datos en una máquina x86 . Esta restricción no se aplica a máquinas que ejecuten sólo `mysql` u otros clientes que puedan estar accediendo a los nodos SQL del cluster.
  - No es posible hacer cambios del esquema en línea tales como los realizados usando `ALTER TABLE` o `CREATE INDEX`, ya que NDB Cluster no soporta autodescubrimiento de tales cambios. (Sin

embargo, puede importar o crear una tabla que use distintos motores, convertirlos a NDB usando `ALTER TABLE tbl_name ENGINE=NDBCLUSTER;`. En tales casos, necesitará ejecutar un comando `FLUSH TABLES` para forzar al cluster a recoger los cambios.)

- Añadir o eliminar nodos en línea no es posible (el cluster debe reiniciarse en tales casos).
- Cuando se usan múltiples servidores de administración debe dar a los nodos explícitamente los IDs en los connectstrings ya que la reserva automática de IDs de nodo no funciona entre múltiples servidores de administración.
- Usando varios servidores de administración debe tener cuidado de tener la misma configuración para todos los servidores de administración. El cluster no hace chequeos para ello.
- El número máximo de nodos de datos es 48.
- El número máximo de nodos en MySQL Cluster es 63. Este número incluye todos los MySQL Servers (nodos SQL), nodos de datos, y servidores de administración.

Este listado trata de ser completo respecto al conjunto de condiciones del principio de esta sección. Puede reportar cualquier discrepancia que encuentre a la base de datos de bugs de MySQL en http://bugs.mysql.com/. Si no planeamos arreglar el problema n MySQL 4.1, lo añadiremos a la lista anterior.

16.9. Mapa de desarrollo de MySQL Cluster

En esta sección, discutimos los cambios en la implementación de MySQL Cluster en MySQL 5.0 comparadas con MySQL 4.1. También discutiremos nuestra planificación para futuras mejoras en MySQL Cluster planeadas para MySQL 5.1.

En el pasado hemos recomendado que los usuarios de MySQL Cluster no usen la versión 5.0 de MySQL ya que MySQL Cluster en las series 5.0.x no estaba completamente testeada. A partir de MySQL 5.0.3-beta la funcionalidad de clustering de MySQL 5.0 es comparable a la de MySQL 4.1. MySQL 4.1 es la recomendada para producción por ahora; sin embargo, MySQL 5.0 es de alta calidad también, y le animamos a testear Cluster en MySQL 5.0 si piensa que está interesado en usarla cuando enter en producción a finales de 2005. Hay pocos cambios entre implementaciones NDB Cluster en MySQL 4.1 y en 5.0, así que la actualización es relativamente sencilla y rápida.

Desde que MySQL 5.0.3-beta está disponible, casi todas las nuevas características significativas desarrolladas para MySQL Cluster van al árbol de MySQL 5.1 . Proporcionamos algunas pistas acerca de lo que Cluster en MySQL 5.1 incluirá en el final de esta sección (consulte Sección 16.9.2, “Mapa de desarrollo de MySQL 5.1 para MySQL Cluster”).

16.9.1. Cambios de MySQL Cluster en MySQL 5.0

MySQL 5.0.3-beta y posterior contiene un número de nuevas características que son de interés:

- **Condiciones Push-Down**: Una consulta como

```
SELECT * FROM t1 WHERE non_indexed_attribute = 1;
```

usará un escaneo de toda la tabla y la condición se evaluará en los nodos de datos del cluster. Por lo tanto no es necesario enviar el registro a la red para evaluación. (Esto es, la transporte de función se usa, en lugar del transporte de datos.) Para este tipo de consultas debe observar un factor de incremento de velocidad de 5-10. Tenga en cuenta que esta característica actualmente está desactivada por defecto (pendiente de más testeo), pero debería funcionar en la mayoría de casos. Esta

característica puede estar activada mediante el comando `SET engine-condition-pushdown=On;` . Alternativamente, puede ejecutar `mysqld` con esta característica activada arrancando MySQL server con la nueva opción `--engine-condition-pushdown`.

Puede usar `EXPLAIN` para determinar cuándo se usan las condiciones push-down .

Un beneficio principal de este cambio es que las consultas se ejecutan en paralelo. Esto significa que las consultas contra columnas no indexadas pueden mejorar de 5 a 10 veces, *veces por el número de nodos de datos*, más rápido que préviamente, ya que CPUs múltiples pueden funcionar en la consulta en paralelo.

- **Decrementado uso de `IndexMemory`**: En MySQL 5.0, cada registro consume aproximadamente 25 bytes de memoria índice, y cada índice único usa 25 bytes por registro de memoria índice (además de alguna memoria de datos ya que se almacena en una tabla separada). Esto es porque no hay almacenamiento de la clave primaria en la memoria índice.

- **Caché de consulta activada para MySQL Cluster**: Consulte Sección 5.12, “La caché de consultas de MySQL” para información acerca de configurar el uso de la caché de consulta.

- **Nuevas optimizaciones**: Una optimización que merece atención particular es que una interfaz de lectura batch se usa ahora en algunas consultas. Por ejemplo, considere la siguiente consulta:

```
SELECT * FROM t1 WHERE primary_key IN (1,2,3,4,5,6,7,8,9,10);
```

  Esta consulta se ejecutará 2 o 3 veces más rápido que en versiones anteriores de MySQL Cluster debido al hecho que todas las 10 búsquedas de clave se envían en un batch único en lugar de uno cada vez.

- **Límite en número de objetos de metadatos** : En MySQL 4.1, cada base de datos de Cluster puede contener un máximo de 1600 objetos de metadatos, incluyendo tablas de base de datos, tablas de sistema, índices y BLOBs. En MySQL 5.0, esperamos incrementar este número a 20,320. Esperamos implementar esta mejora en MySQL 5.0.6 beta a mediados de 2005.

16.9.2. Mapa de desarrollo de MySQL 5.1 para MySQL Cluster

Aquí se habla de un reporte de estado basado en recientes commits en el árbol fuente MySQL 5.1 . Debería tenerse en cuenta que todo el desarrollo 5.1 está sujeto a cambio.

Actualmente hay 4 nuevas funcionalidades mayores en desarrollo para MySQL 5.1:

1. **Integración de MySQL Cluster con MySQL Replication**: Esto hará posible actualizar desde cualquier MySQL Server en el cluster y tener MySQL Replication tratada por uno de los MySQL Servers en el cluster y la instalación de un esclavo consistente.

2. **Soporte para registros basados en disco**: Los registros en disco se soportarán. Los campos índice incluyendo el índice hash de clave primaria debe almacenarse en RAM pero todos los otros campos pueden estar en disco.

3. **Registros de tamaño variable**: Una columna definida como `VARCHAR(255)` actualmente usa 260 bytes de almacenamiento independiente de lo que se almacena en un registro particular. En tablas MySQL 5.1 Cluster sólo la porción del campo tomada por el registro se almacena. Esto hará posible una reducción de los requerimientos de espacio para tales columnas por un factor de 5 en muchos casos.

4. **Partición definida por el usuario**: Los usuarios serán capaces de definir particiones basadas en las partes de los campos de la clave primaria. El MySQL Server será capaz de descubrir si es posible de

eliminar algunas de las particiones de la cláusula `WHERE` . La partición basada en `KEY`, `HASH`, `RANGE`, y `LIST` será posible, así como subparticiones. Esta característica debe estar disponible para otros criterios.

Además, estamos trabajando para incrementar el límite de tamaño de 8k para los registros que contienen columnas de tipos distintos a BLOB o TEXT en tablas cluster. Esto es debido al hecho que los registros están fijados en tamaño y el tamaño de página es de 32,768 bytes (menos 128 bytes para la cabecera del registro.) Esto significa actualmente que si permitimos más de 8k por registro, cualquier espacio que quede (hasta aproximandamente 14,000 bytes) quedaría vacío. En MySQL 5.1, planeamos para arreglar esta limitación para que use más de 8k en un registro dado no resulta en que se gaste lo que queda de la página.

16.10. Preguntas frecuentes sobre MySQL Cluster

- *¿Qué diferencia hay entre usar cluster y replicación?*

  En una replicación, un servidor maestro MySQL actualiza uno o más esclavos. Las transacciones hacen commit secuencialmente, y una transacción lenta puede causar que el esclavo quede desactualizado. Esto significa que si falla el maestro, es posible que el esclavo pueda no haber registrado las últimas transacciones. Si un motor transaccional como InnoDB se usa, una transacción será completa en el esclavo o no aplicada en absoluto, pero la replicación no garantiza que todos los datos en el maestro y el esclavo sean consistentes siempre. En MySQL Cluster, todos los nodos de datos con un commit hecho por algún nodo de datos se realiza un commit para todos los nodos. En caso de un fallo de un nodo de datos, todos los nodos de datos restantes quedarán en un estado consistente.

  En breve, mientras que la replicación estándar MySQL es asíncrona, MySQL Cluster es síncrono.

  Planeamos implementar replicación asíncrona para Cluster en MySQL 5.1. Esto incluye la capacidad de replicar entre dos cluster y entre un cluster MySQL y un MySQL server no cluster.

- *¿Necesito hacer alguna configuración especial de red para el Cluster? (¿Cómo se comunican las máquinas en un cluster?)*

  MySQL Cluster está pensado para usarse en un entorno de gran ancho de banda, con máquinas conectadas via TCP/IP. Su rendimiento depende directamente en la velocidad de conexión entre las máquinas del cluster. Los requerimientos de conectividad mínimo para cluster incluyen una red típica 100-megabit Ethernet o equivalente. Recomendamos usar Ethernet cuando sea posible.

  También se soporta el protocolo SCI (más rápido), pero necesita hardware especial. Consulte Sección 16.7, “Usar interconexiones de alta velocidad con MySQL Cluster” para más información acerca de SCI.

- *¿Cuántas máquinas necesito para ejecutar un cluster, y porqué?*

  Como mínimo se necesitan tres máquinas. Sin embargo, el número mínimo **recomendado** en MySQL Cluster es cuatro: una para el nodo de administración y otra para el de SQL, y dos para servir como nodos de almacenamiento. El propósito de los dos nodos de datos es proporcionar redundancia; el nodo de administración debe ejecutarse en una máquina separada para garantizar servicio de arbitración contínuo en caso que un nodo de datos falle.

- *¿Qué hacen las distintas máquinas en un cluster?*

  Un MySQL Cluster tiene organización física y lógica, con máquinas como elementos físicos. Los elementos lógicos son los **nodos**, y una máquina hospedando un nodo es un **huésped cluster**. Idealmente, habrá un nodo por huésped cluster, aunque es posible ejecutar más de un nodo en una máquina. Hay tres tipos de nodos, cada uno correspondiente a un rol específico en el cluster. Son:

1. **nodo de administración (nodo MGM)** : Proporciona servicios de administración para todo el cluster, incluyendo arranque, parada, copias de seguridad, y datos de configuración en otros nodos. El nodo de administración se implementa como la aplicación `ndb_mgmd`; el cliente de administración usado para controlar MySQL Cluster via nodo MGM es `ndb_mgm`.

2. **nodo de datos**: Almacena y replica datos. La funcionalidad de los nodos de datos la trata una instancia del proceso NDB `ndbd`.

3. **nodo SQL**: Símplemente es una instancia de MySQL Server (`mysqld`) arrancado con la opción `--ndb-cluster`.

- *¿Qué sistemas operativos pueden usar Cluster?*

  En MySQL 5.0, MySQL Cluster se soporta oficialmente en Linux, Mac OS X, y Solaris. Estamos trabajando para añadir soporte a cluster para otras plataformas, incluyendo Windows, y nuestra finalidad es eventualmente ofrecer MySQL Cluster en todas las plataformas en que se soporta MySQL .

  Puede ser posible ejecutar procesos Cluster en otros sistemas operativos. Hemos tenido reportes de usuarios que dicen que han ejecutado Cluster en FreeBSD. Sin embargo, Cluster en cualquier plataforma que no sen las 3 mencionadas aquí se considera software alfa (como mucho), no puede garantizarse el buen funcionamiento en un entorno de producción, y *no lo soporta MySQL AB*.

- *¿Cuáles son los requerimientos de hardware para ejecutar MySQL Cluster?*

  Cluster debe ejecutarse en cualquier plataforma en que los binarios de NDB estén disponibles. Naturalmente, una CPU más rápida y más memoria mejora el rendimiento, y CPUs de 64 bits serán mejores que los procesadores de 32. Debe haber suficiente memoria en las máquinas usadas por los nodos de datos para tratar cada parte de la base de datos (consulte **¿Cuánta RAM necesito?** para más información). Los nodos pueden comunicarse via TCP/IP estándar y su hardware. Para soporte SCI, se necesita hardware especial de red.

- *Como MySQL Cluster usa TCP/IP, ¿significa que puedo usarlo en Internet, con uno o más nodos en una localización remota?*

  Es importante tener presente que *la comunicación entre nodos en MySQL Cluster no es segura* ; no está cifrada ni protegida por ningún otro mecanismo. La configuración más segura para un cluster es en una red privada detrás de un firewall, sin acceso directo a ningún nodo de datos ni de administración desde fuera.

  Es muy dudoso que un cluster se muestre fiable bajo tales condiciones, ya que se diseñó e implementó con la suposición que se ejecutaría bajo condiciones que garantizaran conectividad dedicada de alta velocidad como las encontradas en configuraciones en una LAN con 100 Mbps o gigabit Ethernet (mejor la última). No testeamos ni garantizamos el rendimiento usando algo más lento que esto.

- *¿Debo usar nuevos lenguajes de programación o de consulta para usar Cluster?*

  No. Aunque se usan algunos comandos especializados para administrar y configurar el cluster, solo comandos estándar (My)SQL se necesitan para:

  - Crear, alterar y borrar tablas
  - Insertar, actualizar y borrar datos de tabla
  - Crear, cambiar y borrar índices únicos y primarios
  - Configurar y administrar nodos SQL (servidores MySQL)

- *¿Cómo puedo saber qué significan los mensajes de error o advertencias al usar Cluster?*

  Se puede hacer de dos modos:

  1. Desde el MySQL Monitor, use `SHOW ERRORS` o `SHOW WARNINGS` inmediatamente al recibir la notificación del error o advertencia. También se pueden mostrar en MySQL Query Browser.

  2. De un shell de sistema, use `perror --ndb error_code`.

- *¿Es MySQL Cluster transaccional? ¿Qué tipo de tablas se soportan en Cluster ?*

  Sí. MySQL Cluster utiliza tablas creadas con el motor `NDB` , que soporta transacciones. `NDB` es el único motor que soporta cluster.

- *¿Qué significa “NDB”?*

  Significa "**N**etwork **D**ata**b**ase".

- *¿Qué versiones de MySQL software soportan Cluster? ¿Debo compilarlo de las fuentes?*

  Cluster se soporta en todos los binarios MySQL-max en la serie 5.0, excepto lo que se explica en el siguiente parágrafo. Puede determinar si su servidor soporta o no NDB usando los comandos `SHOW VARIABLES LIKE 'have_%';` o `SHOW ENGINES;`. (Consulte Sección 5.1.2, “El servidor extendido de MySQL `mysqld-max`” para más información.)

  Usuarios de Linux , tengan en cuenta que `NDB` *no* se incluye en los RPM estándar de MySQL server. Desde MySQL 5.0.4, hay paquetes RPM separados para el motor NDB junto con herramientas de administración; consulte la sección de descargas NDB RPM de MySQL 5.0 Downloads . (Antes de 5.0.4, debe usar los binarios `-max` proporcionados como `.tar.gz` . Todavía es posible, pero no es un requerimiento, así que puede usar su administración de RPMs de Linux si lo prefiere.) Puede obtener soporte NDB compilando los binarios `-max` de las fuentes, pero no es necesario hacerlo para usar MySQL Cluster. Para descargar los últimos binarios, RMP o distribución fuente en la serie MySQL 5.0 visite http://dev.mysql.com/downloads/mysql/5.0.html.

- *¿Cuánta RAM necesito? ¿Es posible usar memoria de disco?*

  Actualmente, Cluster sólo funciona en memoria. Esto significa que todos los datos de tabla (incluyendo índices) se almacena en RAM. Por lo tanto, si sus datos ocupan 1 GB de espacio y quiere replicarlo en el cluster, necesitará 2 GB de memoria para ello. Esto es a parte de la memoria necesaria para el sistema operativo y cualquier aplicación ejecutándose en las máquinas del cluster.

  Puede usar la siguiente fórmula para obtener una estimación aproximada de cuánta RAM necesita para cada nodo de datos en el cluster:

  ```
  (SizeofDatabase * NumberOfReplicas * 1.1 ) / NumberOfDataNodes
  ```

  Para calcular los requerimientos de memoria con más exactitud debe determinar, para cada tabla en la base de datos del cluster, el espacio de almacenamiento requerido por registro (consulte Sección 11.5, “Requisitos de almacenamiento según el tipo de columna” para más detalles), y multiplicarlo por el número de registros. Debe recordar tener en cuenta cualquier índice de columna como sigue:

  - Cada clave primaria o índice hash creado para una tabla `NDBCluster` necesita 21-25 bytes por registro. Estos índices usan `IndexMemory`.

  - Cada índice ordenado necesita 10 bytes de almacenamiento por registro, usando `DataMemory`.

- Crear una clave primaria o índice único también crea un índice ordenado, a no ser que este índice se cree con `USING HASH`. En otras palabras, si se crea sin `USING HASH`, una clave primaria o índice único en una tabla Cluster ocupará hasta 31-35 bytes por registro en MySQL 5.0.

  Tenga en cuenta que crear tablas MySQL Cluster con `USING HASH` para todas las claves primarias e índices únicos generalmente causará que las actualizaciones de tablas sean más rápidas. Esto es debido al hecho que se necesita menos memoria (ya que no se crean índices únicos), y que debe usarse menos CPU (ya que se deben leer y actualizar menos índices).

Es especialmente importante tener en mente que **cada** tabla en MySQL Cluster debe tener clave primaria, que el motor NDB creará una clave primaria automáticamente si no se define y que esta clave primaria se crea sin `USING HASH`.

No hay una forma sencilla en MySQL 5.0 para determinar exactamente cuánta memoria se está usando para almacenar índices Cluster en un momento dado; sin embargo, las advertencias se escriben en el log del cluster cuando el 80% de memoria `DataMemory` y/o `IndexMemory` está en uso, y de nuevo al 85%, 90% etc.

A menudo vemos preguntas de usuarios que reportan que, cuando intentan rellenar una base de datos cluster, el proceso de carga termina prematuramente y aparece un mensaje de error como este:

```
ERROR 1114: The table 'my_cluster_table' is full
```

Cuando esto ocurre, la causa es que su configuración no proporciona suficiente RAM para todos los datos de tablas e índice , *incluyendo la clave primaria requerida por el motor `NDB` y creada automáticamente en el caso que la definición de tabla no incluya la definición de la clave primaria*.

Vale la pena tener en cuenta que todos los nodos de datos deben tener la misma cantidad de RAM, ya que ningún nodo de datos en el cluster puede usar más memoria que la mínima cantidad disponible para cualquier nodo de datos individual. En otras palabras, si hay tres máquinas con nodos de datos y dos tienen 3 GB RAM disponibles y otro sólo 1 GB de RAM, entonces cada nodo sólo puede usar 1 GB para el cluster.

- *En caso de un fallo catastrófico — por ejemplo, que la ciudad entera se queda sin electricidad **y** mi UPS falla — ¿perdería todos mis datos?*

  Todas las transacciones con commit se loguean. Por lo tanto, aunque es posible perder algunos datos en caso de catástrofe, debería ser algo limitado. La pérdida de datos puede reducirse minimizando el número de operaciones por transacción. (No es buena idea realizar un gran número de operaciones por transacción en cualquier caso.)

- *¿Es posile usar índices `FULLTEXT` con Cluster?*

  La indexación `FULLTEXT` no se soporta en el motor `NDB` , o por cualquier motor que no sea `MyISAM`. Estamos trabajando para añadir esta funcionalidad en nuevas versiones.

- *¿Puedo ejecutar múltiples nodos en una sola máquina?*

  Es posible pero no recomendable. Una de las razones principales para ejecutar un cluster es proporcionar redundancia; para tener todos los beneficios de esta redundancia, cada nodo debe residir en máquinas separadas. Si tiene múltiples nodos en una misma máquina y esta falla, pierde todos estos nodos. Dado que MySQL Cluster puede ejecutarse en hardware dedicado cargado con sistemas operativos de bajo o ningún coste, vale la pena el gasto en una o dos máquina extra para guardar datos críticos. También vale la pena tener en cuenta que los requerimientos para una máquina cluster ejecutando un nodo de administración son mínimos; esta tarea puede realizarse con una CPU 200 MHz

Pentium y suficiente RAM para el sistema operativo más una mínima cantidad de sobrecarga para los procesos `ndb_mgmd` y `ndb_mgm` .

- *¿Puedo añadir nodos en un cluster sin reiniciarlo?*

  No. Un reinicio es necesario para añadir nuevos nodos MGM o SQL. Al añadir nodos de datos el proceso es más complejo, y necesita los siguientes pasos:

  - Hacer una copia de seguridad completa de todos los datos del cluster.
  - Parar todo el cluster y procesos de los nodos.
  - Reiniciar el cluster, usando la opción `--initial`
  - Restaurar todos los datos desde la copia de seguridad

  En el futuro, esperamos implementar capacidad de reconfiguración “hot” para MySQL Cluster para minimizar (o eliminar) los requerimientos para reiniciar el cluster al añadir nuevos nodos.

- *¿Hay alguna limitación que deba tener en cuenta al usar Cluster?*

  Las tablas `NDB` tienen las siguientes limitaciones:

  - No se soportan todos los conjuntos de caracteres y colaciones.
  - Índices `FULLTEXT` y prefijo no están soportados. Sólo pueden indexarse columnas completas.
  - Capítulo 18, *Extensiones espaciales de MySQL* no se soportan.
  - Sólo se soporta rollback completo para transacciones. Los rollback parciales y rollbacks en checkpoints no se soportan.
  - El máximo número de atributos permitidos por tabla es 128, y los nombres de atributo no pueden tener más de 31 caracteres. Para cada tabla, la longitud máxima combinada del nombre de tabla y de base de datos es 122 caracteres.
  - El tamaño máximo para un registro de tabla es de 8 kilobytes, sin contar BLOBs. No hay límite para el número de registros por tabla; los límites de tamaño de tabla dependen en un número de factores, en particular la cantidad de RAM disponible para cada nodo de datos.
  - El motor NDB no soporta claves foráneas. Como con tablas MyISAM, se ignoran.
  - No se soporta el caché de consulta.

  Para información adicional de limitaciones de cluster, consulte Sección 16.8, “Limitaciones conocidas de MySQL Cluster”.

- *¿Cómo importo una base de datos existente en un cluster?*

  Puede importar bases de datos en MySQL Cluster como con cualquier otra versión de MySQL. A parte de las limitaciones mencionadas anteriormente, el único requerimiento especial es que cualquier tabla que se incluya en el cluster debe usar el motor `NDB`. Esto significa que las tablas deben crearse con la opción `ENGINE=NDB` o `ENGINE=NDBCLUSTER`.

- *¿Cómo se comunican los nodos del cluster entre ellos?*

  Los nodos del Cluster pueden comunicarse mediante tres protocolos: TCP/IP, SHM (memoria compartida), y SCI (Scalable Coherent Interface). Donde está disponible, SHM se usa por defecto

entre nodos residentes en el mismo equipo de cluster. SCI es un protocolo de alta velocidad (1 gigabit por segundo o más), alta disponibilidad usado en construir sistemas escalables de multi procesador; requiere hardware especial y drivers. Consulte Sección 16.7, “Usar interconexiones de alta velocidad con MySQL Cluster” para más información usando SCI como mecanismo de transporte en MySQL Cluster.

- *¿Qué es un “árbitro”?*

  Si uno o más nodos en un cluster fallan, es posible que no todos los nodos del cluster será capaz de “verse” entre ellos. De hecho, es posible que dos conjuntos de nodos puedan a estar aislados de los otros en una partición de red, también conocido como un escenario “split brain”. Este tipo de situación no es deseable ya que cada conjunto de nodos trata de comportarse como si fuera el cluster entero.

  Cuando caen los nodos del cluster, hay dos posibilidades. Si más del 50% de los nodos restantes pueden comunicarse entre ellos, entonces tenemos lo que a veces se llama “reglas de mayoría” , y este conjunto de nodos se consideran como el cluster. El árbitro entra en juego cuando hay un número impar de nodos: en tales casos, el conjunto de nodos al que pertenece el árbitro se considera el cluster, y los nodos que no pertenecen a este grupo se paran.

  La información anterior está simplificada; a continuación hay una explicación más completa:

  Cuando todos los nodos en al menos un grupo de nodos está vivo, la partición de la red no es un problema, porque ninguna porción del cluster puede formar un cluster funcional. El problema real es cuando un grupo no tiene todos sus nodos vivos, en tal caso la partición de red (el escenario “split-brain” mencionado anteriormente) es posible. Cuando se necesita un árbitro, que normalmente es el servidor de administración; sin embargo, es posible configurar cualquier MySQL Server en el cluster para que actúe como el árbitro. El árbitro acepta el primer conjunto de nodos del cluster que contacten con el, y le dice al resto que mueran. La selección del árbitro se controla mediante el parámetro de configuración `ArbitrationRank` para los nodos MySQL Server y de administración. (Consulte Sección 16.4.4.4, “Definición del servidor de administración de MySQL Cluster” para más detalles.) Debe tener en cuenta que el rol de administrador no impone ninguna demanda en la máquina designada, y por lo tanto la máquina árbitro no necesita ser particularmente rápida o tener memoria extra para este propósito.

- *¿Qué tipos de columna soport MySQL Cluster?*

  MySQL Cluster soporta todos los tipos de columna MySQL usuales, con la excepción de los asocidados con las extensiones espaciales. (Consulte Capítulo 18, *Extensiones espaciales de MySQL*.) Además, hay algunas diferencias respecto a los índices cuando se usan con tablas NDB. **Nota**: En MySQL 5.0, las tablas Cluster (esto es, tablas creadas con `ENGINE=NDBCLUSTER`) tiene sólo registros de longitud fija. Esto significa que (por ejemplo, cada registro conteniendo una columna `VARCHAR(255)` necesitará 256 bytes de almacenamiento para esa columna, independientemente del tamaño de los datos almacenados. Este punto se arreglará en futuras versiones.

  Consulte Sección 16.8, “Limitaciones conocidas de MySQL Cluster” para más información.

- *¿Cómo arranco y paro MySQL Cluster?*

  Es necesario arrancar cada nodo en el cluster por separado en el siguiente orden:

  1. Arranque el nodo de administración con el comando `ndb_mgmd` .
  2. Arranque cada nodo de datos con el comando `ndbd`.
  3. Arranque cada servidor MySQL (nodo SQL) usando `mysqld_safe --user=mysql &`.

Cada uno de estos comandos debe ejecutarse en una shell de sistema en la máquina que contenga el nodo afectado. Puede verificar que el cluster está en ejecución arrancando el cliente de administración MGM . `ndb_mgm` en la máquina con el nodo MGM.

- *¿Qué ocurre a los datos del cluster cuando el cluster se para?*

  Los datos en memoria de los nodos de datos se escriben en disco, y se recargan en memoria la siguiente vez que se inicia el cluster.

  Para parar el cluster, introduzca lo siguiente en una shell en la máquina del nodo MGM:

  ```
  shell> ndb_mgm -e shutdown
  ```

  Esto hace que `ndb_mgm`, `ndb_mgm`, y cualquier proceso `ndbd` termina correctamente. MySQL servers corriendo como nodos Cluster SQL pueden pararse usando `mysqladmin shutdown`.

  Para más información, consulte Sección 16.6.1, “Comandos del cliente de administración” y Sección 16.3.6, “Apagado y encendido seguros”.

- *¿Es útil tener más de un nodo de administración para el cluster?*

  Puede ser útil para ser más seguro. Sólo un nodo MGM controla el cluster en un momento dado, pero es posible configurar un MGM como primario, y otro o más nodos adicionales para tomar el control en caso de fallo del nodo MGM primario.

- *¿Puedo mezclar hardware y sistemas operativos distintos en un Cluster?*

  Sí, mientras todas las máquinas y sistemas operativos tengan la misma endian. Es posible usar distintas versiones de MySQL Cluster en nodos distintos; sin embargo, recomendamos que esto se haga sólo como parte del procedimiento de actualización.

- *¿Puedo ejecutar dos nodos de datos en una misma máquina? ¿Dos nodos SQL?*

  Sí. En el caso de múltiples nodos de datos, cada nodo debe usar un directorio de datos distinto. Si quiere ejecutar múltiples nodos SQL en una máquina, entonces cada instancia de `mysqld` debe usar un puerto TCP/IP distinto.

- *¿Puedo usar nombres de equipo con MySQL Cluster?*

  Sí, es posible usar DNS y DHCP para equipos del cluster. Sin embargo, si su aplicación necesita disponibilidad de "cinco nueves", recomendamos usar direcciones IP fijas. Esto es porque hacer la comunicación entre equipos del cluster dependiente de estos servicios introduce puntos de fallo adicionales, y mientras menos haya, mejor.

16.11. Glosario de MySQL Cluster

Los siguientes términos son útiles para enteneder MySQL Cluster o tienen significado eespecial cuando se usan referidos a ellos.

- **Cluster**:

  En sentido genérico, un cluster es un conjunto de máquinas funcionando como unidad y trabajando juntas para tratar una única tarea.

  `NDB Cluster`:

Este es el motor de almacenamiento usado en MySQL para implementar almacenamiento de datos, recuperación y administración distribuída entre varias máquinas.

MySQL Cluster:

Se refiere a un grupo de máquinas trabajando juntas usando el motor `NDB` para soportar una base de datos MySQL distribuída en una arquitectura de *compartición nula* usando *almacenamiento en memoria*.

- **Ficheros de configuración**:

  Ficheros de texto conteniendo directivas e información respecto al cluster, sus máquinas y sus nodos. Son leídos por los nodos de administración de cluster y cuando arranca el cluster. Consulte Sección 16.4.4, “Fichero de configuración” para detalles.

- **Copia de seguridad**:

  Una copia completa de todos los datos de cluster, transacciones y logs, guardados en disco y otro medio de almacenamiento.

- **Restauración**:

  Retornar el cluster a un prévio estado, como se almacenó en la copia de seguridad.

- **Checkpoint**:

  Generalmente hablando, cuando los datos se guardan en disco, se dice que se llega a un checkpoint. Más específicamente para el cluster, es un punto en tiempo donde todas las transacciones que han hecho un commit se guardan en disco. Respecto al motor `NDB`, hay dos clases de checkpoints que trabajan juntas para asegurar que se mantiene una vista consistente del cluster:

  - **Local Checkpoint (LCP)**:

    Este es un checkpoint específico a un nodo; sin embargo LCP se realizan para todos los nodos de forma más o menos concurrentes en todo el cluster. Un LCP inplica guardar todos los datos de los nodos en disco, y esto ocurre normalmente cada pocos minutos. El intervalo preciso varía, y depende de la cantidad de datos almacenada por el nodo, el nivel de la actividad del cluster y otros factores.

  - **Global Checkpoint (GCP)**:

    Un GCP se realiza cada pocos segundos, cuando las transacciones para todos los nodos se sincronizan y el log de redo se vuelca en disco.

- **Equipo Cluster**:

  Una máquina que forma parte del MySQL Cluster. Un cluster tiene una estructura *física* y una *lógica* . Físicamente, el cluster consiste en un número de máquinas , conocidas como *equipos del cluster* (o más símplemente *equipos* . Consulte **Nodo** y **Grupo de Nodos** a continuación.

- **Nodo**:

  Se refiere a un unidad lógica o funcional de MySQL Cluster, y a veces se denomina como *nodo de cluster* . En el contexto de MySQl Cluster, usamos el término “nodo” para indicar un *proceso* en lugar de un componente físico del cluster. Hay tres tipos de nodo requeridos para implementar un MySQL Cluster. Son:

  - **Nodos de administración (MGM)**:

Administra los otros nodos dentro del MySQL Cluster. Proporciona datos de configuración de los otros nodos; arranca y para nodos; trata partición de red; crea copias de seguridad y restaura desde las mismas, y así.

- **Nodos SQL (MySQL server)**:

  Instancias de MySQL Server que sirve como front end para cuardar datos en los **nodos de datos** del cluster. Clientes que quieran almacenan, recuperar o actualizar datos pueden acceder a un nodo SQL sólo como si fuera cualquier otro MySQL Server, empleando los métodos de autenticación usual y API; la distribución subyacente de datos entre grupos de nodos es transparente a los usuarios y aplicaciones. Los nodos SQL acceden a las bases de datos del cluster como un total sin tener en cuenta la distribución de datos entre distintos nodos de datos o máquinas del cluster.

- **Nodos de datos**:

  Estos nodos almacenan los datos. Los fragmentos de datos se almacenan en un conjunto de grupos de nodos. Cada uno de los nodos creando un grupo de nodos almacena una réplica del fragmento para el que ese grupo de nodos es responsable. Actualmente un único cluster puede soportar hasta 48 nodos de datos en total.

Es posible que más de un nodo coexista en una única máquina. (De hecho, es posible tener un cluster completo en una única máquina, aunque *no* es recomendable hacerlo en un entorno de producción.) Puede ser útil recordar que, cuando se trabaja con MySQL Cluster, el término *máquina* se refiere a un componente físico del cluster mientras que *nodo* es un componente lógico o funcional. (un proceso).

Nota respecto a términos obsoletos: En versiones más antiguas de la documentación de MySQL Cluster , los nodos de datos se llaman a veces "nodos de bases de datos" o "nodos DB". Además, los nodos SQL a veces se conocen como "nodos cliente" o "nodos API". Esta terminología antigua ha quedado obsoleta para minimizar confusión, y por estas razones debería evitarse.

- **Grupo de nodos**:

  Conjunto de nodos de datos. Todos los nodos de datos en un grupo de nodos contienen los mismos datos (fragmentos), y todos los nodos en un mismo grupo deben estar en distintas máquinas. Es posible controlar qué nodos pertenecen a qué grupos de nodos.

- **Fallo de nodo**:

  MySQL Cluster no sólo depende de la funcionalidad de un único nodo en el cluster; el cluster puede continuar funcionando si uno o más nodos fallan. El número preciso de fallos de nodo que puede tolerar un único cluster depende en el número de nodos de la configuración del cluster.

- **Reinicio de nodo**:

  El proceso de restaurar un nodo de cluster fallido.

- **Reinicio de nodo inicial**:

  El proceso de arrancar un nodo de cluster con su sistema de fichero eliminado. Esto a veces se usa en actualizaciones de software y en otras circunstancias especiales.

- **Fallo de sistema**:

  Puede ocurrir cuando han fallado tantos nodos que el estado del cluster no puede garantizarse.

- **Reinicio de sistema**:

El proceso de reiniciar el cluster y reinicializar su estado de logs de disco y checkpoints. Se requiere tras una parada planficada o no del cluster.

- **Fragmento**:

  Una porción de una tabla; en el motor `NDB` una tabla se divide y almacena como un número de fragmentos. Un fragmento es a veces llamado “partición”; sin embargo, “fragmento” es la denominación preferida.Las tablas fragmentadas en MySQL Cluster se usan para facilitar balanceo de carga entre máquinas y nodos.

- **Réplica**:

  Bajo el motor `NDB` , cada fragmento de tabla tiene un número de réplicas almacenadas en otros nodos de datos para proporcionar redundancia. Hay actualmente 4 réplicas por fragmento.

- **Transporter**:

  Protocolo que proporciona transferencia de datos entre nodos. MySQL Cluster soporta 4 tipos distintos de conexiones de transporters:

  - TCP/IP (local)

    Protocolo de red habitual que existe bajo HTTP, FTP (y así) en Internet.

  - TCP/IP (remoto)

    Lo mismo que el anterior, excepto que se usa para comunicación remota.

  - SCI

    Scalable **C**oherent **I**nterface es un protocolo de alta velocidad usado para montar sistemas multiprocesador y aplicaciones paralelas. El uso de SCI con MySQL Cluster requiere hardware especializado y se discute en Sección 16.7.1, “Configurar MySQL Cluster para que utilice Sockets SCI”. Para una introducción básica a SCI, consulte este ensayo de dolphinics.com.

  - SHM

    shared **m**emory segments (segmentos de memoria compartida). Donde se soporta, SHM se usa automáticamente para conectar nodos en la misma máquina. La página de man Unix para `shmop(2)` es un buen sitio para obtener información adicional acerca de este tema.

  Nota: El transporter del cluster es interno. Las aplicaciones que usan MySQL Cluster se comunican con nodos SQL como se hace con cualquier otra versión de MySQL Server (via TCP/IP, o a través del uso de sockets Unix o Windows named pipes). Las consultas pueden enviarse y recibirse los resultados usando la API estándar MySQL .

- **`NDB`**:

  Significa **N**etwork **D**ata**b**ase, y se refiere al motor de almacenamiento usando para permitir MySQL Cluster. El motor `NDB` soporta todos los tipos de columna MySQL habituales y comandos SQL , y cumple las reglas ACID. Este motor proporciona soporte para transacciones (commits y rollbacks).

- **Arquitectura de compartición cero**:

  Arquitectura ideal para MySQL Cluster. En un entorno sin compartición, cada nodo se ejecuta en máquinas separadas. La ventaja de este entorno es que ninguna máquina puede ser un punto de fallida único o como cuello de botella del sistema.

- **Almacenamiento en memoria**:

  Todos los datos almacenados en cada nodo de datos se mantiene en memoria en la máquina del nodo. Para cada nodo de datos en el cluster, debe tener disponible una cantidad de RAM igual al tamaño de la base de datos multiplicado por el número de réplicas, dividido por el número de nodos de datos. Por lo tanto, si la base de datos ocupa 1 GB de memoria, y quiere tener 4 réplicas en el cluster, necesita para cada nodo un mínimo de 500 . Tenga en cuenta que esto es además de cualquier requerimiento para el sistema operativo u otra aplicación que puede ejecutarse en el equipo.

- **Tabla**:

  Como es normal en el contexto de bases de datos relacionales, el término “tabla” denota un conjunto ordenado de requistros de idéntica estructura. En MySQL Cluster, una tabla de base de datos se almacena en un nodo de datos como un conjunto de fragmentos, cada uno de ellos se replica en nodos de datos adicionales. El conjunto de nodos de datos replicando el mismo fragmento o conjunto de fragmentos se conoce como *grupo de nodos*.

- **Programas del Cluster**:

  Son programas de línea de comandos usados para ejecutar, configurar y administrar MySQL Cluster. Incluyen demonios:

  - `ndbd`:

    Demonio de nodo de datos (ejecuta un proceso de nodo de datos)

  - `ndb_mgmd`:

    Demonio de servidor de administración (ejecuta un proceso de servidor de administración)

  y programas cliente:

  - `ndb_mgm`:

    El cliente de administración (proporciona una interfaz para ejecutar comandos de administración)

  - `ndb_waiter`:

    Usado para verificar el estado de todos los nodos del cluster

  - `ndb_restore`:

    Restaura datos del cluster de una copia de seguridad

  Para más información de estos programas, consulte Sección 16.5, “Gestión de procesos en MySQL Cluster”.

- **Log de eventos**:

  MySQL Cluster registra eventos por categoría (arranque, parada, errores,checkpoints, y así), prioridad, y severidad. Un listado completo de todos los eventos reportables pueden encontrarse en Sección 16.6.2, “Informes de eventos generados por MySQL Cluster”. Los logs de eventos son de dos tipos:

  - **Log de Cluster**:

    Mantiene un registro de todos los eventos reportables deseados para el cluster entero.

- **Log de Node**:

  Un log separado que se mantiene para cada nodo individual.

Bajo circunstancias normales, es necesario y suficiente mantener y examinar sólo el log del cluster. Los logs de nodo tienen que ser consultados sólo para desarrollo de aplicaciones y depuración.

Capítulo 17. Introducción a MaxDB

Tabla de contenidos

MaxDB is an enterprise-level database. MaxDB is the new name of a database management system formerly called SAP DB.

17.1. Historia de MaxDB

La historia de SAP DB se remonta hacia los principios de 1980 cuando fue desarrollada como un producto comercial(ADABAS). Esta Base de Datos fue cambiando de nombre varias veces desde ese entonces. Cuando SAP AG, una compañía con sede en Walldorf, Alemania, se hizo cargo del desarrollo del producto, fue bautizada como SAP DB.

SAP desarrollo esta base de datos para funcionar como sistema de almacenamiento de todos los sistemas de misión critica de las Aplicaciones de SAP, llamadas R/3. SAP DB fue pensada para proveer una alternativa a sistemas de bases de datos como Oracle, Microsoft SQL Server, y DB2 de IBAM. En Octubre de 2000, SAP AG libero bajo la licencia GNU GPL(ver Apéndice I, *GNU General Public License*), convirtiéndola en Software Open Source. En Octubre de 2003, mas de 2,000 clientes de SAP AG estaban utilizando SAP DB como su principal sistema de bases de datos y otros 2000 clientes estaban utilizando de forma separada como parte de la solución APO/Live cache

En Mayo de 2003 una alianza tecnológica fue conformada entre MySQL AB y SAP AG. Esa alianza fue llamada MySQL AB se encargara del futuro desarrollo de SAP DB, de su nuevo nombre, y de la comercialización de licencias de la re bautizada SAP DB para aquellos clientes que no quieran estar bajo las restricciones impuestas por utilizar esa base de datos bajo el licenciamiento GNU GPL (see Apéndice I, *GNU General Public License*). En Agosto de 2003, SAP DB fue re bautizada como MaxDB por MySQL AB.

17.2. Licenciamiento y soporte

MaxDB puede ser utilizada bajo los mismos sistemas de licenciamiento que los otros productos distribuidos por MySQL AB. De esta manera MaxDB esta disponible bajo la GNU General Public Licence y bajo la licencia comercial. Para mayor informacion sobre licenciamiento ver: http://www.mysql.com/company/legal/licensing/.

MySQL ofrece soporte para MaxDB a quienes no sean clientes de SAP.

La primer version actualizada fue MaxDB 7.5.00, que fue lanzada en Noviembre de 2003

17.3. Enlaces relacionados con MaxDB

La pagina principal para MaxDB es http://www.mysql.com/products/maxdb. La informacion anteriormente disponible en http://www.sapdb.org se traslado a el link anterior.

17.4. Conceptos básicos de MaxDB

MaxDB opera como un producto cliente/servidor. Fue desarrollado para cubrir las demandas de las instalaciones que requieren de una gran volumen de procesamiento de transacciones. Tanto el back up en línea como expansión de la base de datos están soportados. El Servidor Cluster de Microsoft tiene soporte directo para múltiples instalaciones de servidores; otros requerimientos deben ser escritos manualmente. Las herramientas de administración son provistas tanto en aplicaciones Desktop como implementaciones basadas en browser.

17.5. Diferencias de prestaciones entre MaxDB y MySQL

La siguiente lista brinda un sumario de las principales diferencias entre MaxDB y MySQL; Esta lista es parcial.

- MaxDB corre en la modalidad de un sistema cliente/servidor. MaxDB runs as a client/server system. MySQL puede funcionar tanto cliente/servidor como tambien en un sistema ebedded.
- MaxDB no funciona en todas las platformas soportadas por MySQL. Por ejemplo, MaxDB no corre sobre el sistema operativo de IBM OS/2. MaxDB might not run on all platforms supported by MySQL.
- MaxDB uses a proprietary network protocol for client/server communication. MySQL uses either TCP/IP (with or without SSL encryption), sockets (under Unix-like systems), or named pipes (under Windows NT-family systems).
- MaxDB supports stored procedures. For MySQL, stored procedures are implemented in version 5.0. MaxDB also supports programming of triggers through an SQL extension, which is scheduled for MySQL 5.1. MaxDB contains a debugger for stored procedure languages, can cascade nested triggers, and supports multiple triggers per action and row.
- MaxDB is distributed with user interfaces that are text-based, graphical, or Web-based. MySQL is distributed with text-based user interfaces only; graphical user interface (MySQL Control Center, MySQL Administrator) are shipped separately from the main distributions. Web-based user interfaces for MySQL are offered by third parties.
- MaxDB supports a number of programming interfaces that also are supported by MySQL. However, MaxDB does not support RDO, ADO, or .NET, all of which are supported by MySQL. MaxDB supports embedded SQL only with C/C++.
- MaxDB includes administrative features that MySQL does not have: job scheduling by time, event, and alert, and sending messages to a database administrator on alert thresholds.

17.6. Características de interoperabilidad entre MaxDB y MySQL

As part of MaxDB 7.6, the MaxDB Synchronization Manager is released. The Synchronization Manager supports creation of asynchronous replication scenarios between several MaxDB instances. However, interoperability features also are planned, so that the Synchronization Manager supports replication to and from a MySQL server.

In the first release, the Synchronization Manager supports inserting data into MySQL. This means that initially only replication from MaxDB to MySQL is supported. In the course of 2005, exporting of data from a MySQL server to the Synchronization Manager will be added, thus adding support for MySQL to MaxDB replication scenarios.

MaxDB 7.6, with the Synchronization Manager, was released as a beta version in January 2005. The production release is planned for April 2005.

17.7. Palabras reservadas de MaxDB

Like MySQL, MaxDB has a number of reserved words that have special meanings. Normally, they cannot be used as names of identifiers, such as database or table names. The following table lists reserved words in MaxDB, indicates the context in which those words are used, and indicates whether or not they have counterparts in MySQL. If such a counterpart exists, the meaning in MySQL might be identical or differing in some aspects. The main purpose is to list in which respects MaxDB differs from MySQL; therefore, this list is not complete.

Para la lista de palabras reservardas en MySQL, Visite Sección 9.6, “Tratamiento de palabras reservadas en MySQL”.

Reserved in MaxDB	Context of usage in MaxDB	MySQL counterpart
`@`	Can prefix identifier, like “@table”	Not allowed
`ADDDATE()`	Funcion SQL	`ADDDATE()`; nuevo en MySQL 4.1.1
`ADDTIME()`	Funcion SQL	`ADDTIME()`; nuevo en MySQL 4.1.1
`ALPHA`	Funcion SQL	Sin comparacion
`ARRAY`	Tipo de Dato	Sin Implementar
`ASCII()`	Funcion SQL	`ASCII()`, implementado pero con otro significado
`AUTOCOMMIT`	Transactions; `ON` by default	Transactions; `OFF` by default
`BOOLEAN`	Column types; `BOOLEAN` accepts as values only `TRUE`, `FALSE`, and `NULL`	`BOOLEAN` was added in MySQL 4.1.0; it is a synonym for `BOOL` which is mapped to `TINYINT(1)`. It accepts integer values in the same range as `TINYINT` as well as `NULL`. `TRUE` and `FALSE` can be used as aliases for `1` and `0`.
`CHECK`	`CHECK TABLE`	`CHECK TABLE`; similar, but not identical usage
`COLUMN`	Column types	`COLUMN`; noise word
`CHAR()`	SQL function	`CHAR()`; identical syntax; similar, not identical usage
`COMMIT`	Implicit commits of transactions happen when data definition statements are issued	Implicit commits of transactions happen when data definition statements are issued, and also with a number of other statements
`COSH()`	SQL function	Nothing comparable
`COT()`	SQL function	`COT()`; identical syntax and implementation
`CREATE`	SQL, data definition language	`CREATE`
`DATABASE`	SQL function	`DATABASE()`; `DATABASE` is used in a different context; for example, `CREATE DATABASE`
`DATE()`	SQL function	`CURRENT_DATE`
`DATEDIFF()`	SQL function	`DATEDIFF()`; new in MySQL 4.1.1
`DAY()`	SQL function	Nothing comparable
`DAYOFWEEK()`	SQL function	`DAYOFWEEK()`; by default, `1` represents Monday in MaxDB and Sunday in MySQL

DISTINCT	SQL functions AVG, MAX, MIN, SUM	DISTINCT; but used in a different context: SELECT DISTINCT
DROP	DROP INDEX, for example	DROP INDEX; similar, but not identical usage
EBCDIC()	SQL function	Nothing comparable
EXPAND()	SQL function	Nothing comparable
EXPLAIN	Optimization	EXPLAIN; similar, but not identical usage
FIXED()	SQL function	Nothing comparable
FLOAT()	SQL function	Nothing comparable
HEX()	SQL function	HEX(); similar, but not identical usage
INDEX()	SQL function	INSTR() or LOCATE(); similar, but not identical syntaxes and meanings
INDEX	USE INDEX, IGNORE INDEX and similar hints are used right after SELECT; for example, SELECT ... USE INDEX	USE INDEX, IGNORE INDEX and similar hints are used in the FROM clause of a SELECT query; for example, in SELECT ... FROM ... USE INDEX
INITCAP()	SQL function	Nothing comparable
LENGTH()	SQL function	LENGTH(); identical syntax, but slightly different implementation
LFILL()	SQL function	Nothing comparable
LIKE	Comparisons	LIKE; but the extended LIKE MaxDB provides rather resembles the MySQL REGEX
LIKE wildcards	MaxDB supports “%”, “_”, “Control-underline”, “Control-up arrow”, “*”, and “?” as wildcards in LIKE comparisons	MySQL supports “%”, and “_” as wildcards in LIKE comparisons
LPAD()	SQL function	LPAD(); slightly different implementation
LTRIM()	SQL function	LTRIM(); slightly different implementation
MAKEDATE()	SQL function	MAKEDATE(); new in MySQL 4.1.1
MAKETIME()	SQL function	MAKETIME(); new in MySQL 4.1.1
MAPCHAR()	SQL function	Nothing comparable
MICROSECOND()	SQL function	MICROSECOND(); new in MySQL 4.1.1
NOROUND()	SQL function	Nothing comparable
NULL	Column types; comparisons	NULL; MaxDB supports special NULL values that are returned by arithmetic operations that lead to an overflow or a division by zero; MySQL does not support such special values
PI	SQL function	PI(); identical syntax and implementation, but parentheses are mandatory in MySQL
REF	Data type	Nothing comparable
RFILL()	SQL function	Nothing comparable
ROWNO	Predicate in WHERE clause	Similar to LIMIT clause
RPAD()	SQL function	RPAD(); slightly different implementation

RTRIM()	SQL function	RTRIM(); slightly different implementation
SEQUENCE	CREATE SEQUENCE, DROP SEQUENCE	AUTO_INCREMENT; similar concept, but different implementation
SINH()	SQL function	Nothing comparable
SOUNDS()	SQL function	SOUNDEX(); slightly different syntax
STATISTICS	UPDATE STATISTICS	ANALYZE TABLE; similar concept, but different implementation
SUBSTR()	SQL function	SUBSTRING(); slightly different implementation
SUBTIME()	SQL function	SUBTIME(); new in MySQL 4.1.1
SYNONYM	Data definition language: CREATE [PUBLIC] SYNONYM, RENAME SYNONYM, DROP SYNONYM	Nothing comparable
TANH()	SQL function	Nothing comparable
TIME()	SQL function	CURRENT_TIME
TIMEDIFF()	SQL function	TIMEDIFF(); new in MySQL 4.1.1
TIMESTAMP()	SQL function	TIMESTAMP(); new in MySQL 4.1.1
TIMESTAMP() as argument to DAYOFMONTH() and DAYOFYEAR()	SQL function	Nothing comparable
TIMEZONE()	SQL function	Nothing comparable
TRANSACTION()	Returns the ID of the current transaction	Nothing comparable
TRANSLATE()	SQL function	REPLACE(); identical syntax and implementation
TRIM()	SQL function	TRIM(); slightly different implementation
TRUNC()	SQL function	TRUNCATE(); slightly different syntax and implementation
USE	Switches to a new database instance; terminates the connection to the current database instance; all subsequent commands are referred to this database instance	USE; identical syntax, but does not terminate the connection to the current database
USER	SQL function	USER(); identical syntax, but slightly different implementation, and parentheses are mandatory in MySQL
UTC_DIFF()	SQL function	UTC_DATE(); provides a means to calculate the same result as UTC_DIFF()
VALUE()	SQL function, alias for COALESCE()	COALESCE(); identical syntax and implementation
VARIANCE()	SQL function	VARIANCE(); new in MySQL 4.1.0
WEEKOFYEAR()	SQL function	WEEKOFYEAR(); new in MySQL 4.1.1

Capítulo 18. Extensiones espaciales de MySQL

Tabla de contenidos

MySQL 4.1 introuce las extensiones espaciales para permitir la generación, almacenamiento, y análisis de elementos geográficos. Actualmente, estas características están solo disponibles para tablas `MyISAM`.

Este capítulo trata los siguientes temas:

- La base de estas extensiones espaciales es el modelo de geometría OpenGIS
- Formatos de datos para representar datos espaciales
- Cómo usar datos espaciales en MySQL

- Uso de indexación para datos espaciales
- Diferencias entre MySQL y la especificación OpenGIS

18.1. Introducción

MySQL implementa extensiones espaciales siguiendo la especificación del `Consorcio Open GIS` (OGC), un consorcio internacional de más de 250 compañías, agencias y universidades que participan en el desarrollo de soluciones conceptuales públicamente disponibles y que pueden ser útiles para todo tipo de aplicaciones que manejan datos espaciales. El OGC matiene una web en http://www.opengis.org/.

En 1997, el Consorcio Open GIS publicó las *Especificaciones de características simples Open GIS para SQL*, un documento que propone diversas maneras conceptuales de extender un Sistema Gestor de Bases de Datos Relacionales para agregar soporte a datos espaciales. Esta especificación está disponible en http://www.opengis.org/docs/99-049.pdf. Contiene información adicional relevante relacionada con este capítulo.

MySQL implementa un subconjunto del entorno **SQL con Tipos Geométricos** propuesto por el OGC. Este término se refiere a un entorno SQL que ha sido extendido con un conjunto de tipos geométricos. Una columna SQL con valores geométricos se implementa como una columna que tiene un tipo geométrico. Las especificaciones describen un conjunto de tipos geométricos SQL, así como las funciones para analizar y crear valores geométricos sobre esos tipos.

Un **elemento geográfico** es cualquier cosa en el mundo que tenga una ubicación. Un elemento puede ser:

- Una entidad. Por ejemplo, una montaña, un lago, una ciudad.
- Un espacio. Por ejemplo, un área de código postal, los trópicos.
- Una ubicación definible. Por ejemplo, un cruce de carreteras, como un lugar partícular donde dos calles se interseccionan.

También puede encontrar documentos que utilicen el término **elementos geoespaciales** para referirse a elementos geográficos.

Geometría es otra palabra que denota un elemento geográfico. Originalmente la palabra **geometría** significaba medición de la tierra. Otro significado viene de la cartografía, refiriéndose a los elementos geométricos que los cartógrafos utilizan para crear mapas del mundo.

Este capítulo utiliza todos estos términos a modo de sinónimos: This chapter uses all of these terms synonymously: **elemento geográfico**, **elemento geoespacial**, **elemento**, o **geometría**. El término más comúnmente utilizado aquí es **geometría**.

Definamos una **geometría** como *un punto o conjunto de puntos representando cualquier cosa en el mundo que tenga una ubicación*.

18.2. El modelo geométrico OpenGIS

El conjunto de tipos geométricos propuesto por el entorno **SQL con Tipos Geométricos** de OGC's se basa en el **Modelo OpenGIS de Geometría**. En este modelo, cada objeto geométrico tiene las siguientes propiedades generales:

- Está asociado con un Sistema de Referencia Espacial, que describe el espacio de coordenadas en que el objeto está definido.

- Pertenece a alguna clase geométrica.

18.2.1. La jerarquía de las clases geométricas

Las clases geométricas definen una jerarquía de la siguiente manera:

- `Geometry` (no instanciable)
  - `Point` (instanciable)
  - `Curve` (no instanciable)
    - `LineString` (instanciable)
      - `Line`
      - `LinearRing`
  - `Surface` (no instanciable)
    - `Polygon` (instanciable)
  - `GeometryCollection` (instanciable)
    - `MultiPoint` (instanciable)
    - `MultiCurve` (no instanciable)
      - `MultiLineString` (instanciable)
    - `MultiSurface` (no instanciable)
      - `MultiPolygon` (instanciable)

No es posible crear objetos de clases no instanciables. Se pueden crear objetos de clases instanciables. Todas las clases tienen propiedades, y las clases instanciables pueden tener también aserciones (reglas que definen las instancias de clase válidas).

`Geometry` es la clase base. Es una clase abstracta. Las subclases instanciables de `Geometry` están restringidas a objetos geométricos cero-, uni-, y bi-dimensionales que existen en un espacio de coordenadas bidimensional. Todas las clases geométricas instanciables son definidas de manera que las instancias válidas de una clase geométrica sean topológicamente cerradas (es decir, que todas las geometrías definidas incluyen su límite).

La clase base `Geometry` tiene las subclases `Point`, `Curve`, `Surface`, y `GeometryCollection`:

- `Point` representa objetos de cero dimensiones.
- `Curve` representa objetos unidimensionales, y tiene la subclase `LineString`, con sub-subclases `Line` y `LinearRing`.
- `Surface` está diseñado para objetos bidimensionales y tiene la subclase `Polygon`.
- `GeometryCollection` tiene clases especializadas de cero, una y dos dimensiones llamadas `MultiPoint`, `MultiLineString`, y `MultiPolygon` para modelar geometrías correspondientes a colecciones de `Points`, `LineStrings`, y `Polygons`, respectivamente. `MultiCurve` y `MultiSurface` han sido introducidas como superclases abstractas que generalizan las interfaces de la colección para manejar `Curves` y `Surfaces`.

Geometry, Curve, Surface, MultiCurve, y MultiSurface están definidas como clases no instanciables. Definen un conjunto común de métodos para sus subclases y se incluyen para ser extendidas.

Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString, y MultiPolygon son clases instanciables.

18.2.2. La clase Geometry

Geometry es la clase base de la jerarquía. Es una clase no instanciable, pero tiene unas cuantas propiedades que son comunes para todos los valores geométricos creados con cualquiera de las subclases de Geometry. Estas propiedades están descritas en la siguiente lista. (Algunas subclases en concreto tienen sus propiedades específicas, descritas más tarde.)

Propiedades de Geometry

Un valor geométrico tiene las siguientes propiedades:

- Su **tipo**. Cada geometría pertenece a una de las clases instanciables de la jerarquía.
- Su **SRID**, o IDentificador de Referencia eSpacial. Este valor identifica el Sistema de Referencia Espacial asociado a la geometría, que describe el espacio de coordenadas en el que la geometría está definida.

  ç En MySQL, el valor SRID es simplemente un entero asociado con el valor de la geometría. Todos los cálculos se hacen asumiendo una geometría euclídea (planar).
- Sus coordenadas en este Sistema de Referencia Espacial, representadas como números de doble precisión (ocho bytes). Todas las geometrías no vacías incluyen al menos un par de coordenadas (X,Y). Las geometrías vacías no contienen coordenadas.

  Las coordenadas están relacionadas con el SRID. Por ejemplo, en diferentes sistemas de coordenadas, la distancia entre dos objetos puede diferir aún cuando los objetos tengan las mismas coordenadas, porque la distancia en sistemas de coordenadas **planares** y la distancia en sistemas **geocéntricos** (coordenadas en la superficie de la tierra) son cosas diferentes.
- Su **interior**, **límite**, y **exterior**.

  Cada geometría ocupa una posición en el espacio. El exterior de una geometría es todo el espacio no ocupado por la geometría. El interior es el espacio ocupado por la geometría. El límite es la interfaz entre el interior y el exterior de la geometría.
- 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))
  ```
- Si el valor es **simple** o **no-simple**. Los valores geométricos de tipo (LineString, MultiPoint, MultiLineString) son o simples, o no-simples. Cada tipo determina sus propias aserciones para ser simple o no-simple.
- Si el valor es **cerrado** o **no cerrado**. Los valores geométricos de tipo (LineString, MultiString) son o cerrados o no cerrados. Cada tipo determina sus propias aserciones para ser cerrado o no cerrado.
- Si el valor es **vacío** o **no vacío**. Una geometría es vacía si no tiene ningún punto. El exterior, interior, y límite de de una geometría vacía no están definidos (es decir, se representan por un valor NULL). Una geometría vacía está definida para ser siempre simple, y tiene un área de 0.

- Su **dimensión**. Una geometría puede tener una dimensión de −1, 0, 1, o 2:
  - −1 para una geometría vacía.
  - 0 para una geometría sin longitud ni área.
  - 1 para una geometría con longitud diferente de cero y área igual a cero.
  - 2 para una geometría con área diferente de cero.

  Los objetos Point tienen una dimensión de cero. Los objetos LineString tienen una dimensión de 1. Los objetos Polygon tienen una dimensión de 2. Las dimensiones de los objetos MultiPoint, MultiLineString, y MultiPolygon son las mismas que las dimensiones de los elementos que los componen.

18.2.3. La clase Point

Un Punto es una geometría que representa una ubicación única en un espacio de coordenadas.

Ejemplos de Point

- Imagine un mapa a gran escala del mundo con muchas ciudades. Un objeto Point podría representar cada ciudad.
- En un mapa de una ciudad, un objeto Point podría representar una parada de bus.

Propiedades de Point

- Valor de la coordenada X.
- Valor de la coordenada Y.
- Point es definido como una geometría cero-dimensional.
- El límite de un Point es el conjunto vacío.

18.2.4. La clase Curve

Una Curva es una geometría unidimensional, normalmente representada por una secuencia de puntos. Las subclases particulares de Curve definen el tipo de interpolación entre puntos. Curve es una clase no instanciable.

Propiedades de Curve

- Una Curva tiene las coordenadas de sus puntos.
- Una Curva está definida como una geometría unidimensional.
- Una Curva es simple si no pasa sobre el mismo punto dos veces.
- Una Curva es cerrada si su punto inicial es igual a su punto final.
- El límite de una Curva cerrada está vacío.
- El límite de una Curva no cerrada consiste en sus dos puntos finales.
- Una Curva que es simple y cerrada es un Anillo Linear (LinearRing).

18.2.5. La clase LineString

Una LineString es una Curva con interpolación linear entre puntos.

Ejemplos de LineString

- En un mapa del mundo, los objetos LineString podrían representar ríos.
- En un mapa de una ciudad, los objetos LineString podrían representar calles.

Propiedades de LineString

- Un LineString tiene coordenadas de segmentos, definidos por cada par consecutivo de puntos.
- Un LineString es una Línea (Line) si consiste exactamente en dos puntos.
- Un LineString es un LinearRing si es tanto cerrado como simple.

18.2.6. La clase Surface

Una Superficie (Surface) es una geometría bidimensional. Es una clase no instanciable. Su única subclase instanciable es Polygon.

Propiedades de Surface

- Una superficie está definida como una geometría bidimensional.
- La especificación OpenGIS define una Superficie simple como una geometría que consiste de un único “trozo” que está asociado a un único límite exterior y cero o más límites interiores.
- El límite de una Superficie simple es el conjunto de curvas cerradas correspondientes a sus límites exterior e interior.

18.2.7. La clase Polygon

Un Polígono (Polygon) es una Superficie planar que representa una geometría multicara. Se define por un único límite exterior y cero o más límites interiores, donde cada límite interior define un agujero en el Polígono.

Ejemplos de Polygon

- En un mapa de una región, objetos Polygon podrían representar bosques, distritos, etc.

Aserciones de Polygon

- El límite de un Polígono consiste en un conjunto de objetos LinearRing (es decir, objetos LineString que son tanto simples como cerrados) que construyen sus límites exterior e interior.
- Un Polígono no tiene anillos que se crucen. Los anillos en el límite de un Polígono pueden interseccionar un Punto, pero sólo como tangente.
- Un Polígono no tiene líneas, picos o valles.
- Un Polígono tiene un interior que consiste en un conjunto de puntos conectados.
- Un Polígono puede tener agujeros. El exterior de un Polígono con agujeros no está conectado. Cada agujero define un componente conectado del exterior.

Las aserciones precedentes hacen de un Polígono una geometría simple.

18.2.8. La clase GeometryCollection

Una ColecciónDeGeometrías (GeometryCollection) es una geometría que consiste en una colección de una o más geometrías de cualquier clase.

Todos los elementos en una GeometryCollection deben estar en el mismo Sistema de Referencia Espacial (es decir, en el mismo sistema de coordenadas). No existe ninguna otra restricción en los elementos de una GeometryCollection, aunque las subclases de GeometryCollection descritas en las siguientes secciones pueden restringir la membresía. Las restricciones se pueden basar en:

- Tipo de elemento (por ejemplo, un MultiPoint puede contener únicamente elementos de tipo Point)
- Dimensión
- Restricciones en el grado de sobreposición espacial entre elementos

18.2.9. La clase MultiPoint

Un MultiPoint es una colección de geometrías compuesta de elementos Point. Los puntos no están conectados ni ordenados de ningún modo.

Ejemplos de MultiPoint

- En un mapa mundial, un MultiPoint podría representar una cadena de pequeñas islas.
- En un mapa de una ciudad, un MultiPoint podría representar las oficinas de una empresa.

Propiedades de MultiPoint

- Un MultiPoint es una geometría cerodimensional.
- Un MultiPoint es simple si no hay dos de sus valores Point que sean iguales (tengan valores de coordenadas idénticos).
- El límite de un MultiPoint es el conjunto vacío.

18.2.10. La clase MultiCurve

Una MultiCurva (MultiCurve) es una colección de geometrías que se compone de elementos Curve. MultiCurve es una clase no instanciable.

Propiedades de MultiCurve

- Una MultiCurva es una geometría unidimensional.
- Una MultiCurva es simple si, y únicamente si, todos sus elementos son simples; las únicas interesecciones entre dos elementos cualquiera ocurren en puntos que están en los límites de ambos elementos.
- El límite de una MultiCurva se obtiene aplicando la “regla unión módulo 2 (mod 2 union rule)” (también conocida como la “regla par-impar (odd-even rule)”): Un punto está en el límite de unaMultiCurva si está en los límites de un número impar de elementos de MultiCurva.
- Una MultiCurva es cerrada si todos sus elementos son cerrados.

- El límite de una MultiCurva cerrada es siempre vacío.

18.2.11. La clase MultiLineString

Una MultiLineString es una colección de geometrías MultiCurve compuesta de elementos LineString.

Ejemplos de MultiLineString

- En el mapa de una región, una MultiLineString podría representar un sistema de ríos o de autopistas.

18.2.12. La clase MultiSurface

Una MultiSuperficie (MultiSurface) es una colección de geometrías compuesta de elementos de Superficie. MultiSurface es una clase no instanciable. Su única subclase instanciable es MultiPolygon.

Aserciones de MultiSurface

- Dos superficies de MultiSurface no tienen interiores que se intersecciones.
- Dos elementos de MultiSurface tienen límites que interseccionan como máximo en un número finito de puntos.

18.2.13. La clase MultiPolygon

Un MultiPolígono (MultiPolygon) es un objeto MultiSurface compuesto de elementos Polygon.

Ejemplos de MultiPolygon

- En el mapa de una región, un MultiPolygon podría representar un sistema de lagos.

Aserciones de MultiPolygon

- Un MultiPolygon no tiene dos elementos Polygon con interiores que se intersecciones.
- Un MultiPolygon no tiene dos elementos Polygon que se crucen (los cruces están también prohibidos por la aserción previa), o que se toquen en un número infinito de puntos.
- Un MultiPolygon no debe tener líneas de corte, valles, o picos. Un MultiPolygon es un conjunto de puntos regular y cerrado.
- Un MultiPolygon que tenga más de un Polygon tiene un interior que no está conectado. El número de componentes conectados del interior de un MultiPolygon es igual al número de valores Polygon en el MultiPolygon.

Propiedades de MultiPolygon

- Un MultiPolygon es una geometría bidimensional.
- El límite de un MultiPolygon es un conjunto de curvas cerradas (valores LineString) que corresponden a los límites de sus elementos Polygon.
- Cada Curva en el límite de un MultiPolygon está en el límite de exactamente un elemento Polygon.
- Cada Curva en el límite de un elemento Polygon está en el límite del MultiPolygon.

18.3. Formatos de datos espaciales soportados

Esta sección describe los formatos de datos espaciales estándar que suelen utilizarse para representar objetos geométricos en consultas. Son:

- Formato Well-Known Text (WKT)
- Formato Well-Known Binary (WKB)

Internamente, MySQL almacena los valores de geometría en un formato que no es idéntico a ninguno de los formatos WKT o WKB

18.3.1. Formato Well-Known Text (WKT)

La representación Well-Known Text (WKT) de Geometrías está diseñada para intercambiar datos geométricos en formato ASCII.

Ejemplos de representaciones WKT de objetos geométricos son:

- Un `Point`:

```
POINT(15 20)
```

  Nótese que las coordenadas del punto se especifican sin coma separadora.

- Una `LineString` con cuatro puntos:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

  Nótese que los pares de coordenadas de los puntos están separados por comas.

- Un `Polygon` con un anillo exterior y un anillo interior:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- Un `MultiPoint` con tres valores `Point`:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- Una `MultiLineString` con dos valores `LineString`:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- Un `MultiPolygon` con dos valores `Polygon`:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- Una `GeometryCollection` consistente en dos valores `Point` y una `LineString`:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

Puede encontrar una gramática que especifica las reglas formales para la escritura de valores WKT en el documento de especificación OGC referenciado al principio de este capítulo.

18.3.2. Formato Well-Known Binary (WKB)

La representación Well-Known Binary (WKB) de valores geométricos está definida por la especificación OpenGIS. También está definida en el estándar ISO “SQL/MM Part 3: Spatial”.

WKB se utiliza para intercambiar datos como cadenas binarias representadas por valores `BLOB` que contienen información geométrica WKB.

WKB utiliza enteros sin signo de un byte, enteros sin signo de cuatro bytes, y números de ocho bytes de doble precisión (formato IEEE 754). Un byte son ocho bits.

Por ejemplo, un valor WKB que corresponde a un `POINT(1 1)` consiste en esta secuancia de 21 bytes (cada uno representado aquí por dos dígitos hexadecimales):

```
0101000000000000000000F03F000000000000F03F
```

La secuencia puede descomponerse en los siguientes componentes:

```
Orden de byte : 01
Tipo WKB   : 01000000
X             : 0000000000000F03F
Y             : 0000000000000F03F
```

La representación de componentes es como sigue:

- El orden de byte puede ser 0 o 1, para indicar almacenamiento tipo little-endian o big-endian. Los órdenes de byte little-endian y big-endian son también conocidos como Representación de Datos de Red (Network Data Representation (NDR)) y Representación Externa de Datos (External Data Representation (XDR)), respectivamente.
- El tipo WKB es un código que indica el tipo de geometría. Los valores del 1 al 7 significan `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, y `GeometryCollection`.
- Un valor `Point` tiene coordenadas X e Y, cada una representada por un valor de doble precisión.

Los valores WKB que representan valores geométricos más complejos son representados por estructuras de datos más complejas, tal como se detalla en la especificación OpenGIS.

18.4. Crear una base de datos MySQL con capacidades espaciales

Esta sección describe los tipos de datos que usted puede utilizar para representar datos espaciales en MySQL, y las funciones disponibles para crear y obtener datos espaciales.

18.4.1. Tipos de datos espaciales de MySQL

MySQL tiene tipos de datos que corresponden a las clases OpenGIS. Algunos de estos tipos almacenan valores geométricos simples:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types, `POINT` and `LINESTRING` and `POLYGON`, restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` puede almacenar objetos de cualquier tipo. Los otros tipos de colección, `MULTIPOINT` y `MULTILINESTRING` y `MULTIPOLYGON` y `GEOMETRYCOLLECTION`, restringen sus miembros a aquellos que sean de un tipo de geometría particular.

18.4.2. Crear valores espaciales

Esta sección explica como crear valores espaciales utilizando funciones Well-Known Text y Well-Known Binary que están definidas en el estándar OpenGIS, y utilizando funciones específicas de MySQL.

18.4.2.1. Crear valores geométricos utilizando funciones WKT

MySQL proporciona algunas funciones que toman como parámetros de entrada una representación Well-Known Text y, opcionalmente, un identificador de sistema de referencia espacial (SRID). Estas funciones retornan la geometía correspondiente.

`GeomFromText()` acepta una representación WKT de cualquier tipo de geometría como primer argumento. Una implementación también provee funciones de construcción específicas de cada tipo para la construcción de valores geométricos de cada tipo de geometría.

- `GeomCollFromText(`*`wkt`*`[,`*`srid`*`])` , `GeometryCollectionFromText(`*`wkt`*`[,`*`srid`*`])`

  Construye un valor `GEOMETRYCOLLECTION` utilizando su representación WKT y su SRID.

- `GeomFromText(`*`wkt`*`[,`*`srid`*`])` , `GeometryFromText(`*`wkt`*`[,`*`srid`*`])`

  Construye un valor geométrico de cualquier tipo utilizando su representación WKT y su SRID.

- `LineFromText(`*`wkt`*`[,`*`srid`*`])` , `LineStringFromText(`*`wkt`*`[,`*`srid`*`])`

  Construye un valor `LINESTRING` utilizando su representación WKT y su SRID.

- `MLineFromText(`*`wkt`*`[,`*`srid`*`])` , `MultiLineStringFromText(`*`wkt`*`[,`*`srid`*`])`

  Construye un valor `MULTILINESTRING` utilizando su representación WKT y su SRID.

- `MPointFromText(wkt[,srid])`, `MultiPointFromText(wkt[,srid])`

  Construye un valor `MULTIPOINT` utilizando su representación WKT y su SRID.

- `MPolyFromText(wkt[,srid])`, `MultiPolygonFromText(wkt[,srid])`

  Construye un valor `MULTIPOLYGON` utilizando su representación WKT y su SRID.

- `PointFromText(wkt[,srid])`

  Construye un valor `POINT` utilizando su representación WKT y su SRID.

- `PolyFromText(wkt[,srid])`, `PolygonFromText(wkt[,srid])`

  Construye un valor `POLYGON` utilizando su representación WKT y su SRID.

La especificación OpenGIS también describe funciones opcionales para construir valores `Polygon` o `MultiPolygon` basados en la representación WKT de una colección de anillos o valores `LineString` cerrados. Estos valores pueden interseccionarse. MySQL no implementa estas funciones:

- `BdMPolyFromText(wkt,srid)`

  Construye un valor `MultiPolygon` desde un valor `MultiLineString` en formato WKT conteniendo una colección arbitraria de valores `LineString` cerrados.

- `BdPolyFromText(wkt,srid)`

  Construye un valor `Polygon` desde un valor `MultiLineString` en formato WKT conteniendo una colección arbitraria de valores `LineString` cerrados.

18.4.2.2. Crear valores geométricos utilizando funciones WKB

MySQL provee de varias funciones que toman como parámetros de entrada un `BLOB` que contiene una representación Well-Known Binary y, opcionalmente, un identificador de sistema de referencia espacial (SRID). Éstas retornan la geometría correspondiente.

`GeomFromWKB()` accepts a WKB of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromWKB(wkb[,srid])`, `GeometryCollectionFromWKB(wkb[,srid])`

  Construye un valor `GEOMETRYCOLLECTION` utilizando su representación WKB y su SRID.

- `GeomFromWKB(wkb[,srid])`, `GeometryFromWKB(wkb[,srid])`

Construye un valor geométrico de cualquier tipo utilizando su representación WKB y su SRID.

- `LineFromWKB(wkb[,srid])` , `LineStringFromWKB(wkb[,srid])`

  Construye un valor `LINESTRING` utilizando su representación WKB y su SRID.

- `MLineFromWKB(wkb[,srid])` , `MultiLineStringFromWKB(wkb[,srid])`

  Construye un valor `MULTILINESTRING` utilizando su representación WKB y su SRID.

- `MPointFromWKB(wkb[,srid])` , `MultiPointFromWKB(wkb[,srid])`

  Construye un valor `MULTIPOINT` utilizando su representación WKB y su SRID.

- `MPolyFromWKB(wkb[,srid])` , `MultiPolygonFromWKB(wkb[,srid])`

  Construye un valor `MULTIPOLYGON` utilizando su representación WKB y su SRID.

- `PointFromWKB(wkb[,srid])`

  Construye un valor `POINT` utilizando su representación WKB y su SRID.

- `PolyFromWKB(wkb[,srid])` , `PolygonFromWKB(wkb[,srid])`

  Construye un valor `POLYGON` utilizando su representación WKB y su SRID.

La especificación OpenGIS también describe funciones opcionales para construir valores `Polygon` o `MultiPolygon` basándose en la representación WKB de una colección de anillos o valores `LineString` cerrados. Estos valores puede interseccionarse. MySQL no implementa estas funciones:

- `BdMPolyFromWKB(wkb,srid)`

  Construye un valor `MultiPolygon` desde un valor `MultiLineString` en formato WKB que contiene una colección arbitraria de valores `LineString` cerrados.

- `BdPolyFromWKB(wkb,srid)`

  Construye un valor `Polygon` desde un valor `MultiLineString` en formato WKB que contiene una colección arbitraria de valores `LineString` cerrados.

18.4.2.3. Crear valores geométricos usando funciones específicas de MySQL

Nota: MySQL no implementa las funciones enumeradas en esta sección.

MySQL le provee de un conjunto de funciones útiles para crear representaciones WKB de geometías. Las funciones descritas en esta sección son extensiones de MySQL a la especificación OpenGIS. Los resultados de estas funciones son valores `BLOB` que contienen representaciones WKB de valores geométricos sin SRID. Los resultados de estas funciones pueden ser sustituidos como primer argumento por cualquier función de la familia de funciones `GeomFromWKB()`.

- `GeometryCollection(g1,g2,...)`

  Construye una `GeometryCollection` WKB. Si algún argumento no es una representación WKB bien formada de una geometría, el valor retornado es `NULL`.

- `LineString(pt1,pt2,...)`

  Construye un valor `LineString` WKB desde varios parámetos WKB de tipo `Point`. Si alguno de los argumentos no es un `Point` WKB, el valor retornado es `NULL`. Si el número de parámetros `Point` es menor de dos, el valor retornado es `NULL`.

- `MultiLineString(ls1,ls2,...)`

  Construye un valor `MultiLineString` WKB desde varios parámetos WKB de tipo `LineString`. Si alguno de los argumentos no es un `LineString` WKB, el valor retornado es `NULL`.

- `MultiPoint(pt1,pt2,...)`

  Construye un valor `MultiPoint` WKB desde varios parámetos WKB de tipo `Point`. Si alguno de los argumentos no es un `Point` WKB, el valor retornado es `NULL`.

- `MultiPolygon(poly1,poly2,...)`

  Construye un valor `MultiPolygon` WKB desde varios parámetos WKB de tipo `Polygon`. Si alguno de los argumentos no es un `Polygon` WKB, el valor retornado es `NULL`.

- `Point(x,y)`

  Construye un `Point` WKB utilizando sus coordenadas.

- `Polygon(ls1,ls2,...)`

Construye un valor `Polygon` WKB desde varios parámetros `LineString` WKB. Si alguno de los argumentos no representa el WKB de un `LinearRing` (es decir, un `LineString` que no es cerrado y simple), el valor retornado es `NULL`.

18.4.3. Crear columnas espaciales

MySQL le provee de una manera estándar de crear columnas espaciales para tipos geométricos, por ejemplo, con `CREATE TABLE` o `ALTER TABLE`. Actualmente, las columnas espaciales sólo son soportadas por las tablas `MyISAM`.

- Utilice la sentencia `CREATE TABLE` para crear una tabla con una columna espacial:

```
mysql> CREATE TABLE geom (g GEOMETRY);
Query OK, 0 rows affected (0.02 sec)
```

- Utilice la sentencia `ALTER TABLE` para añadir o eliminar una columna espacial a o de una tabla ya existente:

```
mysql> ALTER TABLE geom ADD pt POINT;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> ALTER TABLE geom DROP pt;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

18.4.4. Poblar columnas espaciales

Tras haber creado columnas espaciales, puede poblarlas con datos espaciales.

Los valores deben ser almacenados en formato de geometría internos, pero usted puede convertirlos a ese formato ya sea desde formato Well-Known Text (WKT) o desde formato Well-Known Binary (WKB). Los siguientes ejemplos demuestran cómo insertar valores geométricos en una tabla convirtiendo valores WKT al formato interno de geometrías.

Puede realizar la conversión directamente en la sentencia `INSERT`:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

O puede realizar la conversión previamente al `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

Los siguientes ejemplos insertan más geometrías complejas en la tabla:

```
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));
```

Los ejemplos precedentes utilizan todos `GeomFromText()` para crear valores geométricos. También puede utilizar funciones específicas de tipo:

```
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));
```

Tenga en cuenta que si un programa de aplicación cliente quiere utilizar representaciones WKB de valores geométricos, es responsable de enviar dichas representaciones correctamente formadas en las consultas al servidor. De cualquier modo, hay diversas maneras de satisfacer este requerimiento. Por ejemplo:

- Insertar un valor `POINT(1 1)` con sintaxis hexadecimal literal:

  ```
  mysql> INSERT INTO geom VALUES
      -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
  ```

- Una aplicación ODBC puede enviar una representación WKB, encapsulándola en un comodín utilizando un argumento de tipo `BLOB`:

  ```
  INSERT INTO geom VALUES (GeomFromWKB(?))
  ```

  Otras interfaces de programación pueden soportar un sistema de comodines similar.

- En un programa en C, puede marcar un valor binario utilizando `mysql_real_escape_string()` e incluir el resultado en una consulta que se envía al servidor. Consulte Sección 24.2.3.48, “mysql_real_escape_string()”.

18.4.5. Extraer datos espaciales

Los valores geométricos almacenados en una tabla pueden ser extraídos en formato interno. Además puede también convertirlos al formato WKT o al WKB.

18.4.5.1. Extraer datos espaciales en formato interno

Extraer valroes geométricos utilizando el formato interno puede ser útil en transferencias de tabla a tabla:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

18.4.5.2. Extraer datos espaciales en formato WKT

La función`AsText()` convierte una geometría desde el formato interno a una cadena WKT.

```
mysql> SELECT AsText(g) FROM geom;
```

```
+-------------------------+
| AsText(p1)              |
+-------------------------+
| POINT(1 1)              |
| LINESTRING(0 0,1 1,2 2) |
+-------------------------+
```

18.4.5.3. Recoger datos espaciales en formato WKB

La función `AsBinary()` convierte una geometría desde el formato interno a un `BLOB` que contiene el valor WKB.

```
SELECT AsBinary(g) FROM geom;
```

18.5. Analizar información espacial

Tras haber poblado las columnas espaciales con valores, usted está listo para consultarlos y analizarlos. MySQL provee de una serie de funciones para realizar diversas operaciones sobre datos espaciales. Estas funciones pueden ser agrupadas en cuatro categorías principales de acuerdo con el tipo de operación que realizan:

- Funciones que convierten las geometrías a diversos formatos
- Funciones que proveen de acceso a propiedades cuantitativas o cualitativas de una geometría
- Funciones que describen relaciones entre dos geometrías
- Funciones que crean nuevas geometrías desde otras ya existentes

Las funciones de análisis espacial pueden ser utilizadas en muchos y muy diferentes contextos, tales como:

- Cualquier programa SQL interactivo, como `mysql` o `MySQLCC`
- Programas de aplicación escritos en cualquier lenguaje que soporte una API cliente de MySQL

18.5.1. Funciones de conversión de formato geométrico

MySQL soporta las siguientes funciones para convertir valores geométricos entre formato interno y los formatos WKT o WKB:

- `AsBinary(g)`

  Convierte un valor en formato interno a su representación WKB y devuelve el resultado binario.

- `AsText(g)`

  Convierte un valor en formato interno a su representación WKT y devuelve la cadena resultante.

```
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)  |
+--------------------------+
```

- `GeomFromText(wkt[,srid])`

  Convierte un valor de texto desde su representación WKT al formato interno de geometría y retorna el resultado. También hay varias funciones específicas de cada tipo que están disponibles, tales como `PointFromText()` y `LineFromText()`; consulte Sección 18.4.2.1, “Crear valores geométricos utilizando funciones WKT”.

- `GeomFromWKB(wkb[,srid])`

  Convierte un valor binario desde su representación WKB al formato interno de geometría y retorna el resultado. También hay varias funciones específicas de cada tipo que están disponibles, tales como `PointFromWKB()` y `LineFromWKB()`; consulte Sección 18.4.2.2, “Crear valores geométricos utilizando funciones WKB”.

18.5.2. Funciones `Geometry`

Cada función que pertenece a este grupo toma un valor geométrico como su primer argumento y retorna alguna propiedad cuantitativa o cualitativa de la geometría. Algunas funciones restringen el tipo de sus argumentos. Dichas funciones retornan `NULL` si el argumento es de un tipo geométrico incorrecto. Por ejemplo, `Area()` retorna `NULL` si el tipo del objeto no es `Polygon` ni `MultiPolygon`.

18.5.2.1. Funciones generales de geometría

Las funciones enumeradas en esta sección no restringen el tipo de sus argumentos y aceptan cualquier tipo de valor geométrico.

- `Dimension(g)`

  Retorna la dimensión inherente al valor geométrico *g*. El resultado puede ser -1, 0, 1, o 2 (El significado de estos valores se explica en Sección 18.2.2, “La clase `Geometry`”.)

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

- `Envelope(g)`

  Retorna el rectángulo mínimo que circunscribe (Minimum Bounding Rectangle (MBR)) el valor geométrico *g*. El resultado que se retorna es de tipo `Polygon`.

```
```

```
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))                      |
+-----------------------------------------------------+
```

El polígono está definido por los puntos de la esquina de la caja que lo circunscribe:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- `GeometryType(g)`

  Retorna en una cadena el nombre del tipo de la geometría de la que la instancia *g* es miembro. El nombre corresponde a una de las subclases instanciables de `Geometry`.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+---------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+---------------------------------------+
| POINT                                 |
+---------------------------------------+
```

- `SRID(g)`

  Retorna un entero que indica el Identificador de Sistema de Referencia Espacial del valor geométrico *g*.

  En MySQL, el valor SRID es simplemente un entero asociado con el valor geometrico. Todos los cálculos se realizan asumiendo una geometría Euclídea (planar).

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+---------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+---------------------------------------------+
|                                         101 |
+---------------------------------------------+
```

La especificación OpenGIS también define las siguientes funciones, que MySQL no implementa:

- `Boundary(g)`

  Retorna una geometría que es el cierre del límite combinacional del valor geométrico *g*.

- `IsEmpty(g)`

  Retorna 1 si el valor geométrico *g* corresponde a la geometría vacía, 0 si no está vacía, y -1 si el argumento es `NULL`. Si la geometría está vacía, representa el conjunto de puntos vacío.

- `IsSimple(g)`

  Actualmente esta función es un comodín y no debería ser utilizada. Si se implementara, su comportamiento será como el que se detalla en el siguiente párrafo.

  Retorna 1 si el valor geométrico *g* no tiene puntos geométricos anómalos, tales como auto-intersección o auto-tangencia. `IsSimple()` retorna 0 si el argumento no es simple, y -1 si es `NULL`.

  La descripción de cada clase geométrica instanciable mencionada anteriormente en este capítulo incluye las condiciones específicas que provcan que una instancia de una clase sea clasificada como no simple.

18.5.2.2. Funciones `Point`

Un `Point` consiste en sus coordenadas X e Y, que pueden ser obtenidas utilizando las siguientes funciones:

- `X(p)`

  Retorna el valor de la coordenada X del punto *p* como un número de doble precisión.

```
mysql> SELECT X(GeomFromText('Point(56.7 53.34)'));
+-------------------------------------+
| X(GeomFromText('Point(56.7 53.34)')) |
+-------------------------------------+
|                                56.7 |
+-------------------------------------+
```

- `Y(p)`

  Retorna el valor de la coordenada Y del punto *p* como un número de doble precisión.

```
mysql> SELECT Y(GeomFromText('Point(56.7 53.34)'));
+-------------------------------------+
| Y(GeomFromText('Point(56.7 53.34)')) |
+-------------------------------------+
|                               53.34 |
+-------------------------------------+
```

18.5.2.3. Funciones `LineString`

Un `LineString` se compone de valores `Point`. Puede extraer valores particulares de dentro de una `LineString`, contar los puntos que contiene u obtener su longitud.

- `EndPoint(ls)`

  Devuelve el `Point` que es el punto final del valor `LineString` *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
| POINT(3 3)                          |
+-------------------------------------+
```

- `GLength(ls)`

  Devuelve la longitud del valor `LineString ls` como un número de doble precisión en su sistema de referencia espacial asociado.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+----------------------------+
| GLength(GeomFromText(@ls)) |
+----------------------------+
|           2.8284271247462 |
+----------------------------+
```

- `IsClosed(ls)`

  Retorna 1 si el valor `LineString ls` es cerrado (es decir, su punto inicial `StartPoint()` y punto final `EndPoint()` tienen el mismo valor). Retorna 0 si `ls` es no cerrado, y -1 si es `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT IsClosed(GeomFromText(@ls));
+----------------------------+
| IsClosed(GeomFromText(@ls)) |
+----------------------------+
|                          0 |
+----------------------------+
```

- `NumPoints(ls)`

  Retorna el número de puntos en el valor `LineString ls`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----------------------------+
| NumPoints(GeomFromText(@ls)) |
+-----------------------------+
|                           3 |
+-----------------------------+
```

- `PointN(ls,n)`

  Retorna el punto e-*n*-ésimo en el valor `Linestring` *`ls`*. Los números de punto comienzan por 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)`

  Retorna el `Point` que es el punto inicial del valor `LineString` *`ls`*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+---------------------------------------+
| AsText(StartPoint(GeomFromText(@ls))) |
+---------------------------------------+
| POINT(1 1)                            |
+---------------------------------------+
```

La especificación OpenGIS también define la siguiente función, que MySQL no implementa:

- `IsRing(ls)`

  Retorna 1 si el valor `LineString` *`ls`* es cerrado (es decir, que sus puntos inicial y final tienen el mismo valor) y es simple (no pasa a través del mismo punto más de una vez). Retorna 0 si *`ls`* no es un anillo, y -1 si es `NULL`.

18.5.2.4. Funciones `MultiLineString`

- `GLength(mls)`

  Retorna la longitud del valor `MultiLineString` *`mls`* como un entero de doble precisión. La longitud de *`mls`* es igual a la suma de las longitudes de sus elementos.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----------------------------+
| GLength(GeomFromText(@mls)) |
+-----------------------------+
|             4.2426406871193 |
+-----------------------------+
```

- `IsClosed(mls)`

  Retorna 1 si el valor `MultiLineString` *`mls`* es cerrado (es decir, los valores del punto inicial y el punto final de cada `LineString` en `mls` son iguales entre sí). Retorna 0 si *`mls`* es no cerrado, y -1 si es `NULL`.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+-----------------------------+
| IsClosed(GeomFromText(@mls)) |
+-----------------------------+
|                           0 |
+-----------------------------+
```

18.5.2.5. Funciones Polygon

- Area(*poly*)

  Retorna, como número de doble precisión, el área del valor Polygon *poly*, medido en su sistema de referencia espacial.

```
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 |
+---------------------------+
```

- ExteriorRing(*poly*)

  Retorna el anillo exterior del valor Polygon *poly* como un 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*)

  Retorna el e-*n*-ésimo anillo interior del valor Polygon *poly* como un LineString. Los números de anillo comienzan en 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)`

  Retorna el número de anillos interiores en el valor `Polygon` *`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 |
+-------------------------------------+
```

18.5.2.6. Funciones `MultiPolygon`

- `Area(mpoly)`

  Retorna, como un número de doble precisión, el área del valor `MultiPolygon` *`mpoly`*, medido en su sistema de referencia espacial.

```
mysql> SET @mpoly =
    -> '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 |
+----------------------------+
```

La especificación OpenGIS también define las siguientes funciones, que MySQL no implementa:

- `Centroid(mpoly)`

  Retorna el centroide matemático del valor `MultiPolygon` *`mpoly`* como un `Point`. Este resultado no está garantizado que esté contenido en el `MultiPolygon`.

- `PointOnSurface(mpoly)`

  Retorna un valor `Point` del que se garantiza que está en el valor del `MultiPolygon` *`mpoly`*.

18.5.2.7. Funciones `GeometryCollection`

- `GeometryN(gc,n)`

  Retorna la e-*n*-ésima geometría en el valor `GeometryCollection` *`gc`*. Los números de geometría comienzan por 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)`

  Retorna el número de geometrías contenidas en la `GeometryCollection` *`gc`*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+---------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+---------------------------------+
|                               2 |
+---------------------------------+
```

18.5.3. Funciones que crean nuevas geometrías a partir de unas existentes

18.5.3.1. Funciones geométricas que producen nuevas geometrías

En la sección Sección 18.5.2, “Funciones `Geometry`”, hemos revisado algunas funciones que pueden construir nuevas geometrías a partir de otras ya existentes:

- `Envelope(g)`
- `StartPoint(ls)`
- `EndPoint(ls)`
- `PointN(ls,n)`
- `ExteriorRing(poly)`
- `InteriorRingN(poly,n)`
- `GeometryN(gc,n)`

18.5.3.2. Operadores espaciales

OpenGIS propone varias funciones adicionales que pueden producir geometrías. Están diseñadas para implementar operadores espaciales.

Estas funciones no están implementadas en MySQL. Puede ser que aparezcan en futuras versiones.

- `Buffer(g,d)`

  Retorna una geometría que representa todos los puntos cuya distancia hasta el valor geométrico *`g`* es menor o igual a la distancia *`d`*.

- `ConvexHull(g)`

  Retorna una geometría que representa el borde convexo del valor geométrico *`g`*.

- `Difference(g1,g2)`

  Retorna una geometría que representa el conjunto de puntos resultado de la resta de los valores geométricos *g1* y *g2*.

- `Intersection(g1,g2)`

  Retorna una geometría que representa el conjunto de puntos resultado de la intersección de los valores geométricos *g1* y *g2*.

- `SymDifference(g1,g2)`

  Retorna una geometría que representa el conjunto de puntos resultado de la resta simétrica de los valores geométricos *g1* y *g2*.

- `Union(g1,g2)`

  Retorna una geometría que representa el conjunto de puntos resultado de la unión de los valores geométricos *g1* y *g2*.

18.5.4. Funciones para probar relaciones espaciales entre objetos geométricos

Las funciones descritas en estas secciones toman dos geometrías como parámetros de entrada y retornan una relación cuantitativa o cualitativa entre ellas.

18.5.5. Relaciones entre rectángulos MBR (Minimal Bounding Rectangles)

MySQL le provee con algunas funciones que pueden comprobar relaciones entre los rectángulos mínimos que circunscriben a dos geometrías `g1` y `g2`. Entre ellas se incluyen:

- `MBRContains(g1,g2)`

  Retorna 1 o 0 para indicar si el rectángulo mínimo que circunscribe a *g1* contiene, o no, al rectángulo mínimo que circunscribe a *g2*.

```
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), MBRContains(@g2,@g1);
----------------------+----------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

- `MBRDisjoint(g1,g2)`

  Retorna 1 o 0 para indicar si los rectángulos mínimos que circunscriben a las geometrías *`g1`* y *`g2`* son disjuntas (no se interseccionan), o no.

- `MBREqual(g1,g2)`

  Retorna 1 o 0 para indicar si el rectángulo mínimo que circunscribe a las dos geometrías *`g1`* y *`g2`* es o no es el mismo.

- `MBRIntersects(g1,g2)`

  Retorna 1 o 0 para indicar si los rectángulos mínimos que circunscriben a las geometrías *`g1`* y *`g2`* se interseccionan o no.

- `MBROverlaps(g1,g2)`

  Retorna 1 o 0 para indicar si los rectángulos mínimos que circunscriben a las geometrías *`g1`* y *`g2`* se sobreponen o no.

- `MBRTouches(g1,g2)`

  Retorna 1 o 0 para indicar si los rectángulos mínimos que circunscriben a las geometrías *`g1`* y *`g2`* se tocan o no.

- `MBRWithin(g1,g2)`

  Retorna 1 o 0 para indicar si el rectángulo mínimo que circunscribe a la geometría *`g1`* se encuentra o no dentro del rectángulo mínimo que circunscribe a la geometría *`g2`*.

```
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))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+--------------------+--------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

18.5.6. Funciones que prueban relaciones espaciales entre geometrías

La especificación OpenGIS define las siguientes funciones. Actualmente, MySQL no las implementa de acuerdo a la especificación. Aquéllas que están implementadas retornan el mismo resultado que las

funciones basadas en MBR correspondientes. Esto incluye a las funciones de la siguiente lista, además de `Distance()` y `Related()`.

Estas funciones pueden ser implementadas en futuras versiones con soporte completo para análisis espacial, y no sólo soporte basado en MBR.

Estas funciones operan sobre dos valores geométricos `g1` and `g2`.

- `Contains(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* contiene completamente o no a *`g2`*.

- `Crosses(g1,g2)`

  Retorna 1 si *`g1`* cruza espacialmente a *`g2`*. Retorna `NULL` si `g1` es un `Polygon` o un `MultiPolygon`, o si *`g2`* es un `Point` o un `MultiPoint`. En cualquier otro caso, retorna 0.

  El término *cruza espacialmente* denota una relación espacial entre dos geometrías dadas que tiene las siguientes propiedades:

  - Las dos geometrías se interseccionan
  - La intersección resulta en una geometría que tiene una dimensión que es una unidad menor que la dimensión máxima de las dos geometrías dadas
  - Su intersección no es igual a ninguna de las dos geometrías dadas.

- `Disjoint(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* es o no espacialmente disjunto (no intersecciona) con *`g2`*.

- `Distance(g1,g2)`

  Retorna un número de doble precisión que representa la distancia más corta entre los puntos de ambas geometrías.

- `Equals(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* es o no igual espacialmente a *`g2`*.

- `Intersects(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* intersecciona espacialmente con *`g2`*.

- `Overlaps(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* se superpone espacialmente o no a *`g2`*. El término *superpone espacialmente* se utiliza si dos geometrías interseccionan y la intersección resultante es una geometría de las mismas dimensiones pero no igual a ninguna de las geometrías dadas.

- `Related(g1,g2,pattern_matrix)`

  Retorna 1 o 0 paa indicar si la relación espacial especificada por *`pattern_matrix`* existe entre *`g1`* y *`g2`*. Retorna -1 si los argumentos son `NULL`. La matriz de patrones (pattern_matrix) es una cadena. Su especificación se explicará aquí si la función llega a ser implementada.

- `Touches(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* toca espacialmente o no a *`g2`*. Dos geometrías se *tocan espacialmente* si los interiores de las dos geometrías no interseccionan, pero el límite de una de ellas intersecciona con el límite o el interior de la otra.

- `Within(g1,g2)`

  Retorna 1 o 0 para indicar si *`g1`* está o no espacialmente dentro de *`g2`*.

18.6. Optimización del análisis espacial

Las operaciones de búsqueda en bases de datos no espaciales pueden optimizarse mediante índices. Esto también es aplicable a las bases de datos espaciales. Con la ayuda de una gran variedad de métodos de indexación multidimensional que han sido previamente designados, es posible optimizar las búsquedas espaciales. Los más usuales de estos métodos son:

- Consultas sobre puntos que buscan todos los objetos que contienen un punto dado
- Consultas sobre regiones que buscan todos los objetos que se superponen a una región dada

MySQL utiliza **Árboles R con partición cuadrática** para indexar las columnas espaciales. Un índice espacial se construye utilizando la MBR de una geometría. Para la mayoría de las geometrías, la MBR es el rectángulo mínimo que la circunscribe. Para una línea vertical u horizontal, la MBR es un rectángulo reducido a una línea. Para un punto, la MBR es un rectángulo reducido a un punto.

18.6.1. Crear índices espaciales

MySQL puede crear índices espaciales utilizando una sintaxis similar a la que se utiliza para crear índices normales, pero extendida con la palabra clave `SPATIAL`. Las columnas espaciales que están indexadas, deben ser declaradas, actualmente, como `NOT NULL`. Los siguientes ejemplos demuestran cómo crear índices espaciales.

- Con `CREATE TABLE`:

```
mysql> CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- Con `ALTER TABLE`:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- Con `CREATE INDEX`:

```
mysql> CREATE SPATIAL INDEX sp_index ON geom (g);
```

Para eliminar índices espaciales, utilice `ALTER TABLE` o `DROP INDEX`:

- Con `ALTER TABLE`:

```
mysql> ALTER TABLE geom DROP INDEX g;
```

- Con `DROP INDEX`:

```
mysql> DROP INDEX sp_index ON geom;
```

Ejemplo: Suponga una tabla `geom` que contiene más de 32000 geometrías, que están almacenadas en la columna `g` del tipo `GEOMETRY`. La tabla también tiene una columna `AUTO_INCREMENT` llamada `fid` para almacenar valores de ID de objetos.

```
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 |
+----------+
1 row in set (0.00 sec)
```

Para añadir un índice espacial en la columna `g`, utilice esta sentencia:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

18.6.2. Usar un índice espacial

El optimizador investiga si los índices espaciales disponibles pueden ser utilizados en la búsqueda de consultas que utilicen una función como `MBRContains()` o `MBRWithin()` en la cláusula `WHERE`. Por ejemplo, digamos que queremos encontrar todos los objetos que están en un rectángulo determinado:

```
mysql> SELECT fid,AsText(g) FROM geom WHERE
```

```
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g
+-----+--------------------------------------------------------------------------+
| fid | AsText(g)                                                                |
+-----+--------------------------------------------------------------------------+
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)  |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)  |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)  |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)  |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2) |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)  |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4) |
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2) |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)  |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)        |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)    |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2) |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)      |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4) |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)  |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8) |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)    |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)      |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)    |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)  |
+-----+--------------------------------------------------------------------------+
20 rows in set (0.00 sec)
```

Utilicemos `EXPLAIN` para comprobar en qué manera se ejecuta esta consulta (la columna `id` ha sido eliminada para que el listado quede mejor maquetado en la página):

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g
+-------------+-------+-------+---------------+------+---------+------+------+-------------+
| select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra       |
+-------------+-------+-------+---------------+------+---------+------+------+-------------+
| SIMPLE      | geom  | range | g             | g    |      32 | NULL |   50 | Using where |
+-------------+-------+-------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

Comprobemos qué ocurriría sin un índice espacial:

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g
+-------------+-------+------+---------------+------+---------+------+-------+-------------+
| select_type | table | type | possible_keys | key  | key_len | ref  | rows  | Extra       |
+-------------+-------+------+---------------+------+---------+------+-------+-------------+
| SIMPLE      | geom  | ALL  | NULL          | NULL |    NULL | NULL | 32376 | Using where |
+-------------+-------+------+---------------+------+---------+------+-------+-------------+
1 row in set (0.00 sec)
```

Ejecutemos la sentencia `SELECT` ignorando la clave espacial que tenemos:

```
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g
+-----+--------------------------------------------------------------------------+
| fid | AsText(g)                                                                |
+-----+--------------------------------------------------------------------------+
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2) |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)  |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)        |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)    |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2) |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)      |
```

```
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)   |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)     |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)   |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)       |
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)     |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)     |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)     |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)     |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2) |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)     |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)         |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)       |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)     |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4) |
+-----+-----------------------------------------------------------------------+
20 rows in set (0.46 sec)
```

Cuando el índice no se utiliza, el tiempo de ejecución de esta consulta crece de 0.00 segundos a 0.46 segundos.

En versiones futuras, los índice espaciales podrán también ser utilizados para optimizar otras funciones. Consulte Sección 18.5.4, “Funciones para probar relaciones espaciales entre objetos geométricos”.

18.7. Conformidad y compatibilidad de MySQL

18.7.1. Características GIS que todavía no han sido implementadas

- Vistas adicionales de Metadatos

  La especificación OpenGIS propone diversas vistas adicionales de metadatos. Por ejemplo, una vista de sistema llamada `GEOMETRY_COLUMNS` contiene una descripción de las columnas geométricas, una fila por cada columna geométrica en la base de datos.

- La función OpenGIS `Length()` sobre `LineString` y `MultiLineString` actualmente debe llamarse en MySQL como `GLength()`

  El problema es que existe una función SQL `Length()` que calcula la longitud de las cadenas de caracteres, y a veces no es posible distinguir cuándo la función ha sido llamada en un contexto textual o espacial. Necesitamos resolver esto de alguna manera, o decidirnos por cambiar el nombre de la función.

Capítulo 19. Procedimientos almacenados y funciones

Tabla de contenidos

Los procedimientos almacenados y funciones son nuevas funcionalidades de la versión de MySQL 5.0. Un procedimiento almacenado es un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan relanzar los comandos individuales pero pueden en su lugar referirse al procedimiento almacenado.

Algunas situaciones en que los procedimientos almacenados pueden ser particularmente útiles:

- Cuando múltiples aplicaciones cliente se escriben en distintos lenguajes o funcionan en distintas plataformas, pero necesitan realizar la misma operación en la base de datos.
- Cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos almacenados para todas las operaciones comunes. Esto proporciona un entorno seguro y consistente, y los procedimientos pueden asegurar que cada operación se loguea apropiadamente. En tal entorno, las aplicaciones y los usuarios no obtendrían ningún acceso directo a las tablas de la base de datos, sólo pueden ejectuar algunos procedimientos almacenados.

Los procedimientos almacenados pueden mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. El intercambio que hay es que aumenta la carga del servidor de la base de datos ya que la mayoría del trabajo se realiza en la parte del servidor y no en el cliente. Considere esto si muchas máquinas cliente (como servidores Web) se sirven a sólo uno o pocos servidores de bases de datos.

Los procedimientos almacenados le permiten tener bibliotecas o funciones en el servidor de base de datos. Esta característica es compartida por los lenguajes de programación modernos que permiten este diseño interno, por ejemplo, usando clases. Usando estas características del lenguaje de programación cliente es beneficioso para el programador incluso fuera del entorno de la base de datos.

MySQL sigue la sintaxis SQL:2003 para procedimientos almacenados, que también usa IBM DB2.

La implementación de MySQL de procedimientos almacenados está en progreso. Toda la sintaxis descrita en este capítulo se soporta y cualquier limitación y extensión se documenta apropiadamente. Más

discusión o restricciones de uso de procedimientos almacenados se da en Apéndice H, *Restricciones en características de MySQL*.

Logueo binario para procedimientos almacenados se hace como se describe en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

19.1. Procedimientos almacenados y las tablas de permisos

Los procedimientos almacenados requieren la tabla `proc` en la base de datos `mysql`. Esta tabla se crea durante la instalación de MySQL 5.0. Si está actualizando a MySQL 5.0 desde una versión anterior, asegúrese de actualizar sus tablas de permisos para asegurar que la tabla `proc` existe. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Desde MySQL 5.0.3, el sistema de permisos se ha modificado para tener en cuenta los procedimientos almacenados como sigue:

- El permiso `CREATE ROUTINE` se necesita para crear procedimientos almacenados.
- El permiso `ALTER ROUTINE` se necesita para alterar o borrar procedimientos almacenados. Este permiso se da automáticamente al creador de una rutina.
- El permiso `EXECUTE` se requiere para ejectuar procedimientos almacenados. Sin embargo, este permiso se da automáticamente al creador de la rutina. También, la característica `SQL SECURITY` por defecto para una rutina es `DEFINER`, lo que permite a los usuarios que tienen acceso a la base de datos ejecutar la rutina asociada.

19.2. Sintaxis de procedimientos almacenados

Los procedimientos almacenados y rutinas se crean con comandos `CREATE PROCEDURE` y `CREATE FUNCTION` . Una rutina es un procedimiento o una función. Un procedimiento se invoca usando un comando `CALL` , y sólo puede pasar valores usando variables de salida. Una función puede llamarse desde dentro de un comando como cualquier otra función (esto es, invocando el nombre de la función), y puede retornar un valor escalar. Las rutinas almacenadas pueden llamar otras rutinas almacenadas.

Desde MySQL 5.0.1, los procedimientos almacenadoso funciones se asocian con una base de datos. Esto tiene varias implicaciones:

- Cunado se invoca la rutina, se realiza implícitamente `USE` *`db_name`* (y se deshace cuando acaba la rutina). Los comandos `USE` dentro de procedimientos almacenados no se permiten.
- Puede calificar los nombres de rutina con el nombre de la base de datos. Esto puede usarse para referirse a una rutina que no esté en la base de datos actual. Por ejemplo, para invocar procedimientos almacenados `p` o funciones `f` esto se asocia con la base de datos `test` , puede decir `CALL test.p()` o `test.f()`.
- Cuando se borra una base de datos, todos los procedimientos almacenados asociados con ella también se borran.

(En MySQL 5.0.0, los procedimientos almacenados son globales y no asociados con una base de datos. Heredan la base de datos por defecto del llamador. Si se ejecuta `USE` *`db_name`* desde la rutina, la base de datos por defecto original se restaura a la salida de la rutina.)

MySQL soporta la extensión muy útil que permite el uso de comandos regulares `SELECT` (esto es, sin usar cursores o variables locales) dentro de los procedimientos almacenados. El conjunto de resultados de estas consultas se envía diractamente al cliente. Comandos `SELECT` múltiples generan varios conjuntos de resultados, así que el cliente debe usar una biblioteca cliente de MySQL que soporte conjuntos

de resultados múltiples. Esto significa que el cliente debe usar una biblioteca cliente de MySQL como mínimos desde 4.1.

La siguiente sección describe la sintaxis usada para crear, alterar, borrar, y consultar procedimientos almacenados y funciones.

19.2.1. CREATE PROCEDURE y CREATE FUNCTION

```
CREATE PROCEDURE sp_name ([parameter[,...]])
    [characteristic ...] routine_body

CREATE FUNCTION sp_name ([parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MySQL data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    procedimientos almacenados o comandos SQL válidos
```

Estos comandos crean una rutina almacenada. Desde MySQL 5.0.3, para crear una rutina, es necesario tener el permiso `CREATE ROUTINE` , y los permisos `ALTER ROUTINE` y `EXECUTE` se asignan automáticamente a su creador. Si se permite logueo binario necesita también el permisos `SUPER` como se describe en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

Por defecto, la rutina se asocia con la base de datos actual. Para asociar la rutina explícitamente con una base de datos, especifique el nombre como *`db_name.sp_name`* al crearlo.

Si el nombre de rutina es el mismo que el nombre de una función de SQL, necesita usar un espacio entre el nombre y el siguiente paréntesis al definir la rutina, o hay un error de sintaxis. Esto también es cierto cuando invoca la rutina posteriormente.

La cláusula `RETURNS` puede especificarse sólo con `FUNCTION`, donde es obligatorio. Se usa para indicar el tipo de retorno de la función, y el cuerpo de la función debe contener un comando `RETURN value`.

La lista de parámetros entre paréntesis debe estar siempre presente. Si no hay parámetros, se debe usar una lista de parámetros vacía `()` . Cada parámetro es un parámetro `IN` por defecto. Para especificar otro tipo de parámetro, use la palabra clave `OUT` o `INOUT` antes del nombre del parámetro. Especificando `IN`, `OUT`, o `INOUT` sólo es valido para una `PROCEDURE`.

El comando `CREATE FUNCTION` se usa en versiones anteriores de MySQL para soportar UDFs (User Defined Functions) (Funciones Definidas por el Usuario). Consulte Sección 27.2, “Añadir nuevas funciones a MySQL”. UDFs se soportan, incluso con la existencia de procedimientos almacenados. Un UDF puede tratarse como una función almacenada externa. Sin embargo, tenga en cuenta que los procedimientos almacenados comparten su espacio de nombres con UDFs.

Un marco para procedimientos almacenados externos se introducirá en el futuro. Esto permitira escribir procedimientos almacenados en lenguajes distintos a SQL. Uno de los primeros lenguajes a soportar

será PHP ya que el motor central de PHP es pequeño, con flujos seguros y puede empotrarse fácilmente. Como el marco es público, se espera soportar muchos otros lenguajes.

Un procedimiento o función se considera “determinista” si siempre produce el mismo resultado para los mismos parámetros de entrada, y “no determinista” en cualquier otro caso. Si no se da ni `DETERMINISTIC` ni `NOT DETERMINISTIC` por defecto es `NOT DETERMINISTIC`.

Para replicación, use la función `NOW()` (o su sinónimo) o `RAND()` no hace una rutina no determinista necesariamente. Para `NOW()`, el log binario incluye el tiempo y hora y replica correctamente. `RAND()` también replica correctamente mientras se invoque sólo una vez dentro de una rutina. (Puede considerar el tiempo y hora de ejecución de la rutina y una semilla de número aleatorio como entradas implícitas que son idénticas en el maestro y el esclavo.)

Actualmente, la característica `DETERMINISTIC` se acepta, pero no la usa el optimizador. Sin embargo, si se permite el logueo binario, esta característica afecta si MySQL acepta definición de rutinas. Consulte Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

Varias características proporcionan información sobre la naturaleza de los datos usados por la rutina. `CONTAINS SQL` indica que la rutina no contiene comandos que leen o escriben datos. `NO SQL` indica que la rutina no contiene comandos SQL . `READS SQL DATA` indica que la rutina contiene comandos que leen datos, pero no comandos que escriben datos. `MODIFIES SQL DATA` indica que la rutina contiene comandos que pueden escribir datos. `CONTAINS SQL` es el valor por defecto si no se dan explícitamente ninguna de estas características.

La característica `SQL SECURITY` puede usarse para especificar si la rutina debe ser ejecutada usando los permisos del usuario que crea la rutina o el usuario que la invoca. El valor por defecto es `DEFINER`. Esta característica es nueva en SQL:2003. El creador o el invocador deben tener permisos para acceder a la base de datos con la que la rutina está asociada. Desde MySQL 5.0.3, es necesario tener el permiso `EXECUTE` para ser capaz de ejecutar la rutina. El usuario que debe tener este permiso es el definidor o el invocador, en función de cómo la característica `SQL SECURITY` .

MySQL almacena la variable de sistema `sql_mode` que está en efecto cuando se crea la rutina, y siempre ejecuta la rutina con esta inicialización.

La cláusula `COMMENT` es una extensión de MySQL, y puede usarse para describir el procedimiento almacenado. Esta información se muestra con los comandos `SHOW CREATE PROCEDURE` y `SHOW CREATE FUNCTION` .

MySQL permite a las rutinas que contengan comandos DDL (tales como `CREATE` y `DROP`) y comandos de transacción SQL (como `COMMIT`). Esto no lo requiere el estándar, y por lo tanto, es específico de la implementación.

Los procedimientos almacenados no pueden usar `LOAD DATA INFILE`.

Nota: Actualmente, los procedimientos almacenados creados con `CREATE FUNCTION` no pueden tener referencias a tablas. (Esto puede incluir algunos comandos `SET` que pueden contener referencias a tablas, por ejemplo `SET a:= (SELECT MAX(id) FROM t)`, y por otra parte no pueden contener comandos `SELECT` , por ejemplo `SELECT 'Hello world!' INTO var1`.) Esta limitación se elminará en breve.

Los comandos que retornan un conjunto de resultados no pueden usarse desde una función almacenada. Esto incluye comandos `SELECT` que no usan `INTO` para tratar valores de columnas en variables, comandos `SHOW` y otros comandos como `EXPLAIN`. Para comandos que pueden determinarse al definir la función para que retornen un conjunto de resultados, aparece un mensaje de error `Not allowed to return a result set from a function` (`ER_SP_NO_RETSET_IN_FUNC`). Para comandos que puede determinarse sólo en tiempo de ejecución si retornan un conjunto de resultados, aparece el error `PROCEDURE %s can't return a result set in the given context` (`ER_SP_BADSELECT`).

El siguiente es un ejemplo de un procedimiento almacenado que use un parámetro OUT . El ejemplo usa el cliente mysql y el comando delimiter para cambiar el delimitador del comando de ; a // mientras se define el procedimiento . Esto permite pasar el delimitador ; usado en el cuerpo del procedimiento a través del servidor en lugar de ser interpretado por el mismo mysql.

```
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)
```

Al usar el comando delimiter, debe evitar el uso de la antibarra ('\') ya que es el carácter de escape de MySQL.

El siguiente es un ejemplo de función que toma un parámetro, realiza una operación con una función SQL, y retorna el resultado:

```
mysql> delimiter //

mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
    -> RETURN CONCAT('Hello, ',s,'!');
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

Si el comando RETURN en un procedimiento almacenado retorna un valor con un tipo distinto al especificado en la cláusula RETURNS de la función, el valor de retorno se coherciona al tipo apropiado. Por ejemplo, si una función retorna un valor ENUM o SET, pero el comando RETURN retorna un entero, el valor retornado por la función es la cadena para el miembro de ENUM correspondiente de un conjunto de miembros SET .

19.2.2. ALTER PROCEDURE y ALTER FUNCTION

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]
```

```
characteristic:
    { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

Este comando puede usarse para cambiar las características de un procedimiento o función almacenada. Debe tener el permiso `ALTER ROUTINE` para la rutina desde MySQL 5.0.3. El permiso se otorga automáticamente al creador de la rutina. Si está activado el logueo binario, necesitará el permiso `SUPER` , como se describe en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

Pueden especificarse varios cambios con `ALTER PROCEDURE` o `ALTER FUNCTION`.

19.2.3. DROP PROCEDURE y DROP FUNCTION

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

Este comando se usa para borrar un procedimiento o función almacenado. Esto es, la rutina especificada se borra del servidor. Debe tener el permiso `ALTER ROUTINE` para las rutinas desde MySQL 5.0.3. Este permiso se otorga automáticamente al creador de la rutina.

La cláusula `IF EXISTS` es una extensión de MySQL . Evita que ocurra un error si la función o procedimiento no existe. Se genera una advertencia que puede verse con `SHOW WARNINGS`.

19.2.4. SHOW CREATE PROCEDURE y SHOW CREATE FUNCTION

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

Este comando es una extensión de MySQL . Similar a `SHOW CREATE TABLE`, retorna la cadena exacta que puede usarse para recrear la rutina nombrada.

```
mysql> SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
       Function: hello
       sql_mode:
Create Function: CREATE FUNCTION `test`.`hello`(s CHAR(20)) RETURNS CHAR(50)
RETURN CONCAT('Hello, ',s,'!')
```

19.2.5. SHOW PROCEDURE STATUS y SHOW FUNCTION STATUS

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

Este comando es una extensión de MySQL . Retorna características de rutinas, como el nombre de la base de datos, nombre, tipo, creador y fechas de creación y modificación. Si no se especifica un patrón, le lista la información para todos los procedimientos almacenados, en función del comando que use.

```
mysql> SHOW FUNCTION STATUS LIKE 'hello'\G
*************************** 1. row ***************************
           Db: test
         Name: hello
         Type: FUNCTION
      Definer: testuser@localhost
     Modified: 2004-08-03 15:29:37
      Created: 2004-08-03 15:29:37
Security_type: DEFINER
      Comment:
```

También puede obtener información de rutinas almacenadas de la tabla ROUTINES en INFORMATION_SCHEMA. Consulte Sección 22.1.14, “La tabla INFORMATION_SCHEMA ROUTINES”.

19.2.6. La sentencia CALL

```
CALL sp_name([parameter[,...]])
```

El comando CALL invoca un procedimiento definido préviamente con CREATE PROCEDURE.

CALL puede pasar valores al llamador usando parámetros declarados como OUT o INOUT . También “retorna” el número de registros afectados, que con un programa cliente puede obtenerse a nivel SQL llamando la función ROW_COUNT() y desde C llamando la función de la API C mysql_affected_rows() .

19.2.7. Sentencia compuesta BEGIN ... END

```
[etiqueta_inicio:] BEGIN
    [lista_sentencias]
END [etiqueta_fin]
```

La sintaxis BEGIN ... END se utiliza para escribir sentencias compuestas que pueden aparecer en el interior de procedimientos almacenados y triggers. Una sentencia compuesta puede contener múltiples sentencias, encerradas por las palabras BEGIN y END. *lista_sentencias* es una lista de una o más sentencias. Cada sentencia dentro de *lista_sentencias* debe terminar con un punto y como (;) delimitador de sentencias. *lista_sentencias* es opcional, lo que significa que la sentencia compuesta vacía (BEGIN END es correcta.

El uso de múltiples sentencias requiere que el cliente pueda enviar cadenas de sentencias que contengan el delimitador ;. Esto se gestiona en el cliente de línea de comandos mysql con el comando delimiter. Cambiar el delimitador de fin de sentencia ; (por ejemplo con //) permite utilizar ; en el cuerpo de una rutina. Véase por ejemplo Sección 19.2.1, “CREATE PROCEDURE y CREATE FUNCTION”.

Un comando compuesto puede etiquetarse. No se puede poner *end_label* a no ser que también esté presente *begin_label* , y si ambos están, deben ser iguales.

La cláusula opcional [NOT] ATOMIC no está soportada todavía. Esto significa que no hay un punto transaccional al inicio del bloque de instrucciones y la cláusula BEGIN usada en este contexto no tiene efecto en la transacción actual.

19.2.8. Sentencia DECLARE

El comando DECLARE se usa para definir varios iconos locales de una rutina: las variables locales (consulte Sección 19.2.9, “Variables en procedimientos almacenados”), condiciones y handlers (consulte Sección 19.2.10, “Conditions and Handlers”) y cursores (consulte Sección 19.2.11, “Cursores”). Los comandos SIGNAL y RESIGNAL no se soportan en la actualidad.

DECLARE puede usarse sólo dentro de comandos compuestos BEGIN ... END y deben ser su inicio, antes de cualquier otro comando.

Los cursores deben declararse antes de declarar los handlers, y las variables y condiciones deben declararse antes de declarar los cursores o handlers.

19.2.9. Variables en procedimientos almacenados

Pude declarar y usar una variable dentro de una rutina.

19.2.9.1. Declarar variables locales con DECLARE

```
DECLARE var_name[,...] type [DEFAULT value]
```

Este comando se usa para declarar variables locales. Para proporcionar un valor por defecto para la variable, incluya una cláusula DEFAULT . El valor puede especificarse como expresión, no necesita ser una constante. Si la cláusula DEFAULT no está presente, el valor inicial es NULL.

La visibilidad de una variable local es dentro del bloque BEGIN ... END donde está declarado. Puede usarse en bloques anidados excepto aquéllos que declaren una variable con el mismo nombre.

19.2.9.2. Sentencia SET para variables

```
SET var_name = expr [, var_name = expr] ...
```

El comando SET en procedimientos almacenados es una versión extendida del comando general SET. Las variables referenciadas pueden ser las declaradas dentro de una rutina, o variables de servidor globales.

El comando SET en procedimientos almacenados se implemnta como parte de la sintaxis SET pre-existente. Esto permite una sintaxis extendida de SET a=x, b=y, ... donde distintos tipos de variables (variables declaradas local y globalmente y variables de sesión del servidor) pueden mezclarse. Esto permite combinaciones de variables locales y algunas opciones que tienen sentido sólo para variables de sistema; en tal caso, las opciones se reconocen pero se ignoran.

19.2.9.3. La sentencia SELECT ... INTO

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

Esta sintaxis SELECT almacena columnas seleccionadas directamente en variables. Por lo tanto, sólo un registro puede retornarse.

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

19.2.10. Conditions and Handlers

Ciertas condiciones pueden requerir un tratamiento específico. Estas condiciones pueden estar relacionadas con errores, así como control de flujo general dentro de una rutina.

19.2.10.1. Condiciones DECLARE

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

Este comando especifica condiciones que necesitan tratamiento específico. Asocia un nombre con una condición de error específica. El nombre puede usarse subsecuentemente en un comando DECLARE HANDLER . Consulte Sección 19.2.10.2, “DECLARE handlers”.

Además de valores SQLSTATE , los códigos de error MySQL se soportan.

19.2.10.2. DECLARE handlers

```
DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement

handler_type:
    CONTINUE
  | EXIT
  | UNDO

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
  | mysql_error_code
```

Este comando especifica handlers que pueden tratar una o varias condiciones. Si una de estas condiciones ocurren, el comando especificado se ejecuta.

Para un handler `CONTINUE` , continúa la rutina actual tras la ejecución del comando del handler. Para un handler `EXIT` , termina la ejecución del comando compuesto `BEGIN...END` actual. El handler de tipo `UNDO` todavía no se soporta.

- `SQLWARNING` es una abreviación para todos los códigos SQLSTATE que comienzan con `01`.
- `NOT FOUND` es una abreviación para todos los códigos SQLSTATE que comienzan con `02`.
- `SQLEXCEPTION` es una abreviación para todos los códigos SQLSTATE no tratados por `SQLWARNING` o `NOT FOUND`.

Además de los valores SQLSTATE , los códigos de error MySQL se soportan.

Por ejemplo:

```
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)
```

Tenga en cuenta que `@x` es `3`, lo que muestra que MySQL se ha ejecutado al final del procedimiento. Si la línea `DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;` no está presente,

MySQL habría tomado la ruta por defecto (`EXIT`) tras el segundo `INSERT` fallido debido a la restricción `PRIMARY KEY` , y `SELECT @x` habría retornado `2`.

19.2.11. Cursores

Se soportan cursores simples dentro de procedimientos y funciones almacenadas. La sintaxis es la de SQL empotrado. Los cursores no son sensibles, son de sólo lectura, y no permiten scrolling. No sensible significa que el servidor puede o no hacer una copia de su tabla de resultados.

Los cursores deben declararse antes de declarar los handlers, y las variables y condiciones deben declararse antes de declarar cursores o handlers.

Por ejemplo:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  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 SQLSTATE '02000' SET done = 1;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
       IF b < c THEN
          INSERT INTO test.t3 VALUES (a,b);
       ELSE
          INSERT INTO test.t3 VALUES (a,c);
       END IF;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
  CLOSE cur2;
END
```

19.2.11.1. Declarar cursores

```
DECLARE cursor_name CURSOR FOR select_statement
```

Este comando declara un cursor. Pueden definirse varios cursores en una rutina, pero cada cursor en un bloque debe tener un nombre único.

El comando `SELECT` no puede tener una cláusula `INTO` .

19.2.11.2. Sentencia `OPEN` del cursor

```
OPEN cursor_name
```

Este comando abre un cursor declarado préviamente.

19.2.11.3. Sentencia de cursor `FETCH`

```
FETCH cursor_name INTO var_name [, var_name] ...
```

Este comando trata el siguiente registro (si existe) usando el cursor abierto que se especifique, y avanza el puntero del cursor.

Si no existen más registros disponibles, ocurrirá una condición de Sin Datos con el valor SQLSTATE 02000. Puede configurar un manejador (handler) para detectar esta condición (o para una condición NOT FOUND). Vea un ejemplo en la sección Sección 19.2.11, “Cursores”.

19.2.11.4. Sentencia de cursor `CLOSE`

```
CLOSE cursor_name
```

Este comando cierra un cursor abierto préviamente.

Si no se cierra explícitamente, un cursor se cierra al final del comando compuesto en que se declara.

19.2.12. Constructores de control de flujo

Los constructores `IF`, `CASE`, `LOOP`, `WHILE`, `ITERATE`, y `LEAVE` están completamente implementados.

Estos constructores pueden contener un comando simple, o un bloque de comandos usando el comando compuesto `BEGIN ... END`. Los constructores pueden estar anidados.

Los bucles `FOR` no están soportados.

19.2.12.1. Sentencia `IF`

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

`IF` implementa un constructor condicional básico. Si *`search_condition`* se evalúa a cierto, el comando SQL correspondiente listado se ejectua. Si no coincide ninguna *`search_condition`* se ejecuta el comando listado en la cláusula `ELSE`. *`statement_list`* puede consistir en varios comandos.

Tenga en cuenta que también hay una *función* `IF()`, que difiere del *commando* `IF` descrito aquí. Consulte Sección 12.2, “Funciones de control de flujo”.

19.2.12.2. La sentencia `CASE`

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

O:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
```

```
END CASE
```

El comando `CASE` para procedimientos almacenados implementa un constructor condicional complejo. Si una *`search_condition`* se evalúa a cierto, el comando SQL correspondiente se ejecuta. Si no coincide ninguna condición de búsqueda, el comando en la cláusula `ELSE` se ejecuta.

Nota: La sitaxis de un *comando* `CASE` mostrado aquí para uso dentro de procedimientos almacenados difiere ligeramente de la *expresión* `CASE` SQL descrita en Sección 12.2, “Funciones de control de flujo”. El comando `CASE` no puede tener una cláusula `ELSE NULL` y termina con `END CASE` en lugar de `END`.

19.2.12.3. Sentencia `LOOP`

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

`LOOP` implementa un constructor de bucle simple que permite ejecución repetida de comandos particulares. El comando dentro del bucle se repite hasta que acaba el bucle, usualmente con un comando `LEAVE` .

Un comando `LOOP` puede etiquetarse. *`end_label`* no puede darse hasta que esté presente *`begin_label`* , y si ambos lo están, deben ser el mismo.

19.2.12.4. Sentencia `LEAVE`

```
LEAVE label
```

Este comando se usa para abandonar cualquier control de flujo etiquetado. Puede usarse con `BEGIN ... END` o bucles.

19.2.12.5. La sentencia `ITERATE`

```
ITERATE label
```

`ITERATE` sólo puede aparecer en comandos `LOOP`, `REPEAT`, y `WHILE` . `ITERATE` significa “vuelve a hacer el bucle.”

Por ejemplo:

```
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
```

19.2.12.6. Sentencia `REPEAT`

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
```

```
END REPEAT [end_label]
```

El comando/s dentro de un comando `REPEAT` se repite hasta que la condición *`search_condition`* es cierta.

Un comando `REPEAT` puede etiquetarse. *`end_label`* no puede darse a no ser que *`begin_label`* esté presente, y si lo están, deben ser el mismo.

Por ejemplo:

```
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> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

19.2.12.7. Sentencia `WHILE`

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

El comado/s dentro de un comando `WHILE` se repite mientras la condición *`search_condition`* es cierta.

Un comando `WHILE` puede etiquetarse. *`end_label`* no puede darse a no ser que *`begin_label`* también esté presente, y si lo están, deben ser el mismo.

Por ejemplo:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END
```

19.3. Registro binario de procedimientos almacenados y disparadores

Esta sección describe cómo MySQL trata procedimientos almacenados (procedimientos o funciones) con respecto a logueo binario. La sección también se aplica a disparadores.

El log binario contiene información sobre comandos SQL que modifican contenidos de base de datos. Esta información se almacena en la forma de “eventos” que describen las modificaciones.

El log binario tiene dos propósitos importantes:

- La base de replicación es que el maestro envía los eventos contenidos en su log binario a sus esclavos, que ejecuta estos eventos para hacer los mismos cambios de datos que se hacen en el maestro. Consulte Sección 6.2, “Panorámica de la implementación de la replicación”.
- Ciertas operaciones de recuperación de datos necesitan usar el log binario. Tras hacer una restauración de un fichero de copia de seguridad, los eventos en el log binario que se guardaron tras hacer la copia de seguridad se re-ejecutan. Estos eventos actualizan la base de datos desde el punto de la copia de seguridad. Consulte Sección 5.8.2.2, “Usar copias de seguridad para una recuperación”.

El logueo de procedimientos almacenados difiere antes y después de MySQL 5.0.6. Antes de MySQL 5.0.6, los comandos que crean y usan procedimientos almacenados no se escriben en el log binario, pero los comandos invocados desde procedimientos almacenados se loguean. Suponga que ejecuta los siguientes comandos:

```
CREATE PROCEDURE mysp INSERT INTO t VALUES(1);
CALL mysp;
```

Para este ejemplo, sólo el comando `INSERT` aparecerá en el log binario. Los comandos `CREATE PROCEDURE` y `CALL` no aparecerán. La ausencia de comandos relacionados con rutinas en el log binario significa que procedimientos almacenados no se replican correctamente. También significa que para operaciones de recuperación de datos, re-ejectuar eventos en el log binario no recupera procedimientos almacenados.

Para tratar estos temas de replicación y recuperación de datos, se cambió el logueo de procedimientos almacenados en MySQL 5.0.6. Sin embargo, este cambio provoca nuevos temas, que se presentan en la siguiente discusión.

A no ser que se diga lo contrario, estas notas asumen que no ha activado el logueo binario arrancando el servidor con la opción `--log-bin` . (Si el log binario no se activa, la replicación no es posible, ni está disponible el log binario para replicación de datos.) Consulte Sección 5.10.3, “El registro binario (Binary Log)”.

Las características de logueo binario para comandos de procedimientos almacenados se describen en la siguiente lista. Algunos de los iconos indican problemas que debería conocer, pero en algunos casos, hay inicializaciones de servidor que puede modificar o soluciones que puede usar.

- Los comandos `CREATE PROCEDURE`, `CREATE FUNCTION`, `ALTER PROCEDURE`, y `ALTER FUNCTION` se escriben en el log binario, como lo son `CALL`, `DROP PROCEDURE`, y `DROP FUNCTION`.

  Sin embargo, hay implicaciones de seguridad para replicación: para crear una rutina, un usuario debe tener el permiso `CREATE ROUTINE` , pero un usuario que tenga este permiso puede escribir una rutina para realizar cualquier acción en un servidor esclavo ya que el flujo SQL en el esclavo corre con todos los permisos. Por ejemplo, si el maestro y el esclavo tienen valores de ID del servidor de 1 y 2 respectivamente, un usuario en el maestro puede crear e invocar procedimientos como sigue:

  ```
  mysql> delimiter //
  mysql> CREATE PROCEDURE mysp ()
      -> BEGIN
      ->   IF @@server_id=2 THEN DROP DATABASE accounting; END IF;
      -> END;
      -> //
  ```

```
mysql> delimiter ;
mysql> CALL mysp();
```

Los comandos `CREATE PROCEDURE` y `CALL` se escriben en el log binario, así que los ejecutará el esclavo. Ya que el flujo SQL del esclavo tiene todos los permisos, borra la base de datos `accounting` .

Para evitar este peligro en servidores con logueo binario activado, MySQL 5.0.6 introduce el requerimiento que los creadores de procedimientos almacenados y funciones deben tener el permiso `SUPER` , además del permiso `CREATE ROUTINE` requerido. Similarmente, para usar `ALTER PROCEDURE` o `ALTER FUNCTION`, debe tener el permiso `SUPER` además del permiso `ALTER ROUTINE` . Sin el permiso `SUPER` ocurre un error:

```
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)
```

Puede no querer forzar el requerimiento en los creadores de rutinas que deben tener el permiso `SUPER`. Por ejemplo, todos los usuarios con el permiso `CREATE ROUTINE` en su sistema pueden ser desarrolladores de aplicaciones con experiencia. Para desactivar el requerimiento de `SUPER`, cambie la variable de sistema global `log_bin_trust_routine_creators` a 1. Por defecto, esta variable vale 0, pero puede cambiarla así:

```
mysql> SET GLOBAL log_bin_trust_routine_creators = 1;
```

Puede activar esta variable usando la opción `--log-bin-trust-routine-creators` al arrancar el servidor.

Si el logueo binario no está activado, `log_bin_trust_routine_creators` no se aplica y no se ncesita el permiso `SUPER` para creación de rutinas.

- Una rutina no-determinista que realiza actualizaciones no es repetible, que puede tener dos efectos no deseables:
  - El esclavo será distinto al maestro.
  - Los datos restaurados serán distintos a los originales.

  Para tratar estos problemas, MySQL fuerza los siguientes requerimientos: En un servidor maestro, la creación y alteración de una rutina se rehúsa a no ser que la rutina se declare como determinista o que no modifique datos. Esto significa que cuando crea una rutina, debe declarar que es determinista o que no cambia datos. Dos conjuntos de características de rutinas se aplica aquí:

  - `DETERMINISTIC` y `NOT DETERMINISTIC` indican si una rutina siempre produce el mismo resultado para entradas dadas. Por defecto es `NOT DETERMINISTIC` si no se da ninguna característica, así que debe especificar `DETERMINISTIC` explícitamente para declarar que la rutina es determinista.

    El uso de las funciones `NOW()` (o sus sinónimos) o `RAND()` no hacen una rutina no-determinista necesariamente. Para `NOW()`, el log binario incluye la fecha y hora y replica correctamente. `RAND()` también replica correctamente mientras se invoque sólo una vez dentro de una rutina. (Puede considerar la fecha y hora de ejecución de la rutina y la semilla de números aleatorios como entradas implícitas que son idénticas en el maestro y el esclavo.)
  - `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, y `MODIFIES SQL` proporciona información acerca de si la rutina lee o escribe datos. Tanto `NO SQL` o `READS SQL DATA` indican que una rutina no cambia datos, pero debe especificar uno de estos explícitamente ya que por defecto es `CONTAINS SQL` si ninguna de estas características se da.

Por defecto, para que un comando `CREATE PROCEDURE` o `CREATE FUNCTION` sea aceptado, `DETERMINISTIC` o `NO SQL` y `READS SQL DATA` deben especificarse explícitamente. De otro modo ocurre un error:

```
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)
```

Si asigna a `log_bin_trust_routine_creators` 1, el requerimiento que la rutina sea determinista o que no modifique datos se elimina.

Tenga en cuenta que la naturaleza de una rutina se basa en la “honestidad” del creador: MySQL no comprueba que una rutina declarada `DETERMINISTIC` no contenga comandos que produzcan productos no deterministas.

- Un comando `CALL` se escribe en el log binario si la rutina no retorna error, pero no en otro caso. Cuando una rutina que modifica datos falla, obtiene esta advertencia:

```
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
```

Este logueo del comportamiento tiene un potencial para causar problemas. Si una rutina modifica parcialmente una tabla no transaccional (tal como una tabla `MyISAM`) y retorna un error, el log binario no refleja estos cambios. Para protegerse de esto, debe usar trablas transaccionales en la rutina y modificar las tablas dentro de transacciones.

Si usa la palabra clave `IGNORE` con `INSERT`, `DELETE`, o `UPDATE` para ignorar errores dentro de una rutina, una actualización parcial puede ocurrir sin producir error. Tales comandos se loguean y se replican normalmente.

- Si una función almacenada se invoca dentro de un comando tal como `SELECT` que no modifica datos, la ejecución de la función no se escribe en el log binario, incluso si la función misma modifica datos. Este comportamiento de logueo tiene potencial para causar problemas. Suponga que una función `myfunc()` se define así:

```
CREATE FUNCTION myfunc () RETURNS INT
BEGIN
  INSERT INTO t (i) VALUES(1);
  RETURN 0;
END;
```

Dada esta definición, el comando siguiente modifica la tabla `t` porque `myfunc()` modifica `t`, pero el comando no se escribe en el log binario porque es un `SELECT`:

```
SELECT myfunc();
```

Una solución de este problema es invocar funciones que actualizan dentro de comandos que hacen actualizaciones. Tenga en cuenta que aunque el comando `DO` a veces se ejecuta como efecto colateral de evaluar una expresión, `DO` no es una solución aquí porque no está escrito en el log binario.

- Los comandos ejecutados dentro de una rutina no se escriben en el log binario. Suponga que ejectua los siguientes comandos:

```
CREATE PROCEDURE mysp INSERT INTO t VALUES(1);
CALL mysp;
```

Para este ejemplo, los comandos `CREATE PROCEDURE` y `CALL` aparecerán en el log binario. El comando `INSERT` no aparecerá. Esto arregla el problema que ocurre antes de MySQL 5.0.6 en que los comandos `CREATE PROCEDURE` y `CALL` no se loguearon y `INSERT` se logueó.

- En servidores esclavos, la siguiente limitación se aplica cuando se determina qué eventos del maestro se replican: reglas `--replicate-*-table` no se aplican a comandos `CALL` o a comandos dentro de rutinas: Las reglas en estos casos siempre retornan “replica!”

Los disparadores son similares a los procedimientos almacenados, así que las notas precedentes también se aplican a disparadores con las siguientes excepciones: `CREATE TRIGGER` no tiene una característica `DETERMINISTIC` opcional, así que los disparadores se asumen como deterministas. Sin embargo, esta hipótesis puede ser inválida en algunos casos . Por ejemplo, la función `UUID()` no es determinista (y no replica). Debe ser cuidadoso acerca de usar tales funciones y disparadores.

Los disparadores actualmente no actualizan tablas, pero lo harán en el futuro. Por esta razón, los mensajes de error similares a los de los procedimientos almacenados ocurren con `CREATE TRIGGER` si no tiene el permiso `SUPER` y `log_bin_trust_routine_creators` es 0.

Los temas tratados en esta sección son resultado del hecho que el logueo binario se hace a nivel de comandos SQL. MySQL 5.1 introducirá logueo binario a nivel de registro, lo que ocurre en un nivel más granular que especifica qué cambios hacer a registros individuales como resultado de ejecutar comandos SQL.

Capítulo 20. Disparadores (triggers)

Tabla de contenidos

A partir de MySQL 5.0.2 se incorporó el soporte básico para disparadores (triggers). Un disparador es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular. Por ejemplo, las siguientes sentencias crean una tabla y un disparador para sentencias `INSERT` dentro de la tabla. El disparador suma los valores insertados en una de las columnas de la tabla:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

Este capítulo describe la sintaxis para crear y eliminar disparadores, y muestra algunos ejemplos de cómo utilizarlos. Las restricciones en el uso de disparadores se tratan en Apéndice H, *Restricciones en características de MySQL*.

En Sección 19.3, “Registro binario de procedimientos almacenados y disparadores” se describe la forma en que se realiza el registro binario (binary logging) para los disparadores.

20.1. Sintaxis de `CREATE TRIGGER`

```
CREATE TRIGGER nombre_disp momento_disp evento_disp
    ON nombre_tabla FOR EACH ROW sentencia_disp
```

Un disparador es un objeto con nombre en una base de datos que se asocia con una tabla, y se activa cuando ocurre un evento en particular para esa tabla.

El disparador queda asociado a la tabla *`nombre_tabla`*. Esta debe ser una tabla permanente, no puede ser una tabla `TEMPORARY` ni una vista.

`momento_disp` es el momento en que el disparador entra en acción. Puede ser `BEFORE` (antes) o `AFTER` (despues), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.

`evento_disp` indica la clase de sentencia que activa al disparador. Puede ser `INSERT`, `UPDATE`, o `DELETE`. Por ejemplo, un disparador `BEFORE` para sentencias `INSERT` podría utilizarse para validar los valores a insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores `BEFORE UPDATE`. Pero sí es posible tener los disparadores `BEFORE UPDATE` y `BEFORE INSERT` o `BEFORE UPDATE` y `AFTER UPDATE`.

`sentencia_disp` es la sentencia que se ejecuta cuando se activa el disparador. Si se desean ejecutar múltiples sentencias, deben colocarse entre `BEGIN ... END`, el constructor de sentencias compuestas. Esto además posibilita emplear las mismas sentencias permitidas en rutinas almacenadas. Consulte Sección 19.2.7, “Sentencia compuesta `BEGIN ... END`”.

Note: Antes de MySQL 5.0.10, los disparadores no podían contener referencias directas a tablas por su nombre. A partir de MySQL 5.0.10, se pueden escribir disparadores como el llamado `testref`, que se muestra en este ejemplo:

```
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
|

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);
```

Si en la tabla `test1` se insertan los siguientes valores:

```
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
```

Entonces los datos en las 4 tablas quedarán así:

```
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   |
+----+------+
|  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)
```

Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias `OLD` y `NEW`. `OLD.nombre_col` hace referencia a una columna de una fila existente, antes de ser actualizada o borrada. `NEW.nombre_col` hace referencia a una columna en una nueva fila a punto de ser insertada, o en una fila existente luego de que fue actualizada.

El uso de `SET NEW.nombre_col = valor` necesita que se tenga el privilegio `UPDATE` sobre la columna. El uso de `SET nombre_var = NEW.nombre_col` necesita el privilegio `SELECT` sobre la columna.

Nota: Actualmente, los disparadores no son activados por acciones llevadas a cabo en cascada por las restricciones de claves extranjeras. Esta limitación se subsanará tan pronto como sea posible.

La sentencia `CREATE TRIGGER` necesita el privilegio `SUPER`. Esto se agregó en MySQL 5.0.2.

20.2. Sintaxis de `DROP TRIGGER`

```
DROP TRIGGER [nombre_esquema.]nombre_disp
```

Elimina un disparador. El nombre de esquema es opcional. Si el esquema se omite, el disparador se elimina en el esquema actual.

Anteriormente a la versión 5.0.10 de MySQL, se requería el nombre de tabla en lugar del nombre de esquema. (*`nom_tabla.nom_disp`*).

Nota: cuando se actualice desde una versión anterior de MySQL 5 a MySQL 5.0.10 o superior, se deben eliminar todos los disparadores antes de actualizar y volver a crearlos después, o `DROP TRIGGER` no funcionará luego de la actualización.

La sentencia `DROP TRIGGER` necesita que se posea el privilegio `SUPER`, que se introdujo en MySQL 5.0.2.

20.3. Utilización de disparadores

El soporte para disparadores se incluyó a partir de MySQL 5.0.2. Actualmente, el soporte para disparadores es básico, por lo tanto hay ciertas limitaciones en lo que puede hacerse con ellos. Esta sección trata sobre el uso de los disparadores y las limitaciones vigentes.

Un disparador es un objeto de base de datos con nombre que se asocia a una tabla, y se activa cuando ocurre un evento en particular para la tabla. Algunos usos para los disparadores es verificar valores a ser insertados o llevar a cabo cálculos sobre valores involucrados en una actualización.

Un disparador se asocia con una tabla y se define para que se active al ocurrir una sentencia `INSERT`, `DELETE`, o `UPDATE` sobre dicha tabla. Puede también establecerse que se active antes o despues de la sentencia en cuestión. Por ejemplo, se puede tener un disparador que se active antes de que un registro sea borrado, o después de que sea actualizado.

Para crear o eliminar un disparador, se emplean las sentencias `CREATE TRIGGER` y `DROP TRIGGER`. La sintaxis de las mismas se describe en Sección 20.1, “Sintaxis de `CREATE TRIGGER`” y Sección 20.2, “Sintaxis de `DROP TRIGGER`”.

Este es un ejemplo sencillo que asocia un disparador con una tabla para cuando reciba sentencias `INSERT`. Actúa como un acumulador que suma los valores insertados en una de las columnas de la tabla.

La siguiente sentencia crea la tabla y un disparador asociado a ella:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

La sentencia `CREATE TRIGGER` crea un disparador llamado `ins_sum` que se asocia con la tabla `account`. También se incluyen cláusulas que especifican el momento de activación, el evento activador, y qué hacer luego de la activación:

- La palabra clave `BEFORE` indica el momento de acción del disparador. En este caso, el disparador debería activarse antes de que cada registro se inserte en la tabla. La otra palabra clave posible aqui es `AFTER`.
- La plabra clave `INSERT` indica el evento que activará al disparador. En el ejemplo, la sentencia `INSERT` causará la activación. También pueden crearse disparadores para sentencias `DELETE` y `UPDATE`.
- Las sentencia siguiente, `FOR EACH ROW`, define lo que se ejecutará cada vez que el disparador se active, lo cual ocurre una vez por cada fila afectada por la sentencia activadora. En el ejemplo, la sentencia activada es un sencillo `SET` que acumula los valores insertados en la columna `amount`. La sentencia se refiere a la columna como `NEW.amount`, lo que significa “el valor de la columna `amount` que será insertado en el nuevo registro.”

Para utilizar el disparador, se debe establecer el valor de la variable acumulador a cero, ejecutar una sentencia `INSERT`, y ver qué valor presenta luego la variable.

```
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';
+-----------------------+
| Total amount inserted |
+-----------------------+
```

```
| 1852.48               |
+-----------------------+
```

En este caso, el valor de `@sum` luego de haber ejecutado la sentencia `INSERT` es `14.98 + 1937.50 - 100`, o `1852.48`.

Para eliminar el disparador, se emplea una sentencia `DROP TRIGGER`. El nombre del disparador debe incluir el nombre de la tabla:

```
mysql> DROP TRIGGER account.ins_sum;
```

Debido a que un disparador está asociado con una tabla en particular, no se pueden tener múltiples disparadores con el mismo nombre dentro de una tabla. También se debería tener en cuenta que el espacio de nombres de los disparadores puede cambiar en el futuro de un nivel de tabla a un nivel de base de datos, es decir, los nombres de disparadores ya no sólo deberían ser únicos para cada tabla sino para toda la base de datos. Para una mejor compatibilidad con desarrollos futuros, se debe intentar emplear nombres de disparadores que no se repitan dentro de la base de datos.

Adicionalmente al requisito de nombres únicos de disparador en cada tabla, hay otras limitaciones en los tipos de disparadores que pueden crearse. En particular, no se pueden tener dos disparadores para una misma tabla que sean activados en el mismo momento y por el mismo evento. Por ejemplo, no se pueden definir dos `BEFORE INSERT` o dos `AFTER UPDATE` en una misma tabla. Es improbable que esta sea una gran limitación, porque es posible definir un disparador que ejecute múltiples sentencias empleando el constructor de sentencias compuestas `BEGIN ... END` luego de `FOR EACH ROW`. (Más adelante en esta sección puede verse un ejemplo).

También hay limitaciones sobre lo que puede aparecer dentro de la sentencia que el disparador ejecutará al activarse:

- El disparador no puede referirse a tablas directamente por su nombre, incluyendo la misma tabla a la que está asociado. Sin embargo, se pueden emplear las palabras clave `OLD` y `NEW`. `OLD` se refiere a un registro existente que va a borrarse o que va a actualizarse antes de que esto ocurra. `NEW` se refiere a un registro nuevo que se insertará o a un registro modificado luego de que ocurre la modificación.
- El disparador no puede invocar procedimientos almacenados utilizando la sentencia `CALL`. (Esto significa, por ejemplo, que no se puede utilizar un procedimiento almacenado para eludir la prohibición de referirse a tablas por su nombre).
- El disparador no puede utilizar sentencias que inicien o finalicen una transacción, tal como `START TRANSACTION`, `COMMIT`, o `ROLLBACK`.

Las palabras clave `OLD` y `NEW` permiten acceder a columnas en los registros afectados por un disparador. (`OLD` y `NEW` no son sensibles a mayúsculas). En un disparador para `INSERT`, solamente puede utilizarse `NEW.`*`nom_col`*; ya que no hay una versión anterior del registro. En un disparador para `DELETE` sólo puede emplearse `OLD.`*`nom_col`*, porque no hay un nuevo registro. En un disparador para `UPDATE` se puede emplear `OLD.`*`nom_col`* para referirse a las columnas de un registro antes de que sea actualizado, y `NEW.`*`nom_col`* para referirse a las columnas del registro luego de actualizarlo.

Una columna precedida por `OLD` es de sólo lectura. Es posible hacer referencia a ella pero no modificarla. Una columna precedida por `NEW` puede ser referenciada si se tiene el privilegio `SELECT` sobre ella. En un disparador `BEFORE`, también es posible cambiar su valor con `SET NEW.`*`nombre_col = valor`* si se tiene el privilegio de `UPDATE` sobre ella. Esto significa que un disparador puede usarse para modificar los valores antes que se inserten en un nuevo registro o se empleen para actualizar uno existente.

En un disparador `BEFORE`, el valor de `NEW` para una columna `AUTO_INCREMENT` es 0, no el número secuencial que se generará en forma automática cuando el registro sea realmente insertado.

`OLD` y `NEW` son extensiones de MySQL para los disparadores.

Empleando el constructor `BEGIN ... END`, se puede definir un disparador que ejecute sentencias múltiples. Dentro del bloque `BEGIN`, también pueden utilizarse otras sintaxis permitidas en rutinas almacenadas, tales como condicionales y bucles. Como sucede con las rutinas almacenadas, cuando se crea un disparador que ejecuta sentencias múltiples, se hace necesario redefinir el delimitador de sentencias si se ingresará el disparador a través del programa `mysql`, de forma que se pueda utilizar el caracter '`;`' dentro de la definición del disparador. El siguiente ejemplo ilustra estos aspectos. En él se crea un disparador para `UPDATE`, que verifica los valores utilizados para actualizar cada columna, y modifica el valor para que se encuentre en un rango de 0 a 100. Esto debe hacerse en un disparador `BEFORE` porque los valores deben verificarse antes de emplearse para actualizar el registro:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

Podría parecer más fácil definir una rutina almacenada e invocarla desde el disparador utilizando una simple sentencia `CALL`. Esto sería ventajoso también si se deseara invocar la misma rutina desde distintos disparadores. Sin embargo, una limitación de los disparadores es que no pueden utilizar `CALL`. Se debe escribir la sentencia compuesta en cada `CREATE TRIGGER` donde se la desee emplear.

MySQL gestiona los errores ocurridos durante la ejecución de disparadores de esta manera:

- Si lo que falla es un disparador `BEFORE`, no se ejecuta la operación en el correspondiente registro.
- Un disparador `AFTER` se ejecuta solamente si el disparador `BEFORE` (de existir) y la operación se ejecutaron exitosamente.
- Un error durante la ejecución de un disparador `BEFORE` o `AFTER` deriva en la falla de toda la sentencia que provocó la invocación del disparador.
- En tablas transaccionales, la falla de un disparador (y por lo tanto de toda la sentencia) debería causar la cancelación (rollback) de todos los cambios realizados por esa sentencia. En tablas no transaccionales, cualquier cambio realizado antes del error no se ve afectado.

Capítulo 21. Vistas (Views)

Tabla de contenidos

Las vistas (incluyendo vistas actualizables) fueron introducidas en la versión 5.0 del servidor de base de datos MySQL

En este capítulo se tratan los siguientes temas:

- Creación o modificación de vistas con `CREATE VIEW` o `ALTER VIEW`
- Eliminación de vistas con `DROP VIEW`
- Obtención de información de definición de una vista (metadatos) con `SHOW CREATE VIEW`

Para obtener información sobre las restricciones en el uso de vistas consulte Apéndice H, *Restricciones en características de MySQL*.

Si ha actualizado a MySQL 5.0.1 desde una versión anterior, debería actualizar las tablas de permisos para que contengan los privilegios relacionados con vistas. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

21.1. Sintaxis de `ALTER VIEW`

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW nombre_vista [(columnas)]
    AS sentencia_select
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Esta sentencia modifica la definición de una vista existente. La sintaxis es semejante a la empleada en `CREATE VIEW`. Consulte Sección 21.2, “Sintaxis de `CREATE VIEW`”. Se requiere que posea los permisos `CREATE VIEW` y `DELETE` para la vista, y algún privilegio en cada columna seleccionada por la sentencia `SELECT`.

Esta sentencia se introdujo en MySQL 5.0.1.

21.2. Sintaxis de `CREATE VIEW`

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW nombre_vista [(columnas)]
    AS sentencia_select
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Esta sentencia crea una vista nueva o reemplaza una existente si se incluye la cláusula `OR REPLACE`. La *`sentencia_select`* es una sentencia `SELECT` que proporciona la definición de la vista. Puede estar dirigida a tablas de la base o a otras vistas.

Se requiere que posea el permiso CREATE VIEW para la vista, y algún privilegio en cada columna seleccionada por la sentencia SELECT. Para columnas incluidas en otra parte de la sentencia SELECT debe poseer el privilegio SELECT. Si está presente la cláusula OR REPLACE, también deberá tenerse el privilegio DELETE para la vista.

Toda vista pertenece a una base de datos. Por defecto, las vistas se crean en la base de datos actual. Pera crear una vista en una base de datos específica, indíquela con *base_de_datos.nombre_vista* al momento de crearla.

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por eso, una base de datos no puede contener una tabla y una vista con el mismo nombre.

Al igual que las tablas, las vistas no pueden tener nombres de columnas duplicados. Por defecto, los nombres de las columnas devueltos por la sentencia SELECT se usan para las columnas de la vista . Para dar explícitamente un nombre a las columnas de la vista utilice la clásula *columnas* para indicar una lista de nombres separados con comas. La cantidad de nombres indicados en *columnas* debe ser igual a la cantidad de columnas devueltas por la sentencia SELECT.

Las columnas devueltas por la sentencia SELECT pueden ser simples referencias a columnas de la tabla, pero tambien pueden ser expresiones conteniendo funciones, constantes, operadores, etc.

Los nombres de tablas o vistas sin calificar en la sentencia SELECT se interpretan como pertenecientes a la base de datos actual. Una vista puede hacer referencia a tablas o vistas en otras bases de datos precediendo el nombre de la tabla o vista con el nombre de la base de datos apropiada.

Las vistas pueden crearse a partir de varios tipos de sentencias SELECT. Pueden hacer referencia a tablas o a otras vistas. Pueden usar combinaciones, UNION, y subconsultas. El SELECT inclusive no necesita hacer referencia a otras tablas. En el siguiente ejemplo se define una vista que selecciona dos columnas de otra tabla, así como una expresión calculada a partir de ellas:

```
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 |
+------+-------+-------+
```

La definición de una vista está sujeta a las siguientes limitaciones:

- La sentencia SELECT no puede contener una subconsulta en su cláusula FROM.
- La sentencia SELECT no puede hacer referencia a variables del sistema o del usuario.
- La sentencia SELECT no puede hacer referencia a parámetros de sentencia preparados.
- Dentro de una rutina almacenada, la definición no puede hacer referencia a parámetros de la rutina o a variables locales.
- Cualquier tabla o vista referenciada por la definición debe existir. Sin embargo, es posible que después de crear una vista, se elimine alguna tabla o vista a la que se hace referencia. Para comprobar la definición de una vista en busca de problemas de este tipo, utilice la sentencia CHECK TABLE.

- La definición no puede hacer referencia a una tabla TEMPORARY, y tampoco se puede crear una vista TEMPORARY.
- Las tablas mencionadas en la definición de la vista deben existir siempre.
- No se puede asociar un disparador con una vista.

En la definición de una vista está permitido ORDER BY, pero es ignorado si se seleccionan columnas de una vista que tiene su propio ORDER BY.

Con respecto a otras opciones o cláusulas incluidas en la definición, las mismas se agregan a las opciones o cláusulas de cualquier sentencia que haga referencia a la vista creada, pero el efecto es indefinido. Por ejemplo, si la definición de una vista incluye una cláusula LIMIT, y se hace una selección desde la vista utilizando una sentencia que tiene su propia cláusula LIMIT, no está definido cuál se aplicará. El mismo principio se extiende a otras opciones como ALL, DISTINCT, o SQL_SMALL_RESULT que se ubican a continuación de la palabra reservada SELECT, y a cláusulas como INTO, FOR UPDATE, LOCK IN SHARE MODE, y PROCEDURE.

Si se crea una vista y luego se modifica el entorno de proceso de la consulta a traves de la modificación de variables del sistema, puede afectar los resultados devueltos por la vista:

```
mysql> CREATE VIEW v AS SELECT CHARSET(CHAR(65)), COLLATION(CHAR(65));
Query OK, 0 rows affected (0.00 sec)

mysql> SET NAMES 'latin1';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-------------------+----------------------+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-------------------+----------------------+
| latin1            | latin1_swedish_ci   |
+-------------------+----------------------+
1 row in set (0.00 sec)

mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-------------------+----------------------+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-------------------+----------------------+
| utf8              | utf8_general_ci     |
+-------------------+----------------------+
1 row in set (0.00 sec)
```

La cláusula opcional ALGORITHM es una extensión de MySQL al SQL estándar. ALGORITHM puede tomar tres valores: MERGE, TEMPTABLE, o UNDEFINED. El algoritmo por defecto es UNDEFINED si no se encuentra presente la cláusula ALGORITHM. El algoritmo afecta la manera en que MySQL procesa la vista.

Para MERGE, el texto de una sentencia que haga referencia a la vista y la definición de la vista son mezclados de forma que parte de la definición de la vista reemplaza las partes correspondientes de la consulta.

Para TEMPTABLE, los resultados devueltos por la vista son colocados en una tabla temporal, la cual es luego utilizada para ejecutar la sentencia.

Para UNDEFINED, MySQL determina el algoritmo que utilizará. En ese caso se prefiere MERGE por sobre TEMPTABLE si es posible, ya que MERGE por lo general es más eficiente y porque la vista no puede ser actualizable si se emplea una tabla temporal.

Una razón para elegir explícitamente TEMPTABLE es que los bloqueos en tablas subyacentes pueden ser liberados despues que la tabla temporal fue creada, y antes de que sea usada para terminar el procesamiento de la sentencia. Esto podría resultar en una liberación del bloqueo más rápida que en el algoritmo MERGE, de modo que otros clientes que utilicen la vista no estarán bloqueados mucho tiempo.

El algoritmo de una vista puede ser UNDEFINED en tres situaciones:

- No se encuentra presente una cláusula ALGORITHMen la sentencia CREATE VIEW.
- La sentencia CREATE VIEW tiene explícitamente una cláusula ALGORITHM = UNDEFINED.
- Se especificó ALGORITHM = MERGE para una vista que solamente puede ser procesada usando una tabla temporal. En este caso, MySQL emite una advertencia y establece el algoritmo en UNDEFINED.

Como se dijo anteriormente, MERGE provoca que las partes correspondientes de la definición de la vista se combinen dentro de la sentencia que hace referencia a la vista. El siguiente ejemplo muestra brevemente cómo funciona el algoritmo MERGE. El ejemplo asume que hay una vista v_merge con esta definición:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Ejemplo 1: Suponiendo que se utilice esta sentencia:

```
SELECT * FROM v_merge;
```

MySQL la gestiona del siguiente modo:

- v_merge se convierte en t
- * se convierte en vc1, vc2, que corresponden a c1, c2
- Se agrega la cláusula WHERE de la vista

La sentencia ejecutada resulta ser:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Ejemplo 2: Suponiendo que se utilice esta sentencia:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

Esta sentencia se gestiona en forma similar a la anterior, a excepción de que vc1 < 100 se convierte en c1 < 100 y la cláusula WHERE de la vista se agrega a la cláusula WHERE de la sentencia empleando un conector AND (y se agregan paréntesis para asegurarse que las partes de la cláusula se ejecutarán en el orden de precedencia correcto). La sentencia ejecutada resulta ser:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Necesariamente, la sentencia a ejecutar tiene una cláusula WHERE con esta forma:

```
WHERE (WHERE de la sentencia) AND (WHERE de la vista)
```

El algoritmo MERGE necesita una relación uno-a-uno entre los registros de la vista y los registros de la tabla subyacente. Si esta relación no se sostiene, debe emplear una tabla temporal en su lugar. No se tendrá una relación uno-a-uno si la vista contiene cualquiera de estos elementos:

- Funciones agregadas (SUM(), MIN(), MAX(), COUNT(), etcétera)
- DISTINCT
- GROUP BY
- HAVING
- UNION o UNION ALL
- Hace referencia solamente a valores literales (en tal caso, no hay una tabla subyacente)

Algunas vistas son actualizables. Esto significa que se las puede emplear en sentencias como UPDATE, DELETE, o INSERT para actualizar el contenido de la tabla subyacente. Para que una vista sea actualizable, debe haber una relación uno-a-uno entre los registros de la vista y los registros de la tabla subyacente. Hay otros elementos que impiden que una vista sea actualizable. Más específicamente, una vista no será actualizable si contiene:

- Funciones agregadas (SUM(), MIN(), MAX(), COUNT(), etcétera)
- DISTINCT
- GROUP BY
- HAVING
- UNION o UNION ALL
- Una subconsulta en la lista de columnas del SELECT
- Join
- Una vista no actualizable en la cláusula FROM
- Una subconsulta en la cláusula WHERE que hace referencia a una tabla en la cláusula FROM
- Hace referencia solamente a valores literales (en tal caso no hay una) tabla subyacenta para actualizar.
- ALGORITHM = TEMPTABLE (utilizar una tabla temporal siempre resulta en una vista no actualizable)

Con respecto a la posibilidad de agregar registros mediante sentencias INSERT, es necesario que las columnas de la vista actualizable también cumplan los siguientes requisitos adicionales:

- No debe haber nombres duplicados entre las columnas de la vista.
- La vista debe contemplar todas las columnas de la tabla en la base de datos que no tengan indicado un valor por defecto.
- Las columnas de la vista deben ser referencias a columnas simples y no columnas derivadas. Una columna derivada es una que deriva de una expresión. Estos son algunos ejemplos de columnas derivadas:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

No puede insertar registros en una vista conteniendo una combinación de columnas simples y derivadas, pero puede actualizarla si actualiza únicamente las columnas no derivadas. Considere esta vista:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

En esta vista no pueden agregarse registros porque col2 es derivada de una expresión. Pero será actualizable si no intenta actualizar col2. Esta actualización es posible:

```
UPDATE v SET col1 = 0;
```

Esta actualización no es posible porque se intenta realizar sobre una columna derivada:

```
UPDATE v SET col2 = 0;
```

A veces, es posible que una vista compuesta por múltiples tablas sea actualizable, asumiendo que es procesada con el algoritmo MERGE. Para que esto funcione, la vista debe usar inner join (no outer join o UNION). Además, solamente puede actualizarse una tabla de la definición de la vista, de forma que la cláusula SET debe contener columnas de sólo una tabla de la vista. Las vistas que utilizan UNION ALL no se pueden actualizar aunque teóricamente fuese posible hacerlo, debido a que en la implementación se emplean tablas temporales para procesarlas.

En vistas compuestas por múltiples tablas, INSERT funcionará si se aplica sobre una única tabla. DELETE no está soportado.

La cláusula WITH CHECK OPTION puede utilizarse en una vista actualizable para evitar inserciones o actualizaciones excepto en los registros en que la cláusula WHERE de la *sentencia_select* se evalúe como true.

En la cláusula WITH CHECK OPTION de una vista actualizable, las palabras reservadas LOCAL y CASCADED determinan el alcance de la verificación cuando la vista está definida en términos de otras vistas. LOCAL restringe el CHECK OPTION sólo a la vista que está siendo definida. CASCADED provoca que las vistas subyacentes también sean verificadas. Si no se indica, el valor por defecto es CASCADED. Considere las siguientes definiciones de tabla y vistas:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
    -> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
    -> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
    -> WITH CASCADED CHECK OPTION;
```

Las vistas v2 y v3 estan definidas en términos de otra vista, v1. v2 emplea check option LOCAL, por lo que las inserciones sólo atraviesan la verificación de v2 . v3 emplea check option CASCADED de modo que las inserciones no solamente atraviesan su propia verificación sino tambien las de las vistas subyacentes. Las siguientes sentencias demuestran las diferencias:

```
ql> 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'
```

La posibilidad de actualización de las vistas puede verse afectada por el valor de la variable del sistema updatable_views_with_limit. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

La sentencia CREATE VIEW fue introducida en MySQL 5.0.1. La cláusula WITH CHECK OPTION fue implementada en MySQL 5.0.2.

INFORMATION_SCHEMA contiene una tabla VIEWS de la cual puede obtenerse información sobre los objetos de las vistas. Consulte Sección 22.1.15, “La tabla INFORMATION_SCHEMA VIEWS”.

21.3. Sintaxis de DROP VIEW

```
DROP VIEW [IF EXISTS]
    nombre_vista [, nombre_vista] ...
    [RESTRICT | CASCADE]
```

DROP VIEW elimina una o más vistas de la base de datos. Se debe poseer el privilegio DROP en cada vista a eliminar.

La cláusula IF EXISTS se emplea para evitar que ocurra un error por intentar eliminar una vista inexistente. Cuando se utiliza esta cláusula, se genera una NOTE por cada vista inexistente. Consulte Sección 13.5.4.22, “Sintaxis de SHOW WARNINGS”.

RESTRICT y CASCADE son ignoradas.

Esta sentencia se introdujo en MySQL 5.0.1.

21.4. Sintaxis de SHOW CREATE VIEW

```
SHOW CREATE VIEW nombre_vista
```

Muestra la sentencia CREATE VIEW que se utilizó para crear la vista.

```
mysql> SHOW CREATE VIEW v;
+-------+----------------------------------------------------+
| Table | Create Table                                       |
+-------+----------------------------------------------------+
| v     | CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b` |
+-------+----------------------------------------------------+
```

Esta sentencia fue introducida en MySQL 5.0.1.

Capítulo 22. La base de datos de información INFORMATION_SCHEMA

Tabla de contenidos

El soporte para INFORMATION_SCHEMA está disponible en MySQL 5.0.2 y posterior. Proporciona acceso a los metadatos de la base de datos.

Metadatos son datos acerca de los datos, tales como el nombre de la base de datos o tabla, el tipo de datos de una columna, o permisos de acceso. Otros términos que a veces se usan para esta información son *diccionario de datos* o *catálogo del sistema* .

Ejemplo:

```
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 |
| 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)
```

Explicación: El comando pide una lista de todas las tablas en la base de datos `db5`, en orden alfabético inverso, mostrando tres informaciones: el nombre de la tabla, su tipo y su motor.

`INFORMATION_SCHEMA` es la base de datos de información, que almacena información acerca de todas las otras bases de datos que mantiene el servidor MySQL . Dentro del `INFORMATION_SCHEMA` hay varias tablas de sólo lectura. En realidad son vistas, no tablas, así que no puede ver ningún fichero asociado con ellas.

Cada usuario MySQL tiene derecho a acceder a estas tablas, pero sólo a los registros que se corresponden a los objetos a los que tiene permiso de acceso.

Ventajas de `SELECT`

El comando `SELECT ... FROM INFORMATION_SCHEMA` es una forma más consistente de proporcionar acceso a la información proporcionada por los comandos `SHOW` que soporta MySQL (`SHOW DATABASES`, `SHOW TABLES`, y así). Usar `SELECT` tiene las siguientes ventajas, en comparación a `SHOW`:

- Cumple las reglas de Codd. Esto es, todo acceso se hace por tabla.
- Nadie necesita aprender una nueva sintaxis. Conocen cómo funciona `SELECT` , sólo necesitan aprender los nombres de los objetos.
- El implementador no tiene que preocuparse de palabras clave.
- Hay millones de variaciones de la salida, en lugar de sólo una. Esto proporciona flexibilidad a las aplicaciones con requerimientos cambiantes acerca de los metadatos que necesitan
- La migración es más fácil ya que todos los otros DBMS funcionan así.

Sin embargo, como `SHOW` es popular entre los empleados y usuarios de MySQL, y como puede ser confuso si desaparece, las ventajas de una sintaxis convencional no es razón para eliminar `SHOW`. De hecho, hay mejoras a `SHOW` en MySQL 5.0. Se describen en Sección 22.2, “Extensiones a las sentencias SHOW”.

Estandars

La implementación de la estructura de tablas para el `INFORMATION_SCHEMA` en MySQL sigue el estándar ANSI/ISO SQL:2003 Parte 11 *Schemata*. Nuestra intención es aproximar el cumplimiento de SQL:2003 característica básica F021 *Basic information schema*.

Los usuarios de SQL Server 2000 (que también sigue el estándar) pueden ver una gran similitud. Sin embargo, MySQL omite varias columnas no relevantes para nuestra implementación, y añade columnas que són específicas de MySQL. Una de estas columnas es `engine` en la tabla `INFORMATION_SCHEMA.TABLES`.

Aunque otros DBMS usan una variedad de nombres, como syscat o system, el nombre estándar es `INFORMATION_SCHEMA`.

En efecto, tenemos una nueva base de datos llamada `INFORMATION_SCHEMA`, aunque no hay necesidad de hacer un fichero llamado así. Es posible seleccionar `INFORMATION_SCHEMA` como base de datos por

defecto con un comando USE , pero la única forma de acceder al contenido de sus tablas es con SELECT. No puede insertar, actualizar o borrar su contenido.

Permisos

No hay diferencia entre el requerimiento de permisos para (SHOW) y para SELECT . En cada caso, debe tener algún permiso de un objeto para consultar información acerca de el mismo.

22.1. Las tablas INFORMATION_SCHEMA

Explicación de las siguientes secciones

En las siguientes secciones, tomamos tablas y columnas del INFORMATION_SCHEMA. Para cada columna, hay tres informaciones:

- "Standard Name" indica el nombre SQL estándar para la columna.
- "SHOW name" indica el nombre equivalente al comando SHOW más parecido, si lo hay.
- "Remarks" proporciona información adicional donde sea aplicable.

Para evitar usar nombres reservados del estándar o de DB2, SQL Server, o Oracle, hemos cambiado los nombres de las columnas marcados como *extensión MySQL*. (Por ejemplo, cambiamos COLLATION a TABLE_COLLATION en la tabla TABLES .) Consulte la lista de palabras reservadas al final del artículo: http://www.dbazine.com/gulutzan5.shtml.

La definición para columnas de caracteres (por ejemplo, TABLES.TABLE_NAME), generalmente es VARCHAR(*N*) CHARACTER SET utf8 donde *N* es como mínimo 64.

Cada sección indica qué comando SHOW es equivalente al SELECT que proporciona información de INFORMATION_SCHEMA, o si no hay tal equivalente.

Nota: Por ahora, hay algunas columnas no presentes y algunas que no funcionan. Estamos trabajando en ello y tratamos de actualizar la documentación tal y como se producen los cambios.

22.1.1. La tabla INFORMATION_SCHEMA SCHEMATA

Un esquema es una base de datos, así que la tabla SCHEMATA proporciona información acerca de bases de datos.

Standard Name	SHOW name	Remarks
CATALOG_NAME	-	NULL
SCHEMA_NAME		base de datos
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

Notas:

- Para SQL_PATH, podemos soportar eventualmente algo en MySQL 5.x. De momento siempre es NULL.
- DEFAULT_COLLATION_NAME se añadió en MySQL 5.0.6.

Los siguientes comandos son equivalentes:

```
SELECT SCHEMA_NAME AS `Database`
  FROM INFORMATION_SCHEMA.SCHEMATA
  [WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
  [LIKE 'wild']
```

22.1.2. La tabla INFORMATION_SCHEMA TABLES

La tabla TABLES proporciona información acerca de las tablas en las bases de datos.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	Extensión MySQL
VERSION	Version	Extensión MySQL
ROW_FORMAT	Row_format	Extensión MySQL
TABLE_ROWS	Rows	Extensión MySQL
AVG_ROW_LENGTH	Avg_row_length	Extensión MySQL
DATA_LENGTH	Data_length	Extensión MySQL
MAX_DATA_LENGTH	Max_data_length	Extensión MySQL
INDEX_LENGTH	Index_length	Extensión MySQL
DATA_FREE	Data_free	Extensión MySQL
AUTO_INCREMENT	Auto_increment	Extensión MySQL
CREATE_TIME	Create_time	Extensión MySQL
UPDATE_TIME	Update_time	Extensión MySQL
CHECK_TIME	Check_time	Extensión MySQL
TABLE_COLLATION	Collation	Extensión MySQL
CHECKSUM	Checksum	Extensión MySQL
CREATE_OPTIONS	Create_options	Extensión MySQL
TABLE_COMMENT	Comment	Extensión MySQL

Notas:

- TABLE_SCHEMA y TABLE_NAME son campos simples en SHOW , por ejemplo Table_in_db1.
- TABLE_TYPE debe ser BASE TABLE o VIEW. Si la tabla es temporal, entonces TABLE_TYPE = TEMPORARY. (No hay vistas temporales, así que no es ambíguo.)
- La columna TABLE_ROWS es NULL si la tabla está en la base de datos INFORMATION_SCHEMA .
- No tneemos nada para el conjunto de caracteres por defecto de la tabla. TABLE_COLLATION se acerca, ya que los nombres de colación comienzan con el nombre del conjunto de caracteres.

Los siguientes comandos son equivalentes:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  [WHERE table_schema = 'db_name']
  [WHERE|AND table_name LIKE 'wild']

SHOW TABLES
  [FROM db_name]
  [LIKE 'wild']
```

22.1.3. La tabla INFORMATION_SCHEMA COLUMNS

La tabla COLUMNS proporciona información acerca de columnas en tablas.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		vea las notas
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_KEY	Key	Extensión MySQL
EXTRA	Extra	Extensión MySQL
COLUMN_COMMENT	Comment	Extensión MySQL

Notas:

- En SHOW, el Type incluye valores de varias columnas COLUMNS distintas.
- ORDINAL_POSITION es necesario ya que puede algún día querer decir ORDER BY ORDINAL_POSITION. Al contrario que SHOW, SELECT no tiene ordenación automática.
- CHARACTER_OCTET_LENGTH debe ser el mismo que CHARACTER_MAXIMUM_LENGTH, excepto para conjuntos de caracteres de múltiples bytes.
- CHARACTER_SET_NAME puede derivarse de Collation. Por ejemplo, si dice SHOW FULL COLUMNS FROM t, y ve en la columna Collation un valor de latin1_swedish_ci, el conjunto de caracteres es lo que hay antes del primer subrayado: latin1.

Los siguientes comandos son casi equivalentes:

```
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]
```

22.1.4. La tabla INFORMATION_SCHEMA STATISTICS

La tabla STATISTICS proporciona información acerca de los índices de las tablas.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Base de datos
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Base de datos
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	Extensión MySQL
PACKED	Packed	Extensión MySQL
NULLABLE	Null	Extensión MySQL
INDEX_TYPE	Index_type	Extensión MySQL
COMMENT	Comment	Extensión MySQL

Notas:

- No hay una tabla estándar para índices. La lista precedente es similar a lo que retorna SQL Server 2000 para sp_statistics, excepto que hemos cambiado el nombre QUALIFIER con CATALOG y OWNER con SCHEMA.

  Claramente, la tabla precedente y la salida de SHOW INDEX se derivan del mismo padre. Así que la correlación está cercana.

Los siguientes comandos son equivalentes:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']

SHOW INDEX
  FROM tbl_name
  [FROM db_name]
```

22.1.5. La tabla INFORMATION_SCHEMA USER_PRIVILEGES

La tabla USER_PRIVILEGES proporciona información acerca de permisos globales. Esta información viene de la tabla de permisos mysql.user .

Standard Name	SHOW name	Remarks
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notas:

- Esta tabla no es estándar. Toma sus valores de la tabla mysql.user .

22.1.6. La tabla INFORMATION_SCHEMA SCHEMA_PRIVILEGES

La tabla SCHEMA_PRIVILEGES proporciona información acerca del esquema de permisos (base de datos). Esta información viene de la tabla de permisos mysql.db .

Standard Name	SHOW name	Remarks
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
TABLE_SCHEMA		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notas:

- Esta tabla no es estándar. Toma sus valores de la tabla mysql.db .

22.1.7. La tabla INFORMATION_SCHEMA TABLE_PRIVILEGES

La tabla TABLE_PRIVILEGES proporciona información de permisos de tablas. Esta información viene de la tabla de permisos mysql.tables_priv .

Standard Name	SHOW name	Remarks
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Los siguientes comandos *no* son equivalentes:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES

SHOW GRANTS ...
```

PRIVILEGE_TYPE puede contener uno (y sólo uno) de estos valores: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW.

22.1.8. La tabla INFORMATION_SCHEMA COLUMN_PRIVILEGES

La tabla COLUMN_PRIVILEGES proporciona información acerca de permisos de columnas. Esta información viene de la tabla de permisos mysql.columns_priv .

Standard Name	SHOW name	Remarks
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notas:

- En la salida de SHOW FULL COLUMNS, los permisos están todos en un campo y en minúsculas, por ejemplo, select,insert,update,references. En COLUMN_PRIVILEGES, hay un registro por permiso, y en mayúsculas.
- PRIVILEGE_TYPE puede contener uno (y sólo uno) de estos valores: SELECT, INSERT, UPDATE, REFERENCES.
- Si el usuario tiene el permiso GRANT OPTION , entonces IS_GRANTABLE debe ser YES. De otro modo, IS_GRANTABLE debe ser NO. La salida no lista GRANT OPTION como permisos separado.

Los siguientes comandos *no* son equivalentes:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES

SHOW GRANTS ...
```

22.1.9. La tabla INFORMATION_SCHEMA CHARACTER_SETS

La tabla CHARACTER_SETS proporciona información acerca de los conjuntos de caracteres disponibles.

Standard Name	SHOW name	Remarks
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPION	Description	Extensión MySQL
MAXLEN	Maxlen	Extensión MySQL

Notas:

- Hemos añadido dos columnas no estándar que se corresponden a Description y Maxlen en la salida de SHOW CHARACTER SET.

Los siguientes comandos son equivalentes:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE name LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

22.1.10. La tabla INFORMATION_SCHEMA COLLATIONS

La tabla COLLATIONS proporciona información acerca de colaciones para cada conjunto de caracteres.

Standard Name	SHOW name	Remarks
COLLATION_NAME	Collation	

Notas:

- Hemos añadido cinco columnas no estándar que se corresponden a Charset, Id, Default, Compiled, y Sortlen de la salida de SHOW COLLATION.

Los siguientes comandos son equivalentes:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE collation_name LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

22.1.11. La tabla INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY

La tabla COLLATION_CHARACTER_SET_APPLICABILITY indica qué conjunto de caracteres es aplicable a cada colación. Las columnas son equivalentes a los dos primeros campos mostrados por SHOW COLLATION.

Standard Name	SHOW name	Remarks
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

22.1.12. La tabla INFORMATION_SCHEMA TABLE_CONSTRAINTS

La tabla TABLE_CONSTRAINTS describe qué tablas tienen restricciones.

Standard Name	SHOW name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notas:

- El valor CONSTRAINT_TYPE puede ser UNIQUE, PRIMARY KEY, o FOREIGN KEY.
- La información UNIQUE y PRIMARY KEY es acerca de lo mismo que obtiene del campo Key_name en la salida de SHOW INDEX cuando el campo Non_unique es 0.
- La columna CONSTRAINT_TYPE puede contener uno de estos valores: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. Esta es una columna CHAR (no ENUM) . El valor CHECK no estará disponible hasta que soportemos CHECK.

22.1.13. La tabla INFORMATION_SCHEMA KEY_COLUMN_USAGE

La tabla KEY_COLUMN_USAGE describe qué columnas clave tienen restricciones.

Standard 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		

Notas:

- Si la restricción es una clave foránea, entonces esta es la columna de la clave foránea, no la columna a la que la clave foránea hace referencia.
- El valor de ORDINAL_POSITION es la posición de la columna en la restricción, no la posición de la columna en la tabla. Las posiciones de columnas se numeran comenzando por 1.
- El valor de POSITION_IN_UNIQUE_CONSTRAINT es NULL para restricciones de claves primárias y únicas. Para restricciones de claves foráneas, es la posición ordinal en la clave de la tabla a la que se referencia.

  Por ejemplo, suponga que hay dos tablas llamadas t1 y t3 con las siguientes definiciones:

```
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;
```

Para estas dos tablas, la tabla KEY_COLUMN_USAGE tiene dos registros:

- Un registro con CONSTRAINT_NAME='PRIMARY', TABLE_NAME='t1', COLUMN_NAME='s3', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=NULL.
- Un registro con CONSTRAINT_NAME='CO', TABLE_NAME='t3', COLUMN_NAME='s2', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=1.

- REFERENCED_TABLE_SCHEMA, REFERENCED_TABLE_NAME, y REFERENCED_COLUMN_NAME se añadieron en MySQL 5.0.6.

22.1.14. La tabla INFORMATION_SCHEMA ROUTINES

La tabla ROUTINES proporciona información acerca de rutinas almacenadas (procedimientos y funciones). La tabla ROUTINES no incluye funciones definidas por el usuario (UDFs) de momento.

La columna llamada "mysql.proc name" indica la columna de la tabla mysql.proc que se corresponde a la columna de la tabla INFORMATION_SCHEMA.ROUTINES, si hay alguna.

Standard Name	mysql.proc name	Remarks
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}
DTD_IDENTIFIER		(descriptor del tipo de datos)
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	Extensión MySQL
ROUTINE_COMMENT	comment	Extensión MySQL
DEFINER	definer	Extensión MySQL

Notas:

- MySQL calcula EXTERNAL_LANGUAGE así:
  - Si mysql.proc.language='SQL', entonces EXTERNAL_LANGUAGE es NULL
  - En caso contrario, EXTERNAL_LANGUAGE es lo que hay en mysql.proc.language. Sin embargo, no tenemos idiomas externos de momento, así que siempre es NULL.

22.1.15. La tabla INFORMATION_SCHEMA VIEWS

La tabla VIEWS proporciona información acerca de las vistas en las bases de datos.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		

Notas:

- Hay un nuevo permiso, SHOW VIEW, sin el cual no puede ver la tabla VIEWS.
- La columna VIEW_DEFINITION tiene la mayoría de lo que ve en el campo Create Table que produce SHOW CREATE VIEW . Ignora las palabras antes de SELECT y tras WITH CHECK OPTION. Por ejemplo, si el comando original era:

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
```

  entonces la definición de la vista es:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- La columna CHECK_OPTION siempre tiene un valor de NONE.
- La columna IS_UPDATABLE es YES si la vista es actualizable, NO si la vista no es actualizable.

22.1.16. La tabla INFORMATION_SCHEMA TRIGGERS

La tabla TRIGGERS proporciona información acerca de disparadores.

Esta tabla se implementó inicialmente en MySQL 5.0.10.

Debe tener el permiso SUPER para ver esta tabla.

Standard 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)

Notas:

- Las columnas TRIGGER_SCHEMA y TRIGGER_NAME contienen el nombre de la base de datos en que se produce el dispardor, y el nombre del disparador, respectivamente.
- La columna EVENT_MANIPULATION contiene uno de los valores 'INSERT', 'DELETE', o 'UPDATE'.
- Como se explica en Capítulo 20, *Disparadores (triggers)*, cada disparador se asocia exactamente con una tabla. Las columnas EVENT_OBJECT_SCHEMA y EVENT_OBJECT_TABLE contienen la base de datos en que ocurre esta tabla, y el nombre de la tabla.
- El comando ACTION_ORDER contiene la posición ordinal de la acción del disparador en la lista de todos los disparadores similares en la misma tabla. Actualmente, este valor siempre es 0, porque no es posible tener más de un disparador con el mismo EVENT_MANIPULATION y ACTION_TIMING en la misma tabla.
- La columna ACTION_STATEMENT contiene el comando a ejecutarse cuando el disparador se invoca. Esto es lo mismo que el texto mostrado en la columna Statement de la salida de SHOW TRIGGERS. Tenga en cuenta que este texto usa codificación UTF-8 .
- La columna ACTION_ORIENTATION siempre contiene el valor 'ROW'.
- La columna ACTION_TIMING contiene uno de los dos valores 'BEFORE' o 'AFTER'.
- Las columnas ACTION_REFERENCE_OLD_ROW y ACTION_REFERENCE_NEW_ROW contienen el antiguo y nuevo identificador de columna, respectivamente. Esto significa que ACTION_REFERENCE_OLD_ROW siempre contiene el valor 'OLD' y ACTION_REFERENCE_NEW_ROW siempre contiene el valor 'NEW'.
- Las siguientes columnas actualmente siempre contiene NULL: TRIGGER_CATALOG, EVENT_OBJECT_CATALOG, ACTION_CONDITION, ACTION_REFERENCE_OLD_TABLE, ACTION_REFERENCE_NEW_TABLE, y CREATED.

Ejemplo, usando el disparador ins_sum definido en Sección 20.3, “Utilización de disparadores”:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\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
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                  CREATED: NULL
1 row in set (1.54 sec)
```

Consulte Sección 13.5.4.20, “Sintaxis de SHOW TRIGGERS”.

22.1.17. Otras tablas INFORMATION_SCHEMA

Pretendemos implementar tablas adicionales INFORMATION_SCHEMA . En particular, sabemos de la necesidad de INFORMATION_SCHEMA.PARAMETERS y INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS.

22.2. Extensiones a las sentencias SHOW

Algunas extensiones de comandos SHOW acompañan la implementación de INFORMATION_SCHEMA:

- SHOW puede usarse para obtener información acerca de la estructura de INFORMATION_SCHEMA mismo.
- Varios comandos SHOW aceptan una cláusula WHERE que proporciona más flexibilidad al especificar qué registros mostrar.

Estas extensiones están disponibles desde MySQL 5.0.3.

INFORMATION_SCHEMA es una base de datos de información, así que su nombre se incluye en la salida de SHOW DATABASES. Similarmente, SHOW TABLES puede usarse con INFORMATION_SCHEMA para obtener una lista de sus tablas:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| SCHEMATA                              |
| TABLES                                |
| COLUMNS                               |
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| ROUTINES                              |
| STATISTICS                            |
| VIEWS                                 |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| SCHEMA_PRIVILEGES                     |
```

```
| TABLE_PRIVILEGES                      |
| COLUMN_PRIVILEGES                     |
| TABLE_CONSTRAINTS                     |
| KEY_COLUMN_USAGE                      |
+---------------------------------------+
```

`SHOW COLUMNS` y `DESCRIBE` pueden mostrar información acerca de las columns en tablas `INFORMATION_SCHEMA` individuales.

Varios comandos `SHOW` se han extendido para permitir cláusulas `WHERE` :

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW KEYS
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW VARIABLES
```

La cláusula `WHERE` , si está presente, se evalúa contra los nombres de columna mostrados por el comando `SHOW`. Por ejemplo, el comando `SHOW COLLATION` produce estas columnas de salida:

Por ejemplo, el comando `SHOW CHARACTER SET` produce estas columnas de salida:

```
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   | ISO 8859-1 West European    | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |

...
```

Para usar una cláusula `WHERE` con `SHOW CHARACTER SET`, se referiría a esos nombres de columna. Como ejemplo, el siguiente comando muestra información acerca de conjuntos de caracteres para los que la colación por defecto contiene la cadena `"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 |
+---------+---------------------------+---------------------+--------+
```

Este comando muestra los conjuntos de caracteres de múltiples bytes:

```
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 |
+---------+---------------------------+---------------------+--------+
```

Capítulo 23. Matemáticas de precisión

Tabla de contenidos

MySQL 5 introduce matemáticas de precisión, esto es, tratamiento numérico que proporciona resultados más precisos y más control sobre valores inválidos que en versiones anteriores de MySQL. La matemática de precisión se basa en dos cambios de implementación:

- La introducción de nuevos modos SQL en MySQL 5.0.2 que controlan lo estricto que es el servidor para aceptar o rechazar datos inválidos.
- La introducción en MySQL 5.0.3 de una biblioteca para aritmética de punto fijo.

Estos cambios tienen varias implicaciones para operaciones numéricas:

- Cálculos más precisos.

  Para números exactos, los cálculos no introducen error en coma flotante. En su lugar, se usa precisión exacta. Por ejemplo, un número tal como .0001 se trata como un valor exacto en lugar de valor aproximado, y sumarlo 10,000 veces produce un resultado de 1, no un valor "cercano" a 1.

- Comportamiento bien definido para el redondeo.

  Para números exactos, el resultado de `ROUND()` depende de sus argumentos, no de factores tales como el comportamiento de la biblioteca C subyacente.

- Independencia de plataforma mejorada.

  Las operaciones con números exactos son los mismos entre distintas plataformas tales como Windows y Unix.

- Control sobre tratamiento de datos inválidos.

  Desbordamiento y división por cero pueden detectarse y tratarse como errores. Por ejemplo, puede tratar un valor que es demasiado grande para una columna como un error en lugar de truncarlo para adaptarlo al rango del tipo de datos. Similarmenet, puede tratar la división por cero como un error en lugar que como una operación que produce un resultado de `NULL`. La elección de qué aproximación seguir se determina mediante la variable de sistema `sql_mode` .

Un resultado importante de estos cambios es que MySQL proporciona un mejor cumplimiento del estándar SQL.

La siguiente discusión cubre varios aspectos de cómo funciona la matemática de precisión (incluyendo posibles incompatibilidades con aplicaciones anteriores). Al final, se dan algunos ejemplos que demuestran cómo MySQL 5 trata operaciones numéricas de forma más precisa que anteriormente.

23.1. Tipos de valores numéricos

El ámbito de matemáticas de precisón para operaciones de valores exactos incluyen tipos de datos precisos (`DECIMAL` y tipos interos) y literales de valores numéricos exactos. Los tipos de datos aproximados y literales numéricos se tratan como valores en coma flotante.

Literales numéricos de valores exactos tienen una parte entera o fraccional, o ambas. Pueden tener signo. Ejemplos: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Literales de valores numéricos aproximados se representan en notación científica con una mantisa y exponente. Una o ambas partes pueden tener signo. Ejemplos: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Números que parecen similares no necesitan ser ambos valores exactos o aproximados. Por ejemplo, `2.34` es un valor exacto (punto fijo), mientras que `2.34E0` es un valor aproximado (coma flotante).

El tipo de datos `DECIMAL` es un tipo de punto fijo y los cálculos son exactos. En MySQL, el tipo `DECIMAL` tiene varios sinónimos: `NUMERIC`, `DEC`, `FIXED`. El tipo entero también es un tipo de valor exacto.

Los tipos de datos `FLOAT` y `DOUBLE` son tipos de coma flotante y los cálculos son aproximados. En MySQL, los tipos sinónimos de `FLOAT` o `DOUBLE` son `DOUBLE PRECISION` y `REAL`.

23.2. Cambios en el tipo de datos `DECIMAL`

En MySQL 5.0.3, se hicieron varios cambios en distintos aspectos del tipo de datos `DECIMAL` (y sus sinónimos):

- Numero máximo de dígitos
- Formato de almacenamiento
- Requerimientos de almacenamiento
- Las extensiones MySQL no estándar al rango superior de columnas `DECIMAL`

Algunos de los cambios provocan posibles incompatibilidades para aplicaciones escritas en versiones antiguas de MySQL. Estas incompatibilidades se muestran durante esta sección.

La sintaxis de declaración para columnas `DECIMAL` sigue siendo `DECIMAL(M,D)`, aunque el rango de valores para los argumentos ha cambiado algo:

- *`M`* es el número máximo de dígitos (la precisión). Tiene un rango de 1 a 64. Introduce una posible incompatibilidad para aplicaciones antiguas, ya que versiones prévias de MySQL permiten el rango de 1 a 254.
- *`D`* es el número de dígitos a la derecha del punto decimal (la escala). Tiene el rango de 0 a 30 y no debe ser mayor que *`M`*.

El valor máximo de 64 para *`M`* significa que los cálculos con valores `DECIMAL` son precisos hasta 64 dígitos. Este límite de 64 dígitos de precisión también se aplica a literales con valor exacto, así que el rango máximo de tales literales es diferente al anterior. (Antes de MySQL 5.0.3, los valores decimales podían tener hasta 254 dígitos. Sin embargo, los cálculos se hacían usando coma flotanto y por lo tanto eran aproximados, no exactos.) Este cambio en el rango de valores literales es otra posible fuente de incompatibilidades para aplicaciones antiguas.

Los valores para columnas `DECIMAL` no se representan como cadenas que requieren un byte por dígito o carácter de signo. En su lugar, se usa un formato binario que empaqueta nueve dítigos decimales en

cuatro bytes. Este cambio del formato de almacenamiento de DECIMAL cambia los requerimientos de almacenamiento también. El almacenamiento para las partes enteras y fraccionales de cada valor se determinan por separado. Cada múltiple de nueve dígitos necesita cuatro bytes, y los dígitos restantes necesitan una fracción de cuatro bytes. Por ejemplo, una columna DECIMAL(18,9) tiene nueve dígitos en cada parte del punto decimal, así que la parte entera y fraccional necesitan cuatro bytes cada una. Una columna DECIMAL(20,10) tiene 10 dígitos en cada lado del punto decimal. Cada parte requiere cuatro bytes para nueve de los dígitos, y un byte para el dígito restante.

El almacenamiento requerido para los dígitos restantes lo da la siguiente tabla:

Dígitos Restantes	**Número de Bytes**
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

Como resultado del cambio de cadena de caracteres a formato numérico para almacenamiento DECIMAL, las columnas DECIMAL no necesitan un carácter '+' o dígito '0' precedente. Antes de MySQL 5.0.3, si insertaba '+0003.1' en una columna DECIMAL(5,1) , se almacenaría como +0003.1. Desde MySQL 5.0.3, se almacena como 3.1. Aplicaciones que confían en el antiguo comportamiento deben modificarse teniendo en cuenta este cambio.

El cambio de formato de almacenamiento también significa que las columnas DECIMAL no soportan la extensión no estándar que permitía valores mayores que el rango implicado por la definición de la columna. Antiguamente, se reservaba un byte para almacenar el carácter de signo. Para valores positivos que no necesitaban byte de signo, MySQL permitía almacenar un byte extra. Por ejemplo, una columna DECIMAL(3,0) debe soportar un rango de al menos -999 a 999, pero MySQL debería permetir almacenar valores de 1000 a 9999 también, usando el byte de signo para almacenar un dígito extra. Esta extensión del rango superior de las columnas DECIMAL no se permite. En MySQL 5.0.3 y posteriores, una columna DECIMAL(*M*,*D*) permite como mucho *M*−*D* dígitos a la izquierda del punto decimal. Esto puede resultar en una incompatibilidad si una aplicación tiene confianza en que MySQL permita valores "demasiado grandes".

El estándar SQL requiere que la precisión de NUMERIC(*M*,*D*) sean exactamente *M* dígitos. Para DECIMAL(*M*,*D*), requiere una precisión de al menos *M* dígitos, pero permite más. En MySQL, DECIMAL(*M*,*D*) y NUMERIC(*M*,*D*) son los mismo y ambos tienen una precisión de exactamente *M* dígitos.

Resumen de incompatibilidades:

La siguiente lista resume las incompatibilidades resultantes de cambios de la columna DECIMAL y tratamiento de valores. Puede usarla como guía cuando al portar aplicaciones antiguas para usar con MySQL 5.0.3 y posteriores.

- Para DECIMAL(*M*,*D*), el máximo *M* es 64, no 254.

- Los cálculos que implican valores decimales con valores exactos son precisos hasta 64 dígitos. Esto es menor que el número máximo de dígitos permitidos antes de MySQL 5.0.3 (254 dígitos), pero la precisión exacta es mayor. Los cálculos anteriormente se hacían con punto flotante de doble precisión, que tiene una precisión de 52 bits (acerca de 15 dígitos decimales).
- La extensión no estándar MySQL del rango superior de columnas `DECIMAL` no se soporta.
- Los caracteres precedentes '`+`' y '`0`' no se almacenan.

23.3. Manejo de expresiones

Con matemáticas de precisión, los números con valores exactos se usan tal y como se dan cuando es posible. Por ejemplo, números en comparaciones se usan exactamente como se dan sin cambiar su valor. En modo SQL estricto, para un `INSERT` en una columna con un tipo exacto (`DECIMAL` o entero), se inserta un número con su valor exacto si está dentro del rango de la columna. Cuando se recibe, el valor debe ser el mismo que se insertó. (Sin modo estricto, se permite truncar para `INSERT`.)

El tratamiento de expresiones numéricas depende de qué clase de valores contiene la expresión:

- Si hay presente algún valor aproximado, la expresión es aproximada y se evalúa usando aritmética de punto flotante.
- Si no hay presente ningún valor aproximado, la expresión contiene sólo valores exactos. Si algún valor exacto contiene una parte fraccional (un valor a continuación del punto decimal), la expresión se evalúa usando aritmética exacta `DECIMAL` y una precisión de 64 dígitos. ("Exacto" esta sujeto a los límites de lo que puede representarse en binario. `1.0/3.0` puede representarse como `.333...` con un número finito de dígitos, no como "exactamente un tercio", así que `(1.0/3.0)*3.0` no se evalúa como "exactamente 1.0.")
- En otro caso, la expresión contiene sólo valores enteros. La expresión es exacta y evaluada usando aritmética entera y tiene la misma precisión que `BIGINT` (64 bits).

Si una expresión numérica contiene cualquier cadena de caracteres, se convierten a valores de coma flotante y doble precisión y la expresión es aproximada.

Las inserciones en columnas numéricas están afectadas por el modo SQL, controlada por la varible de sistema sql_mode. (Consulte Sección 1.7.2, “Selección de modos SQL”.) La siguiente discusión menciona el modo estricto (seleccionado por los valores de modo `STRICT_ALL_TABLES` o `STRICT_TRANS_TABLES`) y `ERROR_FOR_DIVISION_BY_ZERO`. Para activar todas las restricciones, puede usar el modo `TRADITIONAL`, que incluye tanto el modo estricto como `ERROR_FOR_DIVISION_BY_ZERO`:

```
mysql> SET sql_mode='TRADITIONAL';
```

Si se inserta un número en una columna de tipo exacto (`DECIMAL` o entero), debe insertarse con su valor exacto si está dentro del rango de la columna.

Si el valor tiene demasiados dígitos en la parte fraccional, se redondea y se genera una advertencia. El redondeo se hace como se describe en "Comportamiento del redondeo".

Si el valor tiene demasiados dígitos en la parte entera, es demasiado grande y se trata como se explica a continuación:

- Si el modo estricto no está activado, el valor se trunca al valor legal más cercano y se genera una advertencia.

- Si el modo estricto está activo, se genera un error de desbordamiento.

Desbordamiento inferior no se deteca, así que su tratamiento no está definido.

Por defecto, la división por cero produce un resultado de `NULL` y ninguna advertencia. Con el modo SQL `ERROR_FOR_DIVISION_BY_ZERO` activado, MySQL trata la división por cero de forma distinta:

- Si el modo estricto no está activo, aparece una advertencia.
- Si el modo estricto está activo, las inserciones y actualizaciones con divisiones por cero están prohibidas y ocurre un error.

En otras palabras, inserciones y actualizaciones que impliquen expresiones que realizan divisiones por cero pueden tratarse como errores, pero esto requiere `ERROR_FOR_DIVISION_BY_ZERO` además del modo estricto.

Suponga que tenemos este comando:

```
INSERT INTO t SET i = 1/0;
```

Esto es lo que ocurre al combinar modo estricto y `ERROR_FOR_DIVISION_BY_ZERO` :

`sql_mode` Valor	Resultado
`''`	No advertencia, no error, i es `NULL`
strict	No advertencia, no error, i es `NULL`
`ERROR_FOR_DIVISION_BY_ZERO`	Advertencia, no error, i es `NULL`
strict,`ERROR_FOR_DIVISION_BY_ZERO`	Error, no se inserta el registro

Para inserciones de cadenas de caracteres en columnas numéricas, las conversiones de cadenas a números se tratan como se muestra si la cadena tiene contenido no numérico:

- Una cadena que no comienza con un número no puede usarse como número y produce un error en modo estricto, o una advertencia en otro caso. Esto incluye la cadena vacía.
- Una cadena que comienza con un número puede convertirse, pero se trunca la parte no numérica final. Esto produce un error en modo estricto, o una advertencia en otro caso.

23.4. Cómo se redondea

Esta sección discute el redondeo de la matemática precisa para la función `ROUND()` y para inserciones en columnas `DECIMAL` .

La función `ROUND()` redondea de forma distinta dependiendo de si su argumento es exacto o aproximada:

- Para valores exactos, `ROUND()` usa la regla "redondeo al alza": Un valor con parte fraccional de .5 o superior se redondea al siguiente entero si es positivo o al anterior entero si es negativo. (En otras palabras, siempre se redondea alejándose del cero.) Un valor con una parte fraccional menor que .5 se redondea al anterior valor entero si es positivo o al siguiente entero si es negativo.
- Para números aproximados, el resultado depende de la biblioteca C. En muchos sistemas, esto significa que `ROUND()` usa la regla "redondeo al número par más próximo": Un valor con un parte fraccional se redondea al siguiente entero par.

El siguiente ejemplo muestra cómo difiere el redondeo para valores exactos y aproximados:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

Para inserciones en una columna `DECIMAL`, el objetivo es un tipo de datos exacto, así que el redondea usa "redondeo al alza" independientemente de si el valor a ser insertado es exacto o aproximado:

```
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    |
+------+
```

23.5. Ejemplos de matemáticas de precisión

Esta sección proporciona algunos ejemplos que muestran cómo la matemática precisa mejora los resultados de consultas en MySQL 5 comparados con versiones anteriores.

Ejemplo 1. Los números se usan con su valor exacto tal y como se da cuando es posible.

Antes de MySQL 5.0.3, los números tratados como valores en coma flotante producen valores inexactos:

```
mysql> SELECT .1 + .2 = .3;
+--------------+
| .1 + .2 = .3 |
+--------------+
|            0 |
+--------------+
```

Desde MySQL 5.0.3, los números se usan tal y como se dan cuando es posible:

```
mysql> SELECT .1 + .2 = .3;
+--------------+
| .1 + .2 = .3 |
+--------------+
|            1 |
+--------------+
```

Sin embargo, para valores en coma flotante, todavía ocurre la inexactitud:

```
mysql> SELECT .1E0 + .2E0 = .3E0;
+--------------------+
| .1E0 + .2E0 = .3E0 |
+--------------------+
|                  0 |
+--------------------+
```

Otra forma de ver la diferencia en el tratamiento de valores aproximados y exactos es añadir un pequeño número en una suma muchas veces. Considere el siguiente procedimiento aproximado, que añade .0001 a una variable 1000 veces.

```
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;
```

La suma de ambos `d` y `f` lógicamente debe ser 1, pero eso es cierto sólo para cálculos decimales. El cálculo de coma flotante introduce pequeños errores:

```
+--------+------------------+
| d      | f                |
+--------+------------------+
| 1.0000 | 0.99999999999991 |
+--------+------------------+
```

Ejemplo 2. La multiplicación se hace con la escala requerida por el estándar SQL. Esto es, para dos números *`X1`* y *`X2`* con escala *`S1`* y *`S2`*, la escala del resultado es *`S1 + S2`*.

Antes de MySQL 5.0.3, esto es lo que ocurre:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
|      0.00 |
+-----------+
```

El valor mostrado es incorrecto. El valor se calcula correctamente en este caso, pero no se muestra en la escala requerida. Para comprobar que el valor calculado realmente es .0001, pruebe:

```
mysql> SELECT .01 * .01 + .0000;
+-------------------+
| .01 * .01 + .0000 |
+-------------------+
|            0.0001 |
+-------------------+
```

Desde MySQL 5.0.3, la escala mostrada es correcta:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
| 0.0001    |
+-----------+
```

Ejemplo 3. El comportamiento del redondeo está bien definido.

Antes de MySQL 5.0.3, el comportamiento para el redondeo (por ejemplo con la función `ROUND()`) depende de la implementación de la biblioteca C subyacente. Esto provoca inconsistencias entre plataformas. Por ejemplo, puede obtener un valor distinto en Windows y en Linux, o un valor distinto en máquinas x86 y PowerPc.

Desde MySQL 5.0.3, el redondeo se realiza así:

El redondeo para columnas `DECIMAL` y de valor exacto usa la regla de "redondeo hacia arriba". Los valoeres con una parte fraccional de .5 o mayor se redondean al entero más cercano y más lejano al cero, como se muestra aquí:

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+------------+-------------+
| ROUND(2.5) | ROUND(-2.5) |
+------------+-------------+
| 3          | -3          |
+------------+-------------+
```

El redondeo de valores en coma flotante todavía usa la biblioteca C, que en muchos sistemas usa la regla "redondeo al número par más cercano". Los valores con cualquier parte fraccional se redondean al entero par más cercano:

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+--------------+---------------+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+--------------+---------------+
|            2 |            -2 |
+--------------+---------------+
```

Ejemplo 4. Para inserciones en tablas, un valor demasiado grande provoca un desbordamiento y un error, no se trunca a un valor legal. (Esto requiere modo estricto.)

Antes de MySQL 5.0.2, se truncaba a un valor legal:

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

Desde MySQL 5.0.2, ocurre un desbordamiento si el modo estricto está activado:

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.10 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)
```

Ejemplo 5. Para inserciones en tablas, la división por cero causa un error, no un resultado `NULL`. (Esto requiere modo estricto y `ERROR_FOR_DIVISION_BY_ZERO`.)

Antes de MySQL 5.0.2, la división por cero tiene un resultado de `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 (0.06 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
| NULL |
+------+
1 row in set (0.01 sec)
```

Desde MySQL 5.0.2, la división por cero es un error si el modo SQL apropiado está activado:

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
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)
```

Ejemplo 6. En MySQL 4, literales de valores aproximados y exactos se convierten en valores de coma flotante y doble precisión:

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;

mysql> DESCRIBE t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | double(3,1) |      |     | 0.0     |       |
| b     | double      |      |     | 0       |       |
+-------+-------------+------+-----+---------+-------+
```

En MySQL 5, el literal de valor aproximado todavía se convierte en un valor de coma flotante, pero el literal de valor exacto se trata como `DECIMAL`:

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;

mysql> DESCRIBE t;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| a     | decimal(3,1) | NO   |     | 0.0     |       |
| b     | double       | NO   |     | 0       |       |
+-------+--------------+------+-----+---------+-------+
```

Ejemplo 7. Si el argumento de una función agregada es un tipo numérico exacto, el resultado debe serlo también, con una escala al menos igual a la del argumento. El resultado no debe siempre ser un valor double.

Considere estos comandos:

```
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;
```

Resultado antes de MySQL 5.0.3:

```
mysql> DESCRIBE y;
+--------+--------------+------+-----+---------+-------+
| 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    |       |
+--------+--------------+------+-----+---------+-------+
```

El resultado es un double independientemente del tipo del argumento.

Resultado desde MySQL 5.0.3:

```
mysql> DESCRIBE y;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| AVG(i) | decimal(64,0) | YES  |     | NULL    |       |
| AVG(d) | decimal(64,0) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

El resultado es un double sólo para el argumento de coma flotante. El resultado es un tipo exacto para argumentos con tipos exactos.

Capítulo 24. APIs de MySQL

Tabla de contenidos

Este capítulo describe las APIs (Application Programming Interface, o Interfaz de Programación de Aplicaciones) disponibles para MySQL, dónde se las puede obtener, y cómo utilizarlas. La API C es tratada en mayor medida, ya que fue desarrollada por el equipo de MySQL, y es la base para la mayoría de las otras APIs.

24.1. Utilidades para el desarrollo de programas MySQL

Esta sección describe algunas utilidades que pueden ser de utilidad al desarrollar programas para MySQL.

- `msql2mysql`

  Un script del shell que convierte programas `mSQL` a MySQL. No contempla todos los casos, pero proporciona un buen punto de partida para realizar la conversión.

- `mysql_config`

  Un script del shell que establece los valores de opciones necesarios al compilar programas MySQL.

24.1.1. msql2mysql —

En un principio, la API C de MySQL se desarrolló con el fin de que fuera muy similar a la existente para el sistema de bases de datos mSQL. Debido a esto, los programas mSQL a menudo pueden convertirse de manera relativamente sencilla para ser empleados con MySQL, cambiando los nombres de las funciones de la API C.

La utilidad `msql2mysql` lleva a cabo la conversión de llamadas a funciones de la API C de mSQL hacia sus equivalentes en MySQL. `msql2mysql` trabaja directamente sobre el fichero original, por lo que debe hacerse una copia del mismo antes de intentar la conversión. Por ejemplo, `msql2mysql` puede emplearse de esta manera:

```
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 Sección 8.12, “La utilidad replace de cambio de cadenas de caracteres”.

24.1.2. mysql_config —

`mysql_config` provides you with useful information for compiling your MySQL client and connecting it to MySQL.

`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.

- `--include`

  Compiler options to find MySQL include files. (Note that normally you would use `--cflags` instead of this option.)

- `--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.

- `--port`

  The default TCP/IP port number, defined when configuring MySQL.

- `--socket`

  The default Unix socket file, defined when configuring MySQL.

- `--version`

  Version number and version 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
                    -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lz -lcrypt -lnsl -lm -lpthread]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [4.0.16]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                    -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use `mysql_config` within a command line to include the value that it displays for a particular option. For example, to compile a MySQL client program, use `mysql_config` as follows:

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

When you use `mysql_config` this way, be sure to invoke it within backtick (`` ` ``) characters. That tells the shell to execute it and substitute its output into the surrounding command.

24.2. La API C de MySQL

The C API code is distributed with MySQL. It is included in the `mysqlclient` library and allows C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients. You can find these in the `clients` directory in the MySQL source distribution.

Most of the other client APIs (all except Connector/J) use the `mysqlclient` library to communicate with the MySQL server. 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. See Capítulo 8, *Programas cliente y utilidades MySQL*, for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (the maximum is 16MB). Because buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in itself cause more resources to be used. This size check is mostly a check for erroneous queries 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 thread's communication buffer is 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 client's default maximum is 16MB, but the default maximum in the server is

1MB. You can increase this by changing the value of the `max_allowed_packet` parameter when the server is started. See Sección 7.5.2, “Afinar parámetros del servidor”.

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.

For programming with threads, see Sección 24.2.15, “Cómo hacer un cliente multihilo”. For creating a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see Sección 24.2.16, “libmysqld, la biblioteca del servidor MySQL incrustado (embedded)”.

24.2.1. Tipos de datos de la API C

- `MYSQL`

  This structure represents a handle to one database connection. It is used for almost all MySQL functions. You should not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be 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 information about a field, such as the field's name, type, and size. Its members are described in more detail here. 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`.

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));
```

The `MYSQL_FIELD` structure contains the members listed here:

- `char * name`

  The name of the field, as a null-terminated string.

- `char * table`

  The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is an empty string.

- `char * def`

  The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `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_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` field
`MYSQL_TYPE_VAR_STRING`	`VARCHAR` 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_NULL	NULL-type field
MYSQL_TYPE_CHAR	Deprecated; use MYSQL_TYPE_TINY instead

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");
```

- unsigned int length

  The width of the field, as specified in the table definition.

- unsigned int max_length

  The maximum width of the field for the result set (the length 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.

- unsigned int flags

  Different bit-flags for the field. The flags value may have zero or more of the following bits set:

Flag Value	Flag Description
NOT_NULL_FLAG	Field can't 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 non-unique 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 (deprecated)
SET_FLAG	Field is a SET (deprecated)
BLOB_FLAG	Field is a BLOB or TEXT (deprecated)
TIMESTAMP_FLAG	Field is a TIMESTAMP (deprecated)

Use of the BLOB_FLAG, ENUM_FLAG, SET_FLAG, and TIMESTAMP_FLAG flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test field->type against MYSQL_TYPE_BLOB, MYSQL_TYPE_ENUM, MYSQL_TYPE_SET, or MYSQL_TYPE_TIMESTAMP instead.

The following example illustrates a typical use of the flags value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the following convenience macros 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.

24.2.2. Panorámica de funciones de la API C

The functions available in the C API are summarized here and described in greater detail in a later section. See Sección 24.2.3, “Descripción de funciones de la API C”.

Function	Description
mysql_affected_rows()	Returns the number of rows changed/deleted/inserted by the last `UPDATE`, `DELETE`, or `INSERT` query.
mysql_change_user()	Changes user and database on an open connection.
mysql_charset_name()	Returns the name of the default character set for the connection.
mysql_close()	Closes a server connection.
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.
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_seek()	Puts the column cursor on a specified column.
mysql_field_count()	Returns the number of result columns for the most recent statement.
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_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_server_version()	Returns version number of server as an integer (new in 4.1).
mysql_get_proto_info()	Returns the protocol version used by the connection.
mysql_get_server_info()	Returns the server version number.
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 MySQL C API library.
mysql_library_init()	Initialize 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_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_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 charset of the connection.
mysql_real_query()	Executes an SQL query specified as a counted string.
mysql_reload()	Tells the server to reload the grant tables.
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 embedded server library.
mysql_server_init()	Initialize embedded server library.
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_stat()	Returns the server status as a string.
mysql_store_result()	Retrieves a complete result set to the client.
mysql_thread_id()	Returns the current thread ID.
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.
mysql_commit()	Commits the transaction.
mysql_rollback()	Rolls back the transaction.
mysql_autocommit()	Toggles autocommit mode on/off.
mysql_more_results()	Checks whether any more results exist.
mysql_next_result()	Returns/initiates the next result in multiple-statement executions.

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling `mysql_library_init()`. The library can be either the `mysqlclient` C client library or the `mysqld` embedded server library, depending on whether the application was linked with the `-libmysqlclient` or `-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()`.

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 don't 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 4.1.10 and 5.0.3. These 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 library no matter whether the application uses the `mysqlclient` or `mysqld` library. For older versions of MySQL, you can call `mysql_server_init()` and `mysql_server_end()` instead.

If you like, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary.

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 hostname, username, 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 strictly older than 5.0.3, of `0` in newer

versions. A value of `1` for this flag indicates, in the event that a query cannot be performed because of a lost connection, to try reconnecting to the server before giving up. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL queries 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`. They should be treated 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. Client programs should choose the approach that is most appropriate for their requirements. 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 `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 don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set 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 queries (retrieving rows only as necessary) without knowing whether or not the query 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 query 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 query returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, etc.), and was not expected to return rows. If `mysql_field_count()` is non-zero, the

query should have returned rows, but didn't. This indicates that the query 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()` allow you to obtain information about the fields that make up the result set (the number of fields, their names and types, etc.). 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, allowing you to determine when an error occurred and what it was.

24.2.3. Descripción de funciones de la API C

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 non-zero to indicate an error. Note that “non-zero” 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 ...
```

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()`.

24.2.3.1. `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

Returns the number of rows changed by the last `UPDATE`, deleted by the last `DELETE` or inserted by the last `INSERT` statement. May be called immediately after `mysql_query()` for `UPDATE`, `DELETE`, or `INSERT` statements. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

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

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%ld products updated",(long) mysql_affected_rows(&mysql));
```

If you specify the flag `CLIENT_FOUND_ROWS` when connecting to `mysqld`, `mysql_affected_rows()` returns the number of rows matched by the `WHERE` statement for `UPDATE` statements.

Note that when you use a `REPLACE` command, `mysql_affected_rows()` returns 2 if the new row replaced an old row. This is because in this case one row was inserted after the duplicate was deleted.

If you use `INSERT ... ON DUPLICATE KEY UPDATE` to insert a row, `mysql_affected_rows()` returns 1 if the row is inserted as a new row and 2 if an existing row is updated.

24.2.3.2. `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 do not include an explicit database specifier.

This function was introduced in MySQL 3.23.3.

`mysql_change_user()` fails if the connected user cannot be authenticated or doesn't have permission to use the database. In this case the user and database are not changed

The `db` parameter may be set to `NULL` if you don't want to have a default database.

Starting from MySQL 4.0.6 this command always performs a `ROLLBACK` of any active transactions, closes all temporary tables, unlocks all locked tables and resets the state as if one had done a new connect. This happens even if the user didn't change.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`.

- `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.

- `ER_UNKNOWN_COM_ERROR`

  The MySQL server doesn't implement this command (probably an old server).

- `ER_ACCESS_DENIED_ERROR`

  The user or password was wrong.

- `ER_BAD_DB_ERROR`

  The database didn't 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));
}
```

24.2.3.3. `mysql_character_set_name()`

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set for the current connection.

Return Values

The default character set

Errors

None.

24.2.3.4. 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.

Errors

None.

24.2.3.5. mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Description

This function is deprecated. It is preferable to 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 can't 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()`.

24.2.3.6. mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

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 if the database was created successfully. Non-zero 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));
}
```

24.2.3.7. `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 and should be 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.

24.2.3.8. `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 Sección D.1, “Depurar un servidor MySQL”. See Sección D.2, “Depuración de un cliente MySQL”.

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");
```

24.2.3.9. 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 if the database was dropped successfully. Non-zero 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_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

24.2.3.10. mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write some debug information to the log. For this to work, the connected user must have the SUPER privilege.

Return Values

Zero if the command was successful. Non-zero 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.

24.2.3.11. 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 non-zero value if the end of the result set was reached and zero if an error occurred.

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 if no error occurred. Non-zero 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));
}
```

24.2.3.12. `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`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`. The server error codes also are listed at Capítulo 26, *Manejo de errores en MySQL*.

Note that some functions like `mysql_fetch_row()` don't 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.

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

24.2.3.13. `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 didn't 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_errno()`, the following two tests are equivalent:

```
if(mysql_errno(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0] != '\0')
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently you can choose error messages in several different languages. See Sección 5.9.2, “Escoger el idioma de los mensajes de error”.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

24.2.3.14. `mysql_escape_string()`

You should use `mysql_real_escape_string()` instead!

This function 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 charset setting.

24.2.3.15. `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 doesn't 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);
}
```

24.2.3.16. `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);
}
```

24.2.3.17. 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. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

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);
}
```

24.2.3.18. 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.

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 characters). `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]);
    }
}
```

24.2.3.19. `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

Note that error is not reset between calls to `mysql_fetch_row()`

- `CR_SERVER_LOST`

  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");
}
```

24.2.3.20. `mysql_field_count()`

```
unsigned int mysql_field_count(MYSQL *mysql)
```

If you are using a version of MySQL earlier than Version 3.22.24, you should use `unsigned int mysql_num_fields(MYSQL *mysql)` instead.

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 non-empty result. This allows 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 Sección 24.2.13.1, “¿Por qué `mysql_store_result()` a veces devuelve `NULL` después de que `mysql_query()` haya dado un resultado?”.

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_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`.

24.2.3.21. `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.

24.2.3.22. `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.

Errors

None.

24.2.3.23. 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()`, etc. 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.

24.2.3.24. 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

```
if (!mysql_set_character_set_name(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    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: %s\n", cs->mbminlen);
    printf("multi byte character max. length: %s\n", cs->mbmaxlen);
}
```

24.2.3.25. mysql_get_client_info()

```
char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

24.2.3.26. mysql_get_client_version()

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the client library version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level, and `ZZ` is the version number within the release level. For example, a value of `40102` represents a client library version of `4.1.2`.

This function was added in MySQL 4.0.16.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

24.2.3.27. mysql_get_host_info()

```
char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server hostname.

Return Values

A character string representing the server hostname and the connection type.

Errors

None.

24.2.3.28. 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.

24.2.3.29. mysql_get_server_info()

```
char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return Values

A character string that represents the server version number.

Errors

None.

24.2.3.30. mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns the version number of the server as an integer.

This function was added in MySQL 4.1.0.

Return Values

A number that represents the MySQL server version in this format:

```
major_version*10000 + minor_version *100 + sub_version
```

For example, 4.1.2 is returned as 40102.

This function is useful in client programs for quickly determining whether some version-specific server capability exists.

Errors

None.

24.2.3.31. mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Description

This function is used to create a legal SQL string that you can use in a SQL statement. See Sección 9.1.1, “Cadenas de caracteres”.

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

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 character.

The return value can be placed into an SQL statement using either `0x`*`value`* or `X'`*`value`*`'` format. However, the return value does not include the `0x` or `X'...'`. The caller must supply whichever of those is desired.

`mysql_hex_string()` was added in MySQL 4.0.23 and 4.1.8.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What's this",11);
end = strmov(end,",0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*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 `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

24.2.3.32. `mysql_info()`

```
char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed query, 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 query, as described here. The numbers are illustrative only; the string contains values appropriate for the query.

- `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`

Note that `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 query. `NULL` if no information is available for the query.

Errors

None.

24.2.3.33. `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.

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.

24.2.3.34. `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.

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 non-special 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 **last** explicit value inserted into the `AUTO_INCREMENT` column.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(expr)` into any column.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by updating any column to `LAST_INSERT_ID(expr)`.
- The value of `mysql_insert_id()` is not affected by statements such as `SELECT` that return a result set.
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

Note that `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 affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

See Sección 12.9.3, “Funciones de información”.

Also note that the value of the SQL `LAST_INSERT_ID()` function always contains the most recently generated `AUTO_INCREMENT` value, and is not reset between statements because the value of that function is maintained in the server. Another difference is that `LAST_INSERT_ID()` is not updated if you set an `AUTO_INCREMENT` column to a specific non-special value.

The reason for the difference 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 a little more exact information of what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

24.2.3.35. `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

Return Values

Zero for success. Non-zero 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.

24.2.3.36. `mysql_library_init()`

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

This is a synonym for the `mysql_server_init()` function. It was added in MySQL 4.1.10 and 5.0.3.

See Sección 24.2.2, “Panorámica de funciones de la API C” for usage information.

24.2.3.37. `mysql_library_end()`

```
void mysql_library_end(void)
```

Description

This is a synonym for the `mysql_server_end()` function. It was added in MySQL 4.1.10 and 5.0.3.

See Sección 24.2.2, “Panorámica de funciones de la API C” for usage information.

24.2.3.38. `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]`.

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.

24.2.3.39. `mysql_list_fields()`

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns a result set consisting of field names 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]`.

Note that it's recommended that you 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`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

24.2.3.40. 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.

24.2.3.41. mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

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.

24.2.3.42. `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

Or:

```
unsigned int mysql_num_fields(MYSQL *mysql)
```

The second form doesn't work on MySQL 3.22.24 or newer. To pass a `MYSQL*` argument, you must use `unsigned int mysql_field_count(MYSQL *mysql)` instead.

Description

Returns the number of columns in a result set.

Note that 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 non-empty result. This allows the client program to take proper action without knowing whether or not the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Sección 24.2.13.1, “¿Por qué `mysql_store_result()` a veces devuelve `NULL` después de que `mysql_query()` haya dado un resultado?”.

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)
            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)` is = 0. This happens only if something went wrong.

24.2.3.43. `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.

Return Values

The number of rows in the result set.

Errors

None.

24.2.3.44. `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.

`mysql_options()` should be called 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, then `arg` should point to the value of the integer.

Possible option values:

Option	Argument Type	Function
`MYSQL_INIT_COMMAND`	`char *`	Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.
`MYSQL_OPT_COMPRESS`	Not used	Use the compressed client/server protocol.
`MYSQL_OPT_CONNECT_TIMEOUT`	`unsigned int *`	Connect timeout in seconds.
`MYSQL_OPT_GUESS_CONNECTION`	Not used	For an application linked against `libmysqld`, this allows the library to guess whether to use the embedded server or a remote server. “Guess” means that if the hostname 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 `libmysqlclient`.
`MYSQL_OPT_LOCAL_INFILE`	optional pointer to uint	If no pointer is given or if pointer points to an `unsigned int != 0` the command `LOAD LOCAL INFILE` is enabled.
`MYSQL_OPT_NAMED_PIPE`	Not used	Use named pipes to connect to a MySQL server on NT.
`MYSQL_OPT_PROTOCOL`	`unsigned int *`	Type of protocol to use. Should be one of the enum values of `mysql_protocol_type` defined in `mysql.h`. New in 4.1.0.
`MYSQL_OPT_READ_TIMEOUT`	`unsigned int *`	Timeout for reads from server (works currently only on Windows on TCP/IP connections). New in 4.1.1.
`MYSQL_OPT_SET_CLIENT_IP`	`char *`	For an application linked against linked against `libmysqld` (with `libmysqld` 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 `libmysqlclient`.
`MYSQL_OPT_USE_EMBEDDED_CONNECTION`	Not used	For an application linked against `libmysqld`, this forces the use of the embedded server for the connection.

		This option is ignored for applications linked against `libmysqlclient`.
`MYSQL_OPT_USE_REMOTE_CONNECTION`	Not used	For an application linked against `libmysqld`, this forces the use of a remote server for the connection. This option is ignored for applications linked against `libmysqlclient`.
`MYSQL_OPT_USE_RESULT`	Not used	This option is new in 4.1.1, but is unused.
`MYSQL_OPT_WRITE_TIMEOUT`	`unsigned int *`	Timeout for writes to server (works currently only on Windows on TCP/IP connections). New in 4.1.1.
`MYSQL_READ_DEFAULT_FILE`	`char *`	Read options from the named option file instead of from `my.cnf`.
`MYSQL_READ_DEFAULT_GROUP`	`char *`	Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`.
`MYSQL_REPORT_DATA_TRUNCATION`	`my_bool *`	Enable or disable reporting of data truncation errors for prepared statements via `MYSQL_BIND.error`. (Default: disabled) New in 5.0.3.
`MYSQL_SECURE_AUTH`	`my_bool*`	Whether to connect to a server that does not support the new 4.1.1 password hashing. New in 4.1.1.
`MYSQL_SET_CHARSET_DIR`	`char*`	The pathname to the directory that contains character set definition files.
`MYSQL_SET_CHARSET_NAME`	`char*`	The name of the character set to use as the default character set.
`MYSQL_SHARED_MEMORY_BASE_NAME`	`char*`	Named of shared memory object for communication to server. Should be same as the option `-shared-memory-base-name` used for the mysqld server you want's to connect to. New in 4.1.0.

Note that 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
`connect-timeout`	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
`compress`	Use the compressed client/server protocol.
`database`	Connect to this database if no database was specified in the connect command.
`debug`	Debug options.
`disable-local-infile`	Disable use of `LOAD DATA LOCAL`.

`host`	Default hostname.
`init-command`	Command to execute when connecting to MySQL server. Will automatically be re-executed when reconnecting.
`interactive-timeout`	Same as specifying `CLIENT_INTERACTIVE` to `mysql_real_connect()`. See Sección 24.2.3.47, “`mysql_real_connect()`”.
`local-infile[=(0|1)]`	If no argument or argument != 0 then enable use of `LOAD DATA LOCAL`.
`max_allowed_packet`	Max size of packet client can read from server.
`multi-results`	Allow multiple result sets from multiple-statement executions or stored procedures. New in 4.1.1.
`multi-statements`	Allow the client to send multiple statements in a single string (separated by '`;`'). New in 4.1.9.
`password`	Default password.
`pipe`	Use named pipes to connect to a MySQL server on NT.
`protocol={TCP | SOCKET | PIPE | MEMORY}`	The protocol to use when connecting to server (New in 4.1)
`port`	Default port number.
`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 (default is "MYSQL"). New in MySQL 4.1.
`socket`	Default socket file.
`user`	Default user.

Note that `timeout` has been replaced by `connect-timeout`, but `timeout` still works for a while.

For more information about option files, see Sección 4.3.2, “Usar ficheros de opciones”.

Return Values

Zero for success. Non-zero if you used an unknown option.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
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 the client to use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

24.2.3.45. `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down, an automatic reconnection is attempted.

This function 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.

Return Values

Zero if the connection to the server is alive. Non-zero if an error occurred. A non-zero 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.

24.2.3.46. `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *query)
```

Description

Executes the SQL query pointed to by the null-terminated string `query`. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (';') or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Sección 24.2.9, “Tratamiento por parte de la API C de la ejecución de múltiples consultas”.

`mysql_query()` cannot be used for queries that contain binary data; you should use `mysql_real_query()` instead. (Binary data may contain the '`\0`' character, which `mysql_query()` interprets as the end of the query string.)

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See Sección 24.2.3.20, “`mysql_field_count()`”.

Return Values

Zero if the query was successful. Non-zero 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.

24.2.3.47. `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:

- The first parameter should be the address of an existing `MYSQL` structure. Before calling `mysql_real_connect()` you must call `mysql_init()` to initialize the `MYSQL` structure. You can change a lot of connect options with the `mysql_options()` call. See Sección 24.2.3.44, “mysql_options()”.

- The value of `host` may be either a hostname or an IP address. If `host` is `NULL` or the string `"localhost"`, a connection to the local host is assumed. If the OS supports sockets (Unix) or named pipes (Windows), they are used instead of TCP/IP to connect to the server.

- 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 username must be specified explicitly. See Sección 25.1.3.2, “Configuring a Connector/ODBC DSN on Windows”.

- 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 allows the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether or not 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.

- `db` is the database name. If `db` is not `NULL`, the connection sets the default database to this value.

- 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 that should be used. 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 in very special circumstances:

Flag Name	Flag Description
CLIENT_COMPRESS	Use compression protocol.
CLIENT_FOUND_ROWS	Return the number of found (matched) rows, not the number of affected rows.
CLIENT_IGNORE_SPACE	Allow spaces after function names. Makes all functions names reserved words.
CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout seconds) of inactivity 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_STATEMENTS	Tell the server that the client may send multiple statements in a single string (separated by ';'). If this flag is not set, multiple-statement execution is disabled. New in 4.1.
CLIENT_MULTI_RESULTS	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This is automatically set if CLIENT_MULTI_STATEMENTS is set. New in 4.1.
CLIENT_NO_SCHEMA	Don't allow the *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	The client is an ODBC client. This changes mysqld to be more ODBC-friendly.
CLIENT_SSL	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the client library.

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.

- `CR_UNKNOWN_HOST`

  Failed to find the IP address for the hostname.

- `CR_VERSION_ERROR`

  A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the `--old-protocol` option.

- `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 non-standard way.

Note that upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API strictly older than 5.0.3, of `0` in newer versions. A value of `1` for this flag indicates, in the event that a query cannot be performed because of a lost connection, to try reconnecting to the server before giving up.

24.2.3.48. `mysql_real_escape_string()`

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

Note that `mysql` must be a valid, open connection. This is needed because the escaping depends on the character set in use by the server.

Description

This function is used to create a legal SQL string that you can use in a SQL statement. See Sección 9.1.1, “Cadenas de caracteres”.

The string in `from` is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in `to` and a terminating null byte is appended. Characters encoded are `NUL` (ASCII 0), '`\n`', '`\r`', '`\`', '`'`', '`"`', and Control-Z (see Sección 9.1, “Valores literales”). (Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. This function quotes the other characters to make them easier to read in log files.)

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 you need 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 character.

If you need to change the character set of the connection, you should 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

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"What's this",11);
*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 `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

24.2.3.49. `mysql_real_query()`

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)
```

Description

Executes the SQL query pointed to by `query`, which should be a string `length` bytes long. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (';') or `\g` to

the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Sección 24.2.9, “Tratamiento por parte de la API C de la ejecución de múltiples consultas”.

You **must** use `mysql_real_query()` rather than `mysql_query()` for queries that contain binary data, because binary data may contain the '`\0`' character. In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the query string.

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See Sección 24.2.3.20, “`mysql_field_count()`”.

Return Values

Zero if the query was successful. Non-zero 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.

24.2.3.50. `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.

Return Values

Zero for success. Non-zero 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.

24.2.3.51. `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 that should be a value returned from `mysql_row_tell()` or from `mysql_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_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()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

24.2.3.52. `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()`.

You should 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.

24.2.3.53. `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 do not include an 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. Non-zero 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.

24.2.3.54. `mysql_set_character_set()`

```
int mysql_set_character_set(MYSQL *mysql, 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. Non-zero 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_charset_name(&mysql, "utf8"))
{
    printf("New client character set: %s\n", mysql_character_set_name(&mysql));
}
```

24.2.3.55. `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:

MYSQL_OPTION_MULTI_STATEMENTS_ON	Enable multiple-statement support.
MYSQL_OPTION_MULTI_STATEMENTS_OFF	Disable multiple-statement support.

This function was added in MySQL 4.1.1.

Return Values

Zero for success. Non-zero 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 didn't support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server didn't support the option one tried to set.

24.2.3.56. `mysql_shutdown()`

```
int mysql_shutdown(MYSQL *mysql, enum enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have `SHUTDOWN` privileges. The `shutdown_level` argument was added in MySQL 4.1.3 (and 5.0.1). The MySQL server currently supports only one type (level of gracefulness) of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Later we wil add more levels and then the `shutdown_level` argument will enable us to choose the desired level. MySQL servers and MySQL clients before and after 4.1.3 are compatible; MySQL servers newer than 4.1.3 accept the `mysql_shutdown(MYSQL *mysql)` call, and MySQL servers older than 4.1.3 accept the new `mysql_shutdown()` call. But dynamically linked executables which have been compiled with older versions of `libmysqlclient` headers, and call `mysql_shutdown()`, need to be used with the old `libmysqlclient` dynamic library.

The shutdown process is described in Sección 5.4, “El proceso de cierre del servidor MySQL”.

Return Values

Zero for success. Non-zero 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.

24.2.3.57. `mysql_sqlstate()`

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the last error. 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 Capítulo 26, *Manejo de errores en MySQL*.

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `'HY000'` (general error) is used for unmapped errors.

This function was added to MySQL 4.1.1.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See Sección 24.2.3.12, “`mysql_errno()`”. See Sección 24.2.3.13, “`mysql_error()`”. See Sección 24.2.7.26, “`mysql_stmt_sqlstate()`”.

24.2.3.58. `mysql_ssl_set()`

```
int 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 OpenSSL 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 pathname to the key file.

- `cert` is the pathname to the certificate file.

- `ca` is the pathname to the certificate authority file.
- `capath` is the pathname to a directory that contains trusted SSL CA certificates in pem format.
- `cipher` is a list of allowable 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.

24.2.3.59. `mysql_stat()`

```
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`

  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.

24.2.3.60. `mysql_store_result()`

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth).

You don't have to call `mysql_store_result()` or `mysql_use_result()` for other queries, but it does not do any harm or cause any notable performance degredation if you call `mysql_store_result()` in all cases. You can detect if the query didn't have a result set by checking if `mysql_store_result()` returns 0 (more about this later on).

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See Sección 24.2.3.20, “`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 query didn't return a result set (if the query was, for example, 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 if `mysql_error()` returns a non-empty string, if `mysql_errno()` returns non-zero, or if `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.)

Once you have called `mysql_store_result()` and got a result back that isn't a null pointer, you may 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.

You must call `mysql_free_result()` once you are done with the result set.

See Sección 24.2.13.1, “¿Por qué `mysql_store_result()` a veces devuelve `NULL` después de que `mysql_query()` haya dado un resultado?”.

Return Values

A `MYSQL_RES` result structure with the results. `NULL` 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.

24.2.3.61. `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.

Return Values

The thread ID of the current connection.

Errors

None.

24.2.3.62. `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`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 shouldn't use `mysql_use_result()` 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 the `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.

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.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

24.2.3.63. `mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of warnings generated during execution of the previous SQL statement.

This function was added in MySQL 4.1.0.

Return Values

The warning count.

Errors

None.

24.2.3.64. `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

24.2.3.65. `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

24.2.3.66. `mysql_autocommit()`

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

24.2.3.67. `mysql_more_results()`

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

Returns true if more results exist from the currently executed query, and the application must call `mysql_next_result()` to fetch the results.

This function was added in MySQL 4.1.0.

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 Sección 24.2.9, “Tratamiento por parte de la API C de la ejecución de múltiples consultas”. See Sección 24.2.3.68, “`mysql_next_result()`”.

Errors

None.

24.2.3.68. mysql_next_result()

```
int mysql_next_result(MYSQL *mysql)
```

Description

If more query results exist, `mysql_next_result()` reads the next query results and returns the status back to application.

You must call `mysql_free_result()` for the preceding query if it returned a result set.

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 query. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If `mysql_next_result()` returns an error, no other statements are executed and there are no more results to fetch.

See Sección 24.2.9, “Tratamiento por parte de la API C de la ejecución de múltiples consultas”.

This function was added in MySQL 4.1.0.

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 didn't 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.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

24.2.4. Sentencias preparadas de la API C

As of MySQL 4.1, the 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.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

The following statements can be used as prepared statements: `CREATE TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE`, and most `SHOW` statements. Other statements are not yet supported.

24.2.5. Tipos de datos de sentencias preparadas de la API C

Note: Some incompatible changes were made in MySQL 4.1.2. See Sección 24.2.7, “Descripciones de funciones de sentencias preparadas de la API C” for details.

Prepared statements mainly use the `MYSQL_STMT` and `MYSQL_BIND` data structures. A third structure, `MYSQL_TIME`, is used to transfer temporal data.

- `MYSQL_STMT`

  This structure represents a prepared statement. A statement is created by calling `mysql_stmt_init()`, which returns a statement handle, that is, a pointer to a `MYSQL_STMT`. The handle is used for all subsequent statement-related functions until you close it with `mysql_stmt_close()`.

  The `MYSQL_STMT` structure has no members that are for application use. Also, you should not try to make a copy of 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, it is used with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`. For output, it is used with `mysql_stmt_bind_result()` to bind result set buffers for use in fetching rows with `mysql_stmt_fetch()`.

  The `MYSQL_BIND` structure contains the following members for use by application programs. Each is used both for input and for output, although sometimes for different purposes depending on the direction of data transfer.

  - `enum enum_field_types buffer_type`

    The type of the buffer. The allowable `buffer_type` values are listed later in this section. For input, `buffer_type` indicates what type of value you are binding to a statement parameter. For output, it indicates what type of value you expect to receive in a result buffer.

  - `void *buffer`

    For input, this is a pointer to the buffer in which a statement parameter's data value is stored. For output, it is a pointer to the buffer in which to return a result set column value. For numeric column types, `buffer` should point to a variable of the proper C type. (If you are associating the variable with a column that has the `UNSIGNED` attribute, the variable should be an `unsigned` C type. Indicate whether the variable is signed or unsigned by using the `is_unsigned` member, described later in this

list.) For date and time column types, `buffer` should point to a `MYSQL_TIME` structure. For character and binary string column types, `buffer` should point to a character buffer.

- `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()`, or the maximum number of 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, `length` points to an `unsigned long` variable that indicates the length of the parameter value stored in `*buffer`; this is used by `mysql_stmt_execute()`. For output value binding, `mysql_stmt_fetch()` places the length of the column value that is returned into the variable that `length` points to.

  `length` is ignored for numeric and temporal data types because the length of the data value is determined by the `buffer_type` value.

- `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. For output, this value is set to true after you fetch a row if the result set column value returned from the statement is `NULL`.

- `my_bool is_unsigned`

  This member is used for integer types. (These correspond to the `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, and `MYSQL_TYPE_LONGLONG` type codes.) `is_unsigned` should be set to true for unsigned types and false for signed types.

- `my_bool error`

  For output, this member is used output to report data truncation errors. Truncation reporting must be enabled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option. When 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. The `error` member was added in MySQL 5.0.3.

To use a `MYSQL_BIND` structure, you should zero its contents to initialize it, and then set the members just described 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));
```

- `MYSQL_TIME`

  This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. This is done by setting the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types, and setting the `buffer` member to point to a `MYSQL_TIME` structure.

The `MYSQL_TIME` structure contains the following members:

- `unsigned int year`

  The year.

- `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 to indicate whether the time is negative.

- `unsigned long second_part`

  The fractional part of the second. This member currently is 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 Sección 24.2.10, “Manejo de valores de fecha y hora por parte de la API C”.

The following table shows the allowable values that may be specified in the `buffer_type` member of `MYSQL_BIND` structures. The table also shows those SQL types that correspond most closely to each `buffer_type` value, and, for numeric and temporal types, the corresponding C type.

`buffer_type` **Value**	**SQL Type**	**C Type**
`MYSQL_TYPE_TINY`	`TINYINT`	`char`
`MYSQL_TYPE_SHORT`	`SMALLINT`	`short int`
`MYSQL_TYPE_LONG`	`INT`	`int`
`MYSQL_TYPE_LONGLONG`	`BIGINT`	`long long int`

MYSQL_TYPE_FLOAT	FLOAT	float
MYSQL_TYPE_DOUBLE	DOUBLE	double
MYSQL_TYPE_TIME	TIME	MYSQL_TIME
MYSQL_TYPE_DATE	DATE	MYSQL_TIME
MYSQL_TYPE_DATETIME	DATETIME	MYSQL_TIME
MYSQL_TYPE_TIMESTAMP	TIMESTAMP	MYSQL_TIME
MYSQL_TYPE_STRING	CHAR	
MYSQL_TYPE_VAR_STRING	VARCHAR	
MYSQL_TYPE_TINY_BLOB	TINYBLOB/TINYTEXT	
MYSQL_TYPE_BLOB	BLOB/TEXT	
MYSQL_TYPE_MEDIUM_BLOB	MEDIUMBLOB/MEDIUMTEXT	
MYSQL_TYPE_LONG_BLOB	LONGBLOB/LONGTEXT	

Implicit type conversion may be performed in both directions.

24.2.6. Panorámica de las funciones de sentencias preparadas de la API C

Note: Some incompatible changes were made in MySQL 4.1.2. See Sección 24.2.7, “Descripciones de funciones de sentencias preparadas de la API C” for details.

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See Sección 24.2.7, “Descripciones de funciones de sentencias preparadas de la API C”.

Function	Description
mysql_stmt_affected_rows()	Returns the number of rows changes, deleted, or inserted by prepared UPDATE, DELETE, or INSERT statement.
mysql_stmt_attr_get()	Get 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 the result set.
mysql_stmt_close()	Frees memory used by 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 the prepared statement.
mysql_stmt_fetch()	Fetches the next row of data from the result set and returns data for all bound columns.
mysql_stmt_fetch_column()	Fetch data for one column of the current row of the 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 the statement handle.
mysql_stmt_init()	Allocates memory for MYSQL_STMT structure and initializes it.

mysql_stmt_insert_id()	Returns the ID generated for an `AUTO_INCREMENT` column by prepared statement.
mysql_stmt_num_rows()	Returns total rows from the statement buffered result set.
mysql_stmt_param_count()	Returns the number of parameters in a prepared SQL statement.
mysql_stmt_param_metadata()	Return parameter metadata in the form of a result set.
mysql_stmt_prepare()	Prepares an SQL string for execution.
mysql_stmt_reset()	Reset 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 the complete result set to the client.

Call `mysql_stmt_init()` to create a statement handle, then `mysql_stmt_prepare` to prepare it, `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()`.

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.

You can also send the text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See Sección 24.2.7.25, “`mysql_stmt_send_long_data()`”.

When statement execution has been completed, the statement handle must be closed 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 `msyql_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 produces 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:

- 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 total 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 packet row by row 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 code, error message, and SQLSTATE value using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

24.2.7. Descripciones de funciones de sentencias preparadas de la API C

To prepare and execute queries, use the functions in the following sections.

In MySQL 4.1.2, the names of several prepared statement functions were changed:

Old Name	New Name
`mysql_bind_param()`	`mysql_stmt_bind_param()`
`mysql_bind_result()`	`mysql_stmt_bind_result()`
`mysql_prepare()`	`mysql_stmt_prepare()`
`mysql_execute()`	`mysql_stmt_execute()`
`mysql_fetch()`	`mysql_stmt_fetch()`
`mysql_fetch_column()`	`mysql_stmt_fetch_column()`
`mysql_param_count()`	`mysql_stmt_param_count()`
`mysql_param_result()`	`mysql_stmt_param_metadata()`
`mysql_get_metadata()`	`mysql_stmt_result_metadata()`
`mysql_send_long_data()`	`mysql_stmt_send_long_data()`

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

Also in 4.1.2, the signature of the `mysql_stmt_prepare()` function was changed to `int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)`. To create a `MYSQL_STMT` handle, you should use the `mysql_stmt_init()` function.

24.2.7.1. `mysql_stmt_affected_rows()`

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

Returns the total number of rows changed, deleted, or inserted by the last executed statement. May be called immediately after `mysql_stmt_execute()` for `UPDATE`, `DELETE`, or `INSERT` statements. For `SELECT` statements, `mysql_stmt_affected_rows()` works like `mysql_num_rows()`.

This function was added in MySQL 4.1.0.

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_stmt_affected_rows()` was called prior to calling `mysql_stmt_store_result()`. Because `mysql_stmt_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).

See Sección 24.2.3.1, “`mysql_affected_rows()`” for additional information on the return value.

Errors

None.

Example

For the usage of `mysql_stmt_affected_rows()`, refer to the Example from Sección 24.2.7.10, “`mysql_stmt_execute()`”.

24.2.7.2. `mysql_stmt_attr_get()`

```
int 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, then `arg` should point to the value of the integer.

See `mysql_stmt_attr_set()` for a list of options and option types. See Sección 24.2.7.3, “`mysql_stmt_attr_set()`”.

This function was added in MySQL 4.1.2.

Return Values

`0` if okay. Non-zero if `option` is unknown.

Errors

None.

24.2.7.3. mysql_stmt_attr_set()

```
int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
const void *arg)
```

Description

Can be used to set affect behavior for a 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. If the option is an integer, then `arg` should point to the value of the integer.

Possible `option` values:

Option	Argument Type	Function
`STMT_ATTR_UPDATE_MAX_LENGTH`	`my_bool *`	If set to 1: Update metadata `MYSQL_FIELD->max_length` in `mysql_stmt_store_result()`.
`STMT_ATTR_CURSOR_TYPE`	`unsigned long *`	Type of cursor to open for statement when `mysql_stmt_execute()` 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.

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.

This function was added in MySQL 4.1.2. 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

`0` if okay. Non-zero if `option` is unknown.

Errors

None.

24.2.7.4. mysql_stmt_bind_param()

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_param()` is used to bind 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 an 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];
```

The members of each `MYSQL_BIND` element that should be set are described in Sección 24.2.5, “Tipos de datos de sentencias preparadas de la API C”.

This function was added in MySQL 4.1.2.

Return Values

Zero if the bind was successful. Non-zero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE`

  Indicates if the bind is to supply the long data in chunks and if the buffer type is non string or binary.

- `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

For the usage of `mysql_stmt_bind_param()`, refer to the Example from Sección 24.2.7.10, “`mysql_stmt_execute()`”.

24.2.7.5. `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 (bind) 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 an 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 doesn't return data values in chunks.

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 does not place data into the newly bound buffers until 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 the address of the length buffer. The members of each `MYSQL_BIND` element that should be set are described in Sección 24.2.5, “Tipos de datos de sentencias preparadas de la API C”.

This function was added in MySQL 4.1.2.

Return Values

Zero if the bind was successful. Non-zero 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

For the usage of `mysql_stmt_bind_result()`, refer to the Example from Sección 24.2.7.13, “mysql_stmt_fetch()”.

24.2.7.6. `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.

This function was added in MySQL 4.1.0.

Return Values

Zero if the statement was freed successfully. Non-zero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

For the usage of `mysql_stmt_close()`, refer to the Example from Sección 24.2.7.10, “`mysql_stmt_execute()`”.

24.2.7.7. `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()`.

This function was added in MySQL 4.1.1.

Return Values

None.

Errors

None.

24.2.7.8. `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`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`. The server error codes also are listed at Capítulo 26, *Manejo de errores en MySQL*.

This function was added in MySQL 4.1.0.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

24.2.7.9. `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. This means the following two tests are equivalent:

```
if (mysql_stmt_errno(stmt))
{
  // an error occurred
}

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. Currently you can choose error messages in several different languages.

This function was added in MySQL 4.1.0.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

24.2.7.10. `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.

If the statement is an `UPDATE`, `DELETE`, or `INSERT`, the total number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`. If this is 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 Sección 24.2.7.13, “`mysql_stmt_fetch()`”.

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.

This function was added in MySQL 4.1.2. Cursor support was added in MySQL 5.

Return Values

Zero if execution was successful. Non-zero 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.

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.

```
#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));
  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 total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
                (unsigned long) affected_rows);

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 BitKeeper source repository.

24.2.7.11. `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.

This function was added in MySQL 4.1.1. Cursor support was added in MySQL 5.

Return Values

Zero if the result set was freed successfully. Non-zero if an error occurred.

Errors

24.2.7.12. 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 prepared INSERT statement into a table which contains an AUTO_INCREMENT field.

See Sección 24.2.3.34, “mysql_insert_id()” for more information.

This function was added in MySQL 4.1.2.

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.

24.2.7.13. 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() that creates a result set or after mysql_stmt_store_result(), which is called after mysql_stmt_execute() to buffer the entire 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 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

This function was added in MySQL 4.1.2.

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 not returned unless truncation reporting is enabled with mysql_options(). To determine which parameters were truncated when this value is returned, check the error members of the MYSQL_BIND parameter structures.

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 Sección 24.2.7.10, “`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;
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];

/* 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 */

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];

/* 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];

/* 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];

/* 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 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 */
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
    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);
}
```

24.2.7.14. `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.

This function was added in MySQL 4.1.2.

Return Values

Zero if the value was fetched successfully. Non-zero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO`

  Invalid column number.

- `CR_NO_DATA`

  The end of the result set has already been reached.

24.2.7.15. `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()`.

This function was added in MySQL 4.1.3.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

24.2.7.16. `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 *)`.

This function was added in MySQL 4.1.2.

Return values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY`

  Out of memory.

24.2.7.17. `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 or not 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.

This function was added in MySQL 4.1.1.

Return Values

The number of rows in the result set.

Errors

None.

24.2.7.18. `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 function was added in MySQL 4.1.2.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

For the usage of `mysql_stmt_param_count()`, refer to the Example from Sección 24.2.7.10, “mysql_stmt_execute()”.

24.2.7.19. `mysql_stmt_param_metadata()`

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

To be added.

This function was added in MySQL 4.1.2.

Description

Return Values

Errors

24.2.7.20. `mysql_stmt_prepare()`

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long
length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `query` 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 allowed 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 allowed 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.

This function was added in MySQL 4.1.2.

Return Values

Zero if the statement was prepared successfully. Non-zero 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 prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns non-zero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

For the usage of `mysql_stmt_prepare()`, refer to the Example from Sección 24.2.7.10, “mysql_stmt_execute()”.

24.2.7.21. `mysql_stmt_reset()`

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Reset the prepared statement on the client and server to state after prepare. This is mainly used to reset data sent with `mysql_stmt_send_long_data()`. Any open cursor for the statement is closed.

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

This function was added in MySQL 4.1.1. Cursor support was added in MySQL 5.

Return Values

Zero if the statement was reset successfully. Non-zero 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.

24.2.7.22. `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 total 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()`.

This function was added in MySQL 4.1.2.

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

For the usage of `mysql_stmt_result_metadata()`, refer to the Example from Sección 24.2.7.13, “mysql_stmt_fetch()”.

24.2.7.23. `mysql_stmt_row_seek()`

```
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()`.

This function was added in MySQL 4.1.1.

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.

24.2.7.24. `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()`.

This function was added in MySQL 4.1.1.

Return Values

The current offset of the row cursor.

Errors

None.

24.2.7.25. 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

Allows an application to send parameter data to the server in pieces (or “chunks”). This function 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.

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 Sección 24.2.7.21, “mysql_stmt_reset()”.

This function was added in MySQL 4.1.2.

Return Values

Zero if the data is sent successfully to server. Non-zero 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_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;

smtt = 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");
  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);
}
```

24.2.7.26. `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 Capítulo 26, *Manejo de errores en MySQL*.

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `"HY000"` (general error) is used for unmapped errors.

This function was added to MySQL 4.1.1.

Return Values

A null-terminated character string containing the SQLSTATE error code.

24.2.7.27. mysql_stmt_store_result()

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

You must call `mysql_stmt_store_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`), and only if you want to buffer the complete result set by the client, so that the subsequent `mysql_stmt_fetch()` call returns buffered data.

It is unnecessary to call `mysql_stmt_store_result()` for other statements, 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 Sección 24.2.7.22, “mysql_stmt_result_metadata()”.

Note: MySQL doesn't 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 doesn't 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 Sección 24.2.7.3, “mysql_stmt_attr_set()”.

This function was added in MySQL 4.1.0.

Return Values

Zero if the results are buffered successfully. Non-zero 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.

24.2.8. Problemas con sentencias preparadas de la API C

Here follows a list of the currently known problems with prepared statements:

- `TIME`, `TIMESTAMP`, and `DATETIME` don't support sub 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 doesn't print the leading zeros. (For example, with `MIN(number-with-zerofill)`).
- When converting a floating point number to a string in the client, the value may be slightly different in the last digits.
- Prepared statements do not use the Query Cache, even in cases where a query does not contain any placeholders. See Sección 5.12.1, “Cómo opera la caché de consultas”.

24.2.9. Tratamiento por parte de la API C de la ejecución de múltiples consultas

From version 4.1, MySQL supports the execution of multiple statements specified in a single query string. To use this capability with a given connection, you must specify the `CLIENT_MULTI_STATEMENTS` option in the flags parameter of `mysql_real_connect()` when opening the connection. You can also set this for an existing connection by calling `mysql_set_server_option(MYSQL_OPTION_MULTI_STATEMENTS_ON)`

By default, `mysql_query()` and `mysql_real_query()` return only the first query status and the subsequent queries status can be processed using `mysql_more_results()` and `mysql_next_result()`.

```
/* Connect to server with option CLIENT_MULTI_STATEMENTS */
mysql_real_connect(..., CLIENT_MULTI_STATEMENTS);

/* Now execute multiple queries */
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");
do
{
  /* Process all results */
  ...
  printf("total affected rows: %lld", mysql_affected_rows(mysql));
  ...
  if (!(result= mysql_store_result(mysql)))
  {
     printf(stderr, "Got fatal error processing query\n");
     exit(1);
  }
  process_result_set(result); /* client function */
  mysql_free_result(result);
} while (!mysql_next_result(mysql));
```

The multiple-statement capability can be used with `mysql_query()` or `mysql_real_query()`. It cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement.

24.2.10. Manejo de valores de fecha y hora por parte de la API C

The new binary protocol available in MySQL 4.1 and above allows 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 Sección 24.2.5, “Tipos de datos de sentencias preparadas de la API C”.

To send temporal data values, you create a prepared statement with `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 you're passing.

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_bind_results()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

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);
..
```

24.2.11. Descripción de funciones de la API C para el control de subprocesos

You need to use the following functions when you want to create a threaded client. See Sección 24.2.15, “Cómo hacer un cliente multihilo”.

24.2.11.1. `my_init()`

```
void my_init(void)
```

Description

This function needs to be called once in the program before calling any MySQL function. This initializes some global variables that MySQL needs. If you are using a thread-safe client library, this also calls `mysql_thread_init()` for this thread.

This is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()` and `mysql_connect()`.

Return Values

None.

24.2.11.2. `mysql_thread_init()`

```
my_bool mysql_thread_init(void)
```

Description

This function needs to be called for each created thread to initialize thread-specific variables.

This is automatically called by `my_init()` and `mysql_connect()`.

Return Values

Zero if successful. Non-zero if an error occurred.

24.2.11.3. `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()`.

Note that this function **is not invoked automatically** by the client library. It must be called explicitly to avoid a memory leak.

Return Values

None.

24.2.11.4. mysql_thread_safe()

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client is compiled as thread-safe.

Return Values

1 is the client is thread-safe, 0 otherwise.

24.2.12. Descripción de las funciones de la API C del servidor incrustado (embedded)

If you want to allow your application to be linked against the embedded MySQL server library, you must use the `mysql_server_init()` and `mysql_server_end()` functions. See Sección 24.2.16, “libmysqld, la biblioteca del servidor MySQL incrustado (embedded)”.

However, to provide improved memory management, even programs that are linked with `-lmysqlclient` rather than `-lmysqld` should include calls to begin and end use of the library. As of MySQL 4.1.10 and 5.0.3, the `mysql_library_init()` and `mysql_library_end()` functions can be used to do this. These 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`. For more information, see Sección 24.2.2, “Panorámica de funciones de la API C”.

24.2.12.1. mysql_server_init()

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function **must** be called once in the program using the embedded server before calling any other MySQL function. It starts the server and initializes any subsystems (`mysys`, `InnoDB`, etc.) that the server uses. If this function is not called, the next call to `mysql_init()` executes `mysql_server_init()`. If you are using the DBUG package that comes with MySQL, you should call this after you have called `my_init()`.

The `argc` and `argv` arguments are analogous to the arguments to `main()`. The first element of `argv` is ignored (it typically contains the program name). For convenience, `argc` may be `0` (zero) if there are no command-line arguments for the server. `mysql_server_init()` makes a copy of the arguments so it's safe to destroy `argv` or `groups` after the call.

If you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `NULL`-terminated list of strings in `groups` selects which groups in the option files are active. See Sección 4.3.2, “Usar ficheros de opciones”. For convenience, `groups` may be `NULL`, in which case the `[server]` and `[embedded]` groups are active.

Example

```
#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_server_init(sizeof(server_args) / sizeof(char *),
                        server_args, server_groups))
    exit(1);

  /* Use any MySQL API functions here */

  mysql_server_end();

  return EXIT_SUCCESS;
}
```

Return Values

0 if okay, 1 if an error occurred.

24.2.12.2. `mysql_server_end()`

```
void mysql_server_end(void)
```

Description

This function **must** be called once in the program after all other MySQL functions. It shuts down the embedded server.

Return Values

None.

24.2.13. Preguntas y problemas comunes en el uso de la API C

24.2.13.1. ¿Por qué `mysql_store_result()` a veces devuelve `NULL` después de que `mysql_query()` haya dado un resultado?

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 couldn't 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 non-empty 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 non-zero value, the statement should have produced a non-empty 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()`.

24.2.13.2. Qué resultados se puede obtener de una consulta

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`.

  In MySQL 3.23, there is an exception when `DELETE` is used without a `WHERE` clause. In this case, the table is re-created as an empty table and `mysql_affected_rows()` returns zero for the number of records affected. In MySQL 4.0, `DELETE` always returns the correct number of rows deleted. For a fast recreate, 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 Sección 24.2.3.34, “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.

24.2.13.3. Cómo obtener el ID único del último registro insertado

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.

You can check from your C applications whether a value was stored into 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);
}
```

For more information, see Sección 24.2.3.34, “mysql_insert_id()”.

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 non-magic value (that is, a value that is not `NULL` and not `0`).

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
```

Note that `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()`.

24.2.13.4. Problemas enlazando con la API C

When linking with the C API, the following errors may occur on some systems:

```
gcc -g -o client test.o -L/usr/local/lib/mysql -lmysqlclient -lsocket -lnsl

Undefined        first referenced
 symbol          in file
floor            /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

If this happens on your system, you must include the math library by adding `-lm` to the end of the compile/link line.

24.2.14. Generar programas cliente

If you compile MySQL clients that you've written yourself or that you obtain from a third-party, they must be linked using the `-lmysqlclient -lz` option on 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 `-L/usr/local/mysql/lib -lmysqlclient -lz` on the link command.

For clients that use MySQL header files, you may need to specify a `-I` option when you compile them (for example, `-I/usr/local/mysql/include`), so the compiler can find the header files.

To make it simpler to compile MySQL programs on Unix, we have provided the `mysql_config` script for you. See Sección 24.1.2, “mysql_config — ”.

You can use it to compile a MySQL client as follows:

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

The `sh -c` is needed to get the shell not to treat the output from `mysql_config` as one word.

24.2.15. Cómo hacer un cliente multihilo

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` 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, the socket handling should be thread-safe.

New in 4.0.16: To not abort the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_server_init`(), `mysql_init()` or `mysql_connect()`. If you want to have your own `SIGPIPE` handler, you should first call `mysql_server_init()` and then install your handler. In older versions of MySQL `SIGPIPE` was blocked, but only in the thread safe client library, for every call to `mysql_init()`.

In the older binaries we distribute on our Web site (http://www.mysql.com/), the client libraries are not normally compiled with the thread-safe option (the Windows binaries are by default compiled to be thread-safe). Newer binary distributions should have both a normal and a thread-safe client library.

To get a threaded client where you can interrupt the client from other threads and set timeouts when talking with the MySQL server, you should use the `-lmysys`, `-lmystrings`, and `-ldbug` libraries and the `net_serv.o` code that the server uses.

If you don't need interrupts or timeouts, you can just compile a thread-safe client library (`mysqlclient_r`) and use this. See Sección 24.2, “La API C de MySQL”. In this case, you don't have to worry about the `net_serv.o` object file or the other MySQL libraries.

When using a threaded client and you want to use timeouts and interrupts, you can make great use of the routines in the `thr_alarm.c` file. If you are using routines from the `mysys` library, the only thing you must remember is to call `my_init()` first! See Sección 24.2.11, “Descripción de funciones de la API C para el control de subprocesos”.

All functions except `mysql_real_connect()` are by default thread-safe. The following notes describe how to compile a thread-safe client library and use it in a thread-safe manner. (The notes below for `mysql_real_connect()` actually apply to `mysql_connect()` as well, but because `mysql_connect()` is deprecated, you should be using `mysql_real_connect()` anyway.)

To make `mysql_real_connect()` thread-safe, you must recompile the client library with this command:

```
shell> ./configure --enable-thread-safe-client
```

This creates a thread-safe client library `libmysqlclient_r`. (Assuming that your OS has a thread-safe `gethostbyname_r()` function.) This library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Two threads can't send a query to the MySQL server at the same time on the same connection. In particular, you have to ensure that between a `mysql_query()` and `mysql_store_result()` no other thread is using the same connection.
- Many threads can access different result sets that are retrieved with `mysql_store_result()`.
- If you use `mysql_use_result`, you have to 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()`.
- If you want to use multiple threads on the same connection, you must have a mutex lock around your `mysql_query()` and `mysql_store_result()` call combination. Once `mysql_store_result()` is ready, the lock can be released and other threads may query the same connection.
- If you program with POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

You need to know the following if you have a thread that is calling MySQL functions which did not create the connection to the MySQL database:

When you call `mysql_init()` or `mysql_connect()`, 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()` or `mysql_connect()`, 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.

The get things to work smoothly you have to do the following:

1. Call `my_init()` at the start of your program if it calls any other MySQL function before calling `mysql_real_connect()`.

2. Call `mysql_thread_init()` in the thread handler before calling any MySQL function.
3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

You may get some errors because of undefined symbols when linking your client with `libmysqlclient_r`. In most cases this is because you haven't included the thread libraries on the link/compile line.

24.2.16. libmysqld, la biblioteca del servidor MySQL incrustado (embedded)

24.2.16.1. Panorámica de la librería del servidor MySQL incrustado (embedded)

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:

Function	When to Call
`mysql_server_init()`	Should be called before any other MySQL function is called, preferably early in the `main()` function.
`mysql_server_end()`	Should be called before your program exits.
`mysql_thread_init()`	Should be called in each thread you create that accesses MySQL.
`mysql_thread_end()`	Should be called before calling `pthread_exit()`

Then you must link your code with `libmysqld.a` instead of `libmysqlclient.a`.

The `mysql_server_xxx()` functions are also included in `libmysqlclient.a` to allow you to change between the embedded and the client/server version by just linking your application with the right library. See Sección 24.2.12.1, “`mysql_server_init()`”.

24.2.16.2. Compilar programas con `libmysqld`

To get a `libmysqld` library you should configure MySQL with the `--with-embedded-server` option. See Sección 2.8.2, “Opciones típicas de `configure`”.

When you link your program with `libmysqld`, you must also include the system-specific `pthread` libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing `mysql_config --libmysqld-libs`.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly call any thread functions in your code.

To compile a C program to include the necessary files to embed the MySQL server library into a compiled version of a program, use the GNU C compiler (`gcc`). The compiler will need to know where to find various files and need instructions on how to compile the program. Below is an example of how a program could be compiled from the command-line:

```
gcc mysql_test.c -o mysql_test -lz \
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Immediately following the `gcc` command is the name of the uncompiled C program file. After it, the `-o` option is given to indicate that the file name that follows is the name that the compiler is to give to the output file, the compiled program. The next line of code tells the compiler to obtain the location of the include files and libraries and other settings for the system on which it's compiled. Because of a problem with `mysql_config`, the option `-lz` (for compression) is added here. The `mysql_config` piece is contained in backticks, not single quotes.

24.2.16.3. Restricciones cuando se utiliza el servidor MySQL incrustado (embedded)

The embedded server has the following limitations:

- No support for `ISAM` tables. (This is mainly done to make the library smaller)
- No user-defined functions (UDFs).
- No stack trace on core dump.
- No internal RAID support. (This is not normally needed as most current operating systems support big files).
- You cannot set this up as a master or a slave (no replication).
- You cannot connect to an embedded server from an outside process with sockets or TCP/IP. However, you can connect to an intermediate application, which in turn can connect to an embedded server on the behalf of a remote client or outside process.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

24.2.16.4. Opciones con el servidor incrustado (embedded)

Any options that may be given with the `mysqld` server daemon, may be used with an embedded server library. Server options may be given in an array as an argument to the `mysql_server_init()`, which initializes the server. They also may be given in an option file like `my.cnf`. To specify an option file for a C program, use the `--defaults-file` option as one of the elements of the second argument of the `mysql_server_init()` function. See Sección 24.2.12.1, “mysql_server_init()” for more information on the `mysql_server_init()` function.

Using option files can make it easier to switch between a client/server application and one where MySQL is embedded. Put common options under the `[server]` group. These are read by both MySQL versions. Client/server-specific options should go under the `[mysqld]` section. Put options specific to the embedded MySQL server library in the `[embedded]` section. Options specific to applications go under section labeled `[ApplicationName_SERVER]`. See Sección 4.3.2, “Usar ficheros de opciones”.

24.2.16.5. Cosas por hacer (TODO) en el servidor incrustado (embedded)

- We are going to provide options to leave out some parts of MySQL to make the library smaller.
- There is still a lot of speed optimization to do.
- Errors are written to `stderr`. We will add an option to specify a filename for these.
- We have to change InnoDB not to be so verbose when using the embedded version. If your database does not contain InnoDB tables, to suppress related messages you can add the `--skip-innodb` option to the options file under the group `[libmysqd_server]`, or when initializing the server with `mysql_server_init()`.

24.2.16.6. Ejemplos de servidor incrustado (embedded)

These two example programs should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed, mostly with file paths. These examples are designed to give enough details for you to understand the problem, without the clutter that is a necessary part of a real application. The first example is very straightforward. The second example is a little more advanced with some error checking. The first is followed by a command-line entry for compiling the program. The second is followed by a GNUmake file that may be used for compiling instead.

Example 1

`test1_libmysqld.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = { "mysql_test", "--defaults-file=my.cnf" };
int num_elements = sizeof(server_options)/ sizeof(char *);

static char *server_groups[] = { "libmysqld_server", "libmysqld_client" };

int main(void)
{
   mysql_server_init(num_elements, server_options, server_groups);
   mysql = mysql_init(NULL);
   mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
   mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

   mysql_real_connect(mysql, NULL,NULL,NULL, "database1", 0,NULL,0);

   mysql_query(mysql, "SELECT column1, column2 FROM table1");

   results = mysql_store_result(mysql);

   while((record = mysql_fetch_row(results))) {
      printf("%s - %s \n", record[0], record[1]);
   }

   mysql_free_result(results);
   mysql_close(mysql);
   mysql_server_end();

   return 0;
}
```

Here is the command line for compiling the above program:

```
gcc test1_libmysqld.c -o test1_libmysqld -lz \
 `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Example 2

To try out the example, create an `test2_libmysqld` directory at the same level as the mysql-4.0 source directory. Save the `test2_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test2_libmysqld` directory.

test2_libmysqld.c

```
/*
 * A simple example client, using the embedded MySQL server library
*/

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
  "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
  MYSQL *one, *two;

  /* mysql_server_init() must be called before any other mysql
   * functions.
   *
   * You can use mysql_server_init(0, NULL, NULL), and it
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test2_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
   * If you link this client against the normal mysqlclient
   * library, this function is just a stub that does nothing.
   */
  mysql_server_init(argc, argv, (char **)server_groups);

  one = db_connect("test");
  two = db_connect(NULL);

  db_do_query(one, "SHOW TABLE STATUS");
  db_do_query(two, "SHOW DATABASES");

  mysql_close(two);
  mysql_close(one);

  /* This must be called after all other mysql functions */
  mysql_server_end();

  exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
```

```
  va_list ap;
  va_start(ap, fmt);
  vfprintf(stderr, fmt, ap);
  va_end(ap);
  (void)putc('\n', stderr);
  if (db)
    db_disconnect(db);
  exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
  MYSQL *db = mysql_init(NULL);
  if (!db)
    die(db, "mysql_init failed: no memory");
  /*
   * Notice that the client and server use separate group names.
   * This is critical, because the server does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
  if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
    die(db, "mysql_real_connect failed: %s", mysql_error(db));

  return db;
}

void
db_disconnect(MYSQL *db)
{
  mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
  if (mysql_query(db, query) != 0)
    goto err;

  if (mysql_field_count(db) > 0)
  {
    MYSQL_RES   *res;
    MYSQL_ROW    row, end_row;
    int num_fields;

    if (!(res = mysql_store_result(db)))
      goto err;
    num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)))
    {
      (void)fputs(">> ", stdout);
      for (end_row = row + num_fields; row < end_row; ++row)
        (void)printf("%s\t", row ? (char*)*row : "NULL");
      (void)fputc('\n', stdout);
    }
    (void)fputc('\n', stdout);
    mysql_free_result(res);
  }
  else
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  return;

err:
  die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

GNUmakefile

```
# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-4.0/include
#lib      := $(HOME)/mysql-4.0/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lz -lm -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
        rm -f $(targets) $(objects) *.core
```

24.2.16.7. Licenciamiento del servidor incrustado (embedded)

We encourage everyone to promote free software by releasing code under the GPL or a compatible license. For those who are not able to do this, another option is to purchase a commercial license for the MySQL code from MySQL AB. For details, please see http://www.mysql.com/company/legal/licensing/.

24.3. API PHP de MySQL

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with the Apache Web server.

PHP actually provides two different MySQL API extensions:

- **mysql** - Available for PHP versions 4 and 5, this extension is intended for use with MySQL versions prior to MySQL 4.1. This extension does not support the improved authentication protocol used in MySQL 4.1.1 and later, nor does it support prepared statements or multiple statements. If you wish to use this extension with MySQL 4.1 and more recent versions of MySQL, you will likely want to configure the MySQL server to use the `--old-passwords` option (see Sección A.2.3, “`Client does not support authentication protocol`”). This extension is documented on the PHP Website at http://php.net/mysql.
- **mysqli** - Stands for "MySQL, Improved", this extension is available only in PHP 5. It is intended for use with MySQL 4.1.1 and later. This extension fully supports the authentication protocol used in MySQL

4.1.1 and subsequent MySQL releases as well as the Prepared Statements and Multiple Statements APIs. In addition, this extension provides an advanced, object-oriented programming interface. You can read the documentation for the **mysqli** extension at http://php.net/mysqli. A helpful article can be found at http://www.zend.com/php5/articles/php5-mysqli.php.

The PHP distribution and documentation are available from the PHP Website.

24.3.1. Problemas comunes con MySQL y PHP

- Error: "Maximum Execution Time Exceeded" This is a PHP limit; go into the `php.ini` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- Error: "Fatal error: Call to unsupported or undefined function mysql_connect() in .." This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This is described in detail in the PHP manual.
- Error: "undefined reference to `uncompress'" This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.
- Error: "Client does not support authentication protocol" This is most often encountered when trying to use the older **mysql** extension with MySQL 4.1.1 and later. Possible solutions are: downgrade to MySQL 4.0; switch to PHP 5 and the newer **mysqli** extension; or configure the MySQL server with `--old-passwords`. (See Sección A.2.3, “`Client does not support authentication protocol`”, for more information.)

24.4. La API Perl de MySQL

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 server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

Installation instructions for Perl DBI support are given in Sección 2.13, “Notas sobre la instalación de Perl”.

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 forth 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. MySQL AB hosts a list specifically about `DBD::mysql`; see Sección 1.6.1.1, “Las listas de correo de MySQL”.
- 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/.

24.5. API C++ de MySQL

`MySQL++` is a MySQL API for C++. Warren Young has taken over this project. More information can be found at http://www.mysql.com/products/mysql++/.

24.5.1. Borland C++

You can compile the MySQL Windows source with Borland C++ 5.02. (The Windows source includes only projects for Microsoft VC++, for Borland C++ you have to do the project files yourself.)

One known problem with Borland C++ is that it uses a different structure alignment than VC++. This means that you run into problems if you try to use the default `libmysql.dll` libraries (that were compiled with VC++) with Borland C++. To avoid this problem, only call `mysql_init()` with `NULL` as an argument, not a pre-allocated `MYSQL` structure.

24.6. La API Python de MySQL

`MySQLdb` 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/.

24.7. La API Tcl de MySQL

`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/.

24.8. El visor de MySQL Eiffel

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.

Capítulo 25. Conectores

Tabla de contenidos

Este capítulo describe los Conectores MySQL, controladores (drivers) que proporcionan a los programas cliente conectividad con el servidor MySQL. Existen actualmente cinco conectores MySQL:

- Connector/ODBC proporciona soporte a nivel de controlador para la conexión con un servidor MySQL usando la API de Conectividad de Bases de datos Abierta (ODBC por sus siglas en inglés). Con este controlador la conexión ODBC es posible desde las plataformas Windows, Unix y Mac OS X.

- Connector/NET permite a los desarrolladores crear aplicaciones .NET usando los datos almacenados en una base de datos MySQL. Connector/NET implementa una interfaz ADO.NET totalmente funcional y proporciona soporte para su uso con herramientas compatibles con ADO.NET. Las aplicaciones que se desee usen Connector/NET pueden escribirse en cualquier lenguaje .NET soportado.

- El Plugin Visual Studio MySQL trabaja con Connector/NET y Visual Studio 2005. Este plugin es un proveedor DDEX, lo que significa que se pueden usar herramientas de manipulación de esquemas y datos dentro de Visual Studio para crear y editar objetos dentro de una base de datos MySQL.
- Connector/J proporciona soporte de controlador para conectar con MySQL desde una aplicación Java usando la API de Conectividad con Bases de Datos Java estándar (JDBC).
- Connector/MXJ es una herramienta que permite poner en marcha y administrar fácilmente el servidor y la base de datos MySQL a través de una aplicación Java
- Connector/PHP es un controlador para conectar Windows con PHP. Proporciona las extensiones `mysql` y `mysqli` para su uso con MySQL 5.0.18 y posteriores.

Para información de cómo conectar a un servidos MySQL usando otros lenguajes e interfaces distintos a los detallados en este apartado, incluyendo Perl, Python y PHP para otras plataformas y ambientes, por favor vea el capítulo Capítulo 24, *APIs de MySQL*.

25.1. MySQL Connector/ODBC

The MySQL Connector/ODBC is the name for the family of MySQL ODBC drivers (previously called MyODBC drivers) that provide access to a MySQL database using the industry standard Open Database Connectivity (ODBC) API. This reference covers Connector/ODBC 3.51, a version of the API that provides ODBC 3.5x compliant access to a MySQL database. MySQL Connector/ODBC es el nombre de la familia de controladores MySQL ODBC (anteriormente llamados Controladores MyODBC) que proporcionan acceso a una base de datos MySQL usando el estándar industrial de Conectividad de Base de Datos Abierta (del inglés Open Database Connectivity)

The manual for versions of Connector/ODBC older than 3.51 can be located in the corresponding binary or source distribution.

For more information on the ODBC API standard and how to use it, refer to http://www.microsoft.com/data/.

The application development part of this reference assumes a good working knowledge of C, general DBMS knowledge, and finally, but not least, familiarity with MySQL. For more information about MySQL functionality and its syntax, refer to http://dev.mysql.com/doc/.

Typically, you need to install Connector/ODBC only on Windows machines. For Unix and Mac OS X you can use the native MySQL network or named pipe to communicate with your MySQL database. You may need Connector/ODBC for Unix or Mac OS X if you have an application that requires an ODBC interface to communicate with the database. Applications that require ODBC to communicate with MySQL include ColdFusion, Microsoft Office, and Filemaker Pro.

If you want to install the Connector/ODBC connector on a Unix host, then you must also install an ODBC manager.

If you have questions that are not answered in this document, please send a mail message to `<myodbc@lists.mysql.com>`.

25.1.1. Introduction to Connector/ODBC

ODBC (Open Database Connectivity) provides a way for client programs to access a wide range of databases or data sources. ODBC is a standardized API that allows connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls, error codes, and data types that can be used to develop database-independent applications. ODBC usually is used when database independence or simultaneous access to different data sources is required.

For more information about ODBC, refer to http://www.microsoft.com/data/.

25.1.1.1. Connector/ODBC Versions

There are currently two version of Connector/ODBC available:

- Connector/ODBC 5.0, currently in beta status, has been designed to extend the functionality of the Connector/ODBC 3.51 driver and incorporate full support for the functionality in the MySQL 5.0 server release, including stored procedures and views. Applications using Connector/ODBC 3.51 will be compatible with Connector/ODBC 5.0, while being able to take advantage of the new features. Features and functionality of the Connector/ODBC 5.0 driver are not currently included in this guide.
- Connector/ODBC 3.51 is the current release of the 32-bit ODBC driver, also known as the MySQL ODBC 3.51 driver. This version is enhanced compared to the older Connector/ODBC 2.50 driver. It has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) in order to continue to provide all functionality of ODBC for accessing MySQL.
- MyODBC 2.50 is the previous version of the 32-bit ODBC driver from MySQL AB that is based on ODBC 2.50 specification level 0 (with level 1 and 2 features). Information about the MyODBC 2.50 driver is included in this guide for the purposes of comparison only.

> **Nota**
>
> From this section onward, the primary focus of this guide is the Connector/ODBC 3.51 driver. More information about the MyODBC 2.50 driver in the documentation included in the installation packages for that version. If there is a specific issue (error or known problem) that only affects the 2.50 version, it may be included here for reference.

> **Nota**
>
> Version numbers for MySQL products are formatted as X.X.X. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.XX.XX. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

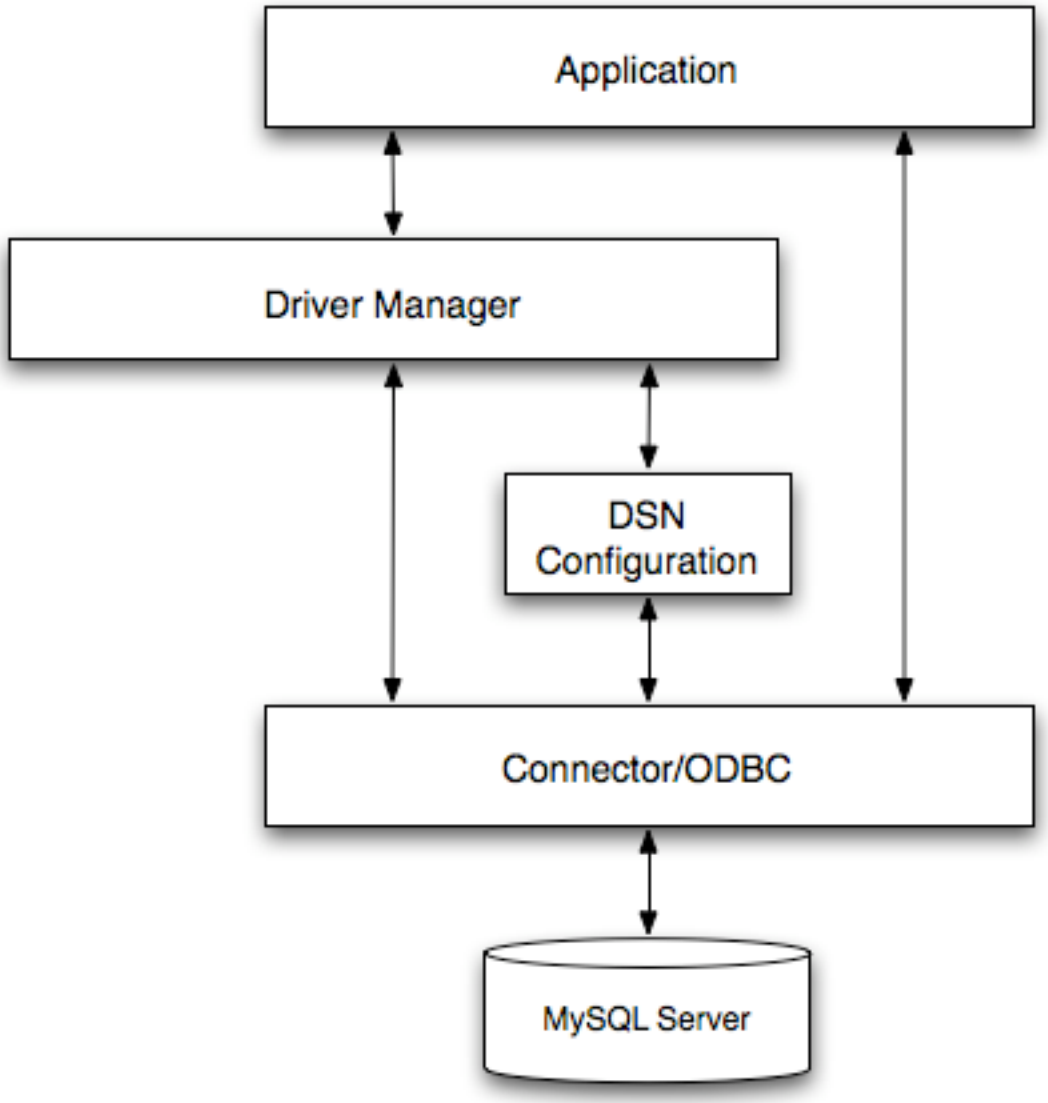
25.1.1.2. General Information About ODBC and Connector/ODBC

Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. It is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A survey of ODBC functions supported by Connector/ODBC is given at Sección 25.1.5.1, “Connector/ODBC API Reference”. For general information about ODBC, see http://www.microsoft.com/data/.

Connector/ODBC Architecture

The Connector/ODBC architecture is based on five components, as shown in the following diagram:



- **Application:**

  The Application uses the ODBC API to access the data from the MySQL server. The ODBC API in turn uses the communicates with the Driver Manager. The Application communicates with the Driver Manager using the standard ODBC calls. The Application does not care where the data is stored, how it is stored, or even how the system is configured to access the data. It needs to know only the Data Source Name (DSN).

  A number of tasks are common to all applications, no matter how they use ODBC. These tasks are:

  - Selecting the MySQL server and connecting to it
  - Submitting SQL statements for execution
  - Retrieving results (if any)
  - Processing errors

- Committing or rolling back the transaction enclosing the SQL statement
- Disconnecting from the MySQL server

Because most data access work is done with SQL, the primary tasks for applications that use ODBC are submitting SQL statements and retrieving any results generated by those statements.

- **Driver manager:**

  The Driver Manager is a library that manages communication between application and driver or drivers. It performs the following tasks:

  - Resolves Data Source Names (DSN). The DSN is a configuration string that identifies a given database driver, database, database host and optionally authentication information that enables an ODBC application to connect to a database using a standardized reference.

    Because the database connectivity information is identified by the DSN, any ODBC compliant application can connect to the data source using the same DSN reference. This eliminates the need to separately configure each application that needs access to a given database; instead you instruct the application to use a pre-configured DSN.

  - Loading and unloading of the driver required to access a specific database as defined within the DSN. For example, if you have configured a DSN that connects to a MySQL database then the driver manager will load the Connector/ODBC driver to enable the ODBC API to communicate with the MySQL host.

  - Processes ODBC function calls or passes them to the driver for processing.

- **Connector/ODBC Driver:**

  The Connector/ODBC driver is a library that implements the functions supported by the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL.

- **DSN Configuration:**

  The ODBC configuration file stores the driver and database information required to connect to the server. It is used by the Driver Manager to determine which driver to be loaded according to the definition in the DSN. The driver uses this to read connection parameters based on the DSN specified. For more information, Sección 25.1.3, “Connector/ODBC Configuration”.

- **MySQL Server:**

  The MySQL database where the information is stored. The database is used as the source of the data (during queries) and the destination for data (during inserts and updates).

ODBC Driver Managers

An ODBC Driver Manager is a library that manages communication between the ODBC-aware application and any drivers. Its main functionality includes:

- Resolving Data Source Names (DSN).
- Driver loading and unloading.
- Processing ODBC function calls or passing them to the driver.

Both Windows and Mac OS X include ODBC driver managers with the operating system. Most ODBC Driver Manager implementations also include an administration application that makes the configuration of DSN and drivers easier. Examples and information on these managers, including Unix ODBC driver managers are listed below:

- Microsoft Windows ODBC Driver Manager (`odbc32.dll`), http://www.microsoft.com/data/.
- Mac OS X includes `ODBC Administrator`, a GUI application that provides a simpler configuration mechanism for the Unix iODBC Driver Manager. You can configure DSN and driver information either through ODBC Administrator or through the iODBC configuration files. This also means that you can test ODBC Administrator configurations using the `iodbctest` command. http://www.apple.com.
- `unixODBC` Driver Manager for Unix (`libodbc.so`). See http://www.unixodbc.org, for more information. The `unixODBC` Driver Manager includes the Connector/ODBC driver 3.51 in the installation package, starting with version `unixODBC` 2.1.2.
- `iODBC` ODBC Driver Manager for Unix (`libiodbc.so`), see http://www.iodbc.org, for more information.

25.1.2. Connector/ODBC Installation

You can install the Connector/ODBC drivers using two different methods, a binary installation and a source installation. The binary installation is the easiest and most straightforward method of installation. Using the source installation methods should only be necessary on platforms where a binary installation package is not available, or in situations where you want to customize or modify the installation process or Connector/ODBC drivers before installation.

25.1.2.1. Where to Get Connector/ODBC

MySQL AB distributes all its products under the General Public License (GPL). You can get a copy of the latest version of Connector/ODBC binaries and sources from the MySQL AB Web site http://dev.mysql.com/downloads/.

For more information about Connector/ODBC, visit http://www.mysql.com/products/myodbc/.

For more information about licensing, visit http://www.mysql.com/company/legal/licensing/.

25.1.2.2. Supported Platforms

Connector/ODBC can be used on all major platforms supported by MySQL. You can install it on:

- Windows 95, 98, Me, NT, 2000, XP, and 2003
- All Unix-like Operating Systems, including: AIX, Amiga, BSDI, DEC, FreeBSD, HP-UX 10/11, Linux, NetBSD, OpenBSD, OS/2, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix
- Mac OS X and Mac OS X Server

If a binary distribution is not available for a particular platform, see Sección 25.1.2.4, “Installing Connector/ODBC from a source distribution”, to build the driver from the original source code. You can contribute the binaries you create to MySQL by sending a mail message to `<myodbc@lists.mysql.com>`, so that it becomes available for other users.

25.1.2.3. Installing Connector/ODBC from a binary distribution

Using a binary distribution offers the most straightforward method for installing Connector/ODBC. If you want more control over the driver, the installation location and or to customize elements of the driver you

will need to build and install from the source. See the Sección 25.1.2.4, “Installing Connector/ODBC from a source distribution”.

Installing Connector/ODBC from a Binary Distribution on Windows

Before installing the Connector/ODBC drivers on Windows you should ensure that your Microsoft Data Access Components (MDAC) are up to date. You can obtain the latest version from the Microsoft Data Access and Storage Web site.

There are three available distribution types to use when installing for Windows. The contents in each case are identical, it is only the installation method which is different.

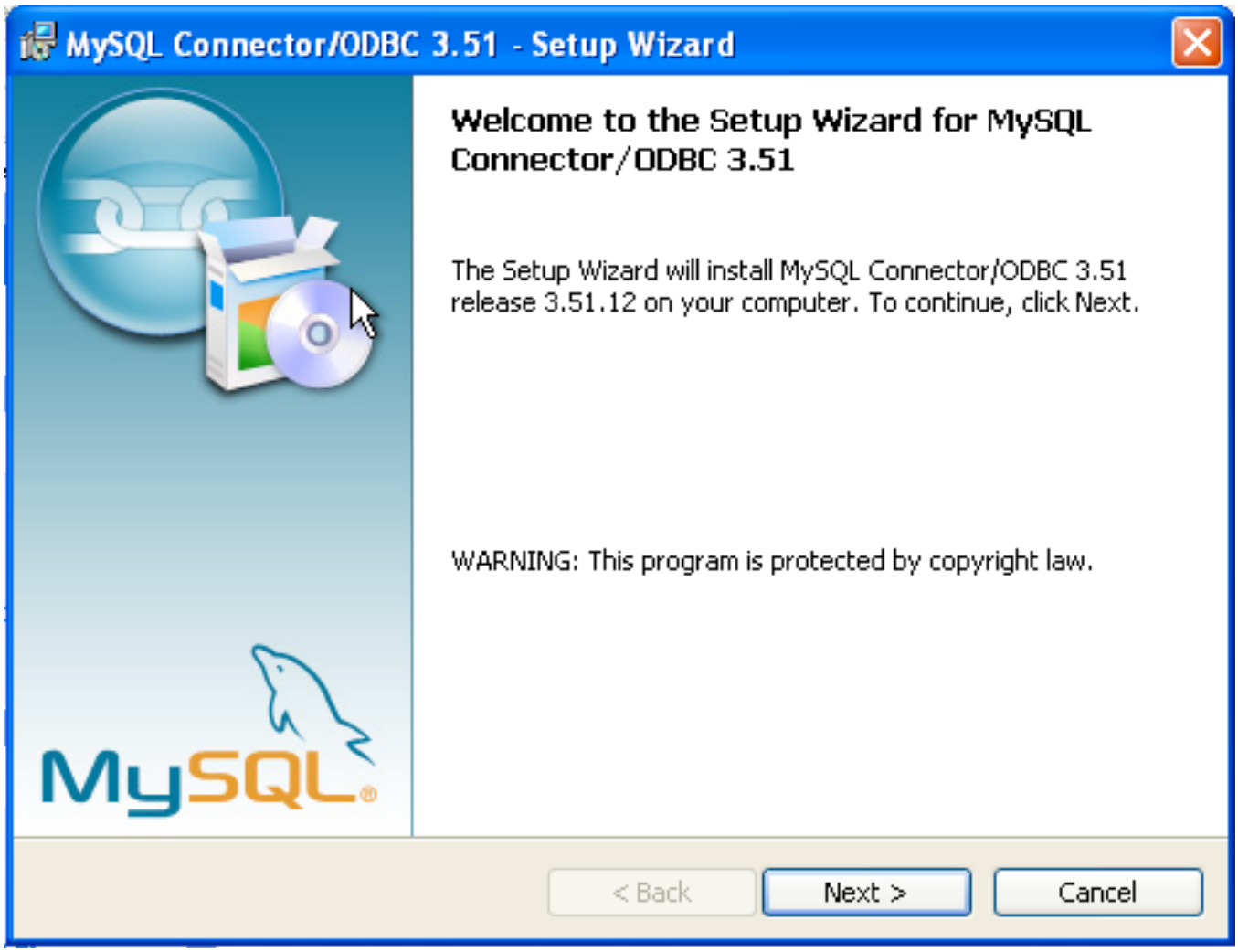
- Zipped installer consists of a Zipped package containing a standalone installation application. To install from this package, you must unzip the installer, and then run the installation application. See “Installing the Windows Connector/ODBC Driver using an installer” to complete the installation.
- MSI installer, an installation file that can be used with the installer included in Windows 2000, Windows XP and Windows Server 2003. See “Installing the Windows Connector/ODBC Driver using an installer” to complete the installation.
- Zipped DLL package, containing the DLL files that need must be manually installed. See “Installing the Windows Connector/ODBC Driver using the Zipped DLL package” to complete the installation.

Installing the Windows Connector/ODBC Driver using an installer

The installer packages offer a very simple method for installing the Connector/ODBC drivers. If you have downloaded the zipped installer then you must extract the installer application. The basic installation process is identical for both installers.

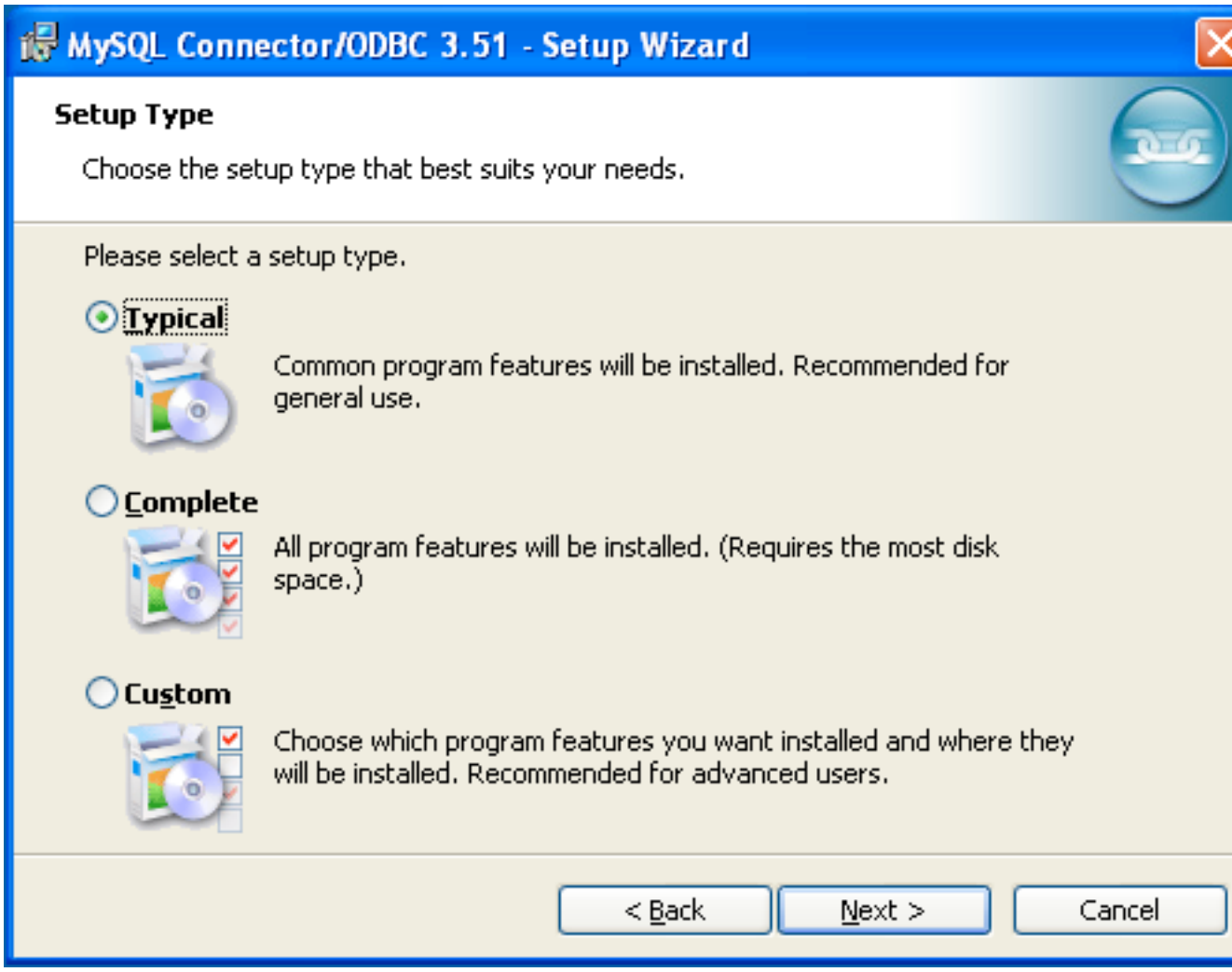
You should follow these steps to complete the installation:

1. Double click on the standalone installer that you extracted, or the MSI file you downloaded.
2. The MySQL Connector/ODBC 3.51 - Setup Wizard will start. Click the Next button to begin the installation process.

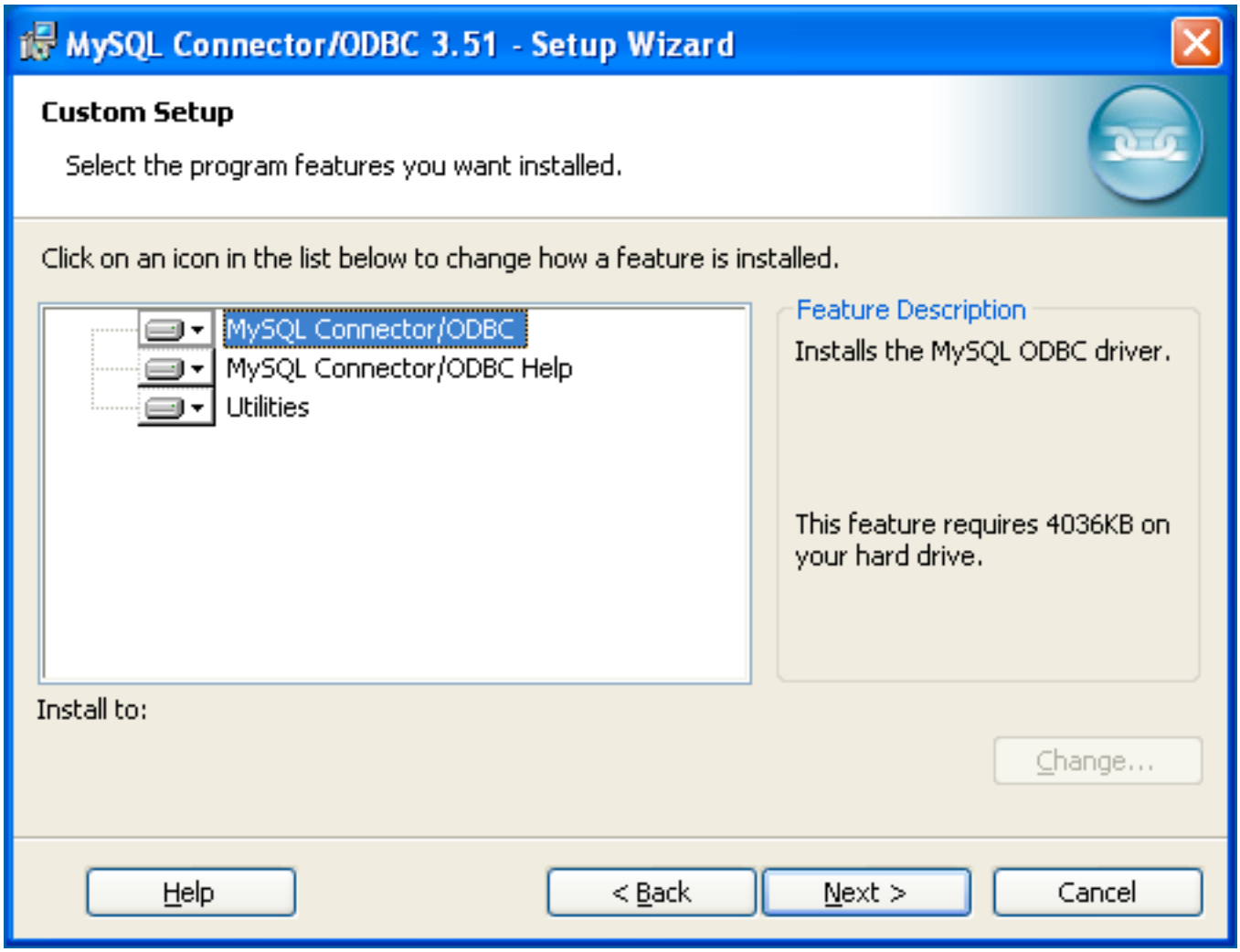


3. You will need to choose the installation type. The Typical installation provides the standard files you will need to connect to a MySQL database using ODBC. The Complete option installs all the available files, including debug and utility components. It is recommended you choose one of these two options to complete the installation. If choose one of these methods, click Next and then proceed to step 5.

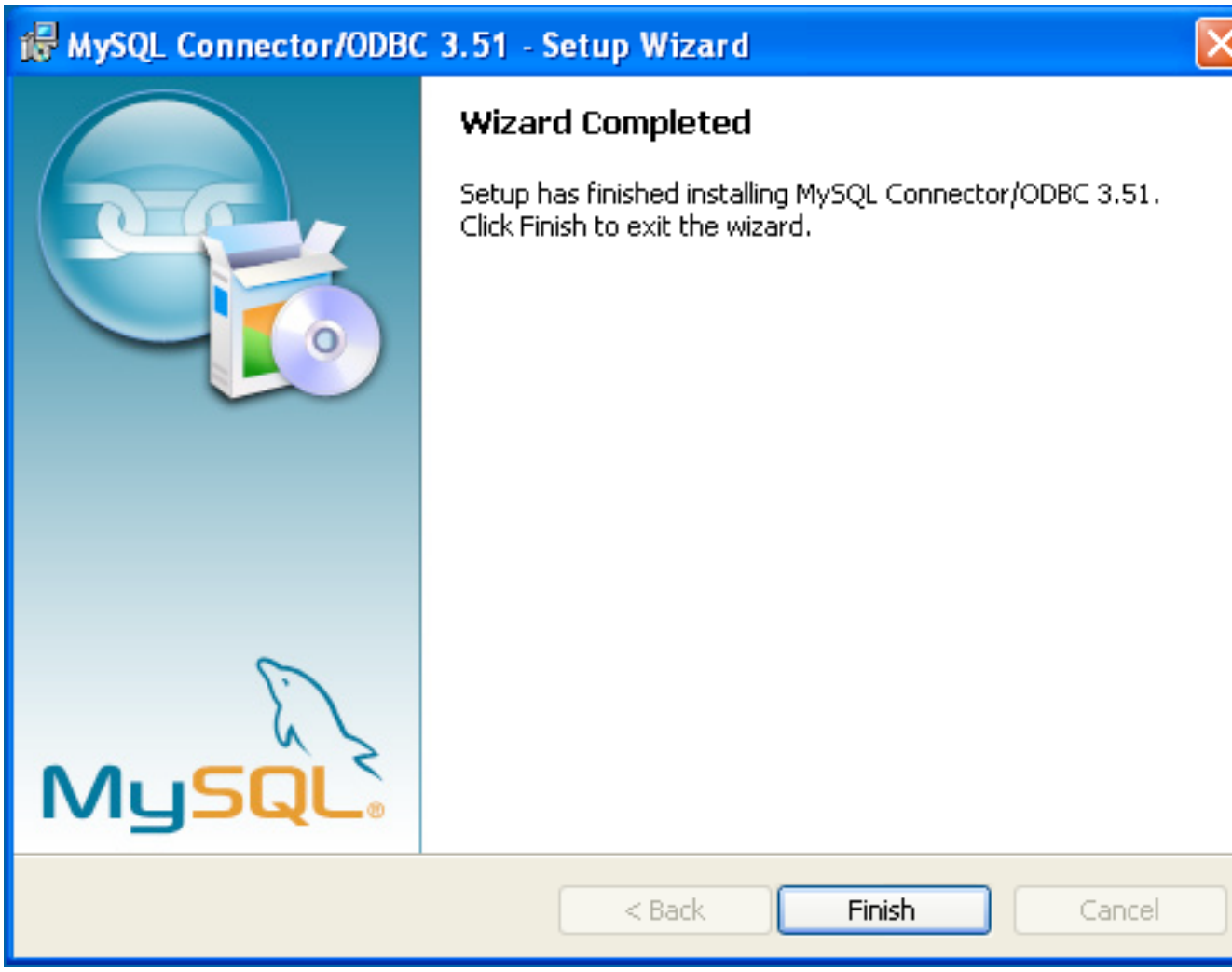
   You may also choose a Custom installation, which enables you to select the individual components that you want to install. You have chosen this method, click Next and then proceed to step 4.



4. If you have chosen a custom installation, use the popups to select which components to install and then click Next to install the necessary files.



5. Once the files have copied to your machine, the installation is complete. Click Finish to exit the installer.



Now the installation is complete, you can continue to configure your ODBC connections using Sección 25.1.3, “Connector/ODBC Configuration”.

Installing the Windows Connector/ODBC Driver using the Zipped DLL package

If you have downloaded the Zipped DLL package then you must install the individual files required for Connector/ODBC operation manually. Once you have unzipped the installation files, you can either perform this operation by hand, executing each statement individually, or you can use the included Batch file to perform an installation to the default locations.

To install using the Batch file:

1. Unzip the Connector/ODBC Zipped DLL package.
2. Open a Command Prompt.
3. Change to the directory created when you unzipped the Connector/ODBC Zipped DLL package.
4. Run `Install.bat`:

```
C:\> Install.bat
```

This will copy the necessary files into the default location, and then register the Connector/ODBC driver with the Windows ODBC manager.

If you want to copy the files to an alternative location - for example, to run or test different versions of the Connector/ODBC driver on the same machine, then you must copy the files by hand. It is however not recommended to install these files in a non-standard location. To copy the files by hand to the default installation location use the following steps:

1. Unzip the Connector/ODBC Zipped DLL package.
2. Open a Command Prompt.
3. Change to the directory created when you unzipped the Connector/ODBC Zipped DLL package.
4. Copy the library files to a suitable directory. The default is to copy them into the default Windows system directory `\Windows\System32`:

```
C:\> copy lib\myodbc3S.dll \Windows\System32
C:\> copy lib\myodbc3S.lib \Windows\System32
C:\> copy lib\myodbc3.dll \Windows\System32
C:\> copy lib\myodbc3.lib \Windows\System32
```

5. Copy the Connector/ODBC tools. These must be placed into a directory that is in the system `PATH`. The default is to install these into the Windows system directory `\Windows\System32`:

```
C:\> copy bin\myodbc3i.exe \Windows\System32
C:\> copy bin\myodbc3m.exe \Windows\System32
C:\> copy bin\myodbc3c.exe \Windows\System32
```

6. Optionally copy the help files. For these files to be accessible through the help system, they must be installed in the Windows system directory:

```
C:\> copy doc\*.hlp \Windows\System32
```

7. Finally, you must register the Connector/ODBC driver with the ODBC manager:

```
C:\> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;\
  DRIVER=myodbc3.dll;SETUP=myodbc3S.dll"
```

You must change the references to the DLL files and command location in the above statement if you have not installed these files into the default location.

Handling Installation Errors

On Windows, you may get the following error when trying to install the older MyODBC 2.50 driver:

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL.
Restart Windows and try installing again (before running any
applications which use ODBC)
```

The reason for the error is that another application is currently using the ODBC system. Windows may not allow you to complete the installation. In most cases, you can continue by pressing `Ignore` to copy the rest of the Connector/ODBC files and the final installation should still work. If it doesn't, the solution is to re-boot your computer in “safe mode.” Choose safe mode by pressing F8 just before your machine starts Windows during re-booting, install the Connector/ODBC drivers, and re-boot to normal mode.

Installing Connector/ODBC from a Binary Distribution on Unix

There are two methods available for installing Connector/ODBC on Unix from a binary distribution. For most Unix environments you will need to use the tarball distribution. For Linux systems, there is also an RPM distribution available.

Installing Connector/ODBC from a Binary Tarball Distribution

To install the driver from a tarball distribution (`.tar.gz` file), download the latest version of the driver for your operating system and follow these steps that demonstrate the process using the Linux version of the tarball:

```
shell> su root
shell> gunzip mysql-connector-odbc-3.51.11-i686-pc-linux.tar.gz
shell> tar xvf mysql-connector-odbc-3.51.11-i686-pc-linux.tar
shell> cd mysql-connector-odbc-3.51.11-i686-pc-linux
```

Read the installation instructions in the `INSTALL-BINARY` file and execute these commands.

```
shell> cp libmyodbc* /usr/local/lib
shell> cp odbc.ini /usr/local/etc
shell> export ODBCINI=/usr/local/etc/odbc.ini
```

Then proceed on to Sección 25.1.3.4, “Configuring a Connector/ODBC DSN on Unix”, to configure the DSN for Connector/ODBC. For more information, refer to the `INSTALL-BINARY` file that comes with your distribution.

Installing Connector/ODBC from an RPM Distribution

To install or upgrade Connector/ODBC from an RPM distribution on Linux, simply download the RPM distribution of the latest version of Connector/ODBC and follow the instructions below. Use `su root` to become `root`, then install the RPM file.

If you are installing for the first time:

```
shell> su root
 shell> rpm -ivh mysql-connector-odbc-3.51.12.i386.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
shell> rpm -Uvh mysql-connector-odbc-3.51.12.i386.rpm
```

If there is any dependency error for MySQL client library, `libmysqlclient`, simply ignore it by supplying the `--nodeps` option, and then make sure the MySQL client shared library is in the path or set through `LD_LIBRARY_PATH`.

This installs the driver libraries and related documents to `/usr/local/lib` and `/usr/share/doc/MyODBC`, respectively. Proceed onto Sección 25.1.3.4, “Configuring a Connector/ODBC DSN on Unix”.

To **uninstall** the driver, become `root` and execute an `rpm` command:

```
shell> su root
shell> rpm -e mysql-connector-odbc
```

Installing Connector/ODBC on Mac OS X

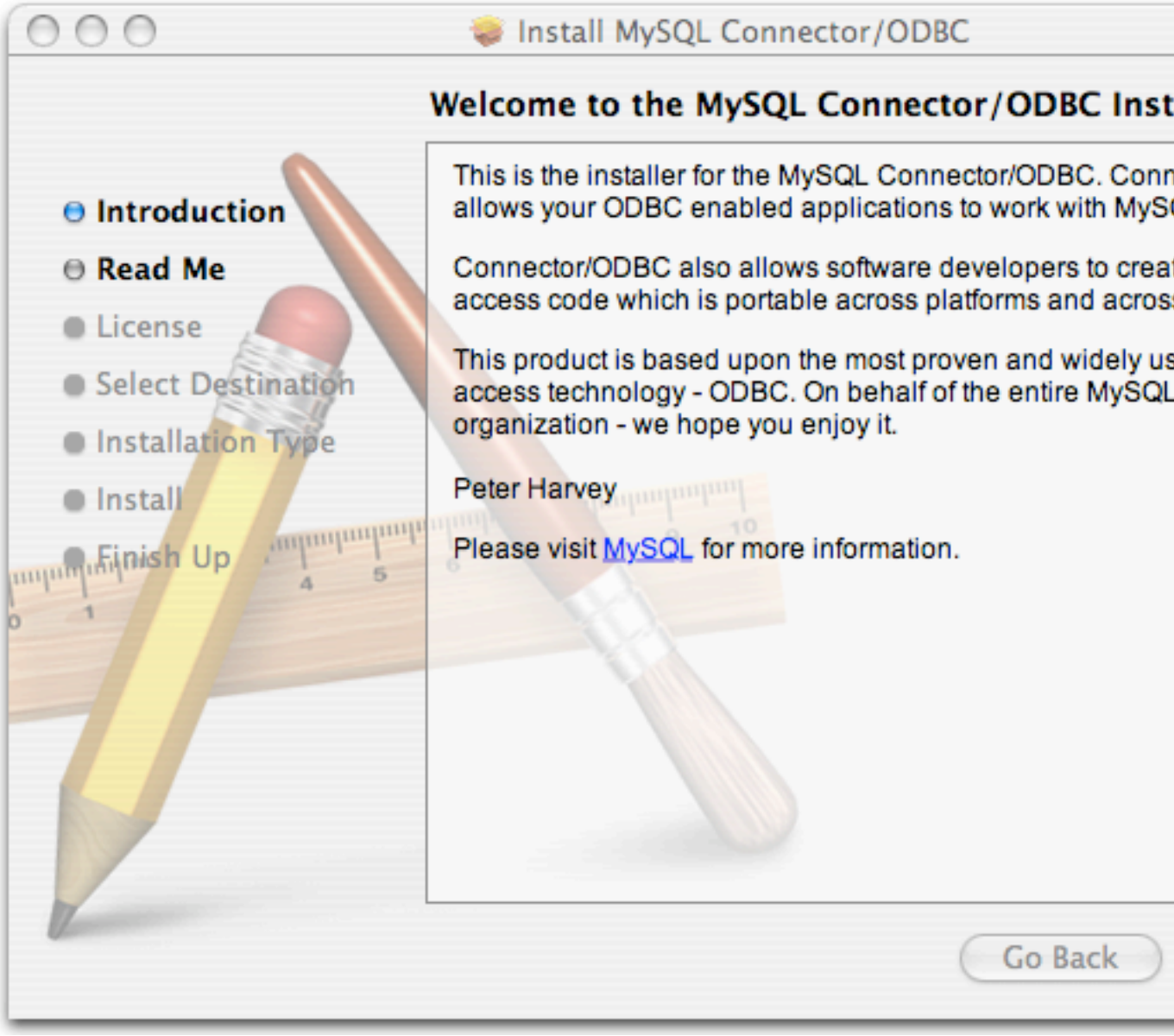
Mac OS X is based on the FreeBSD operating system, and you can normally use the MySQL network port for connecting to MySQL servers on other hosts. Installing the Connector/ODBC driver enables you to connect to MySQL databases on any platform through the ODBC interface. You should only need to install the Connector/ODBC driver when your application requires an ODBC interface. Applications that require or can use ODBC (and therefore the Connector/ODBC driver) include ColdFusion, Filemaker Pro, 4th Dimension and many other applications.

Mac OS X includes its own ODBC manager, based on the `iODBC` manager. Mac OS X includes an administration tool that provides easier administration of ODBC drivers and configuration, updating the underlying `iODBC` configuration files.

Installing the Connector/ODBC Driver

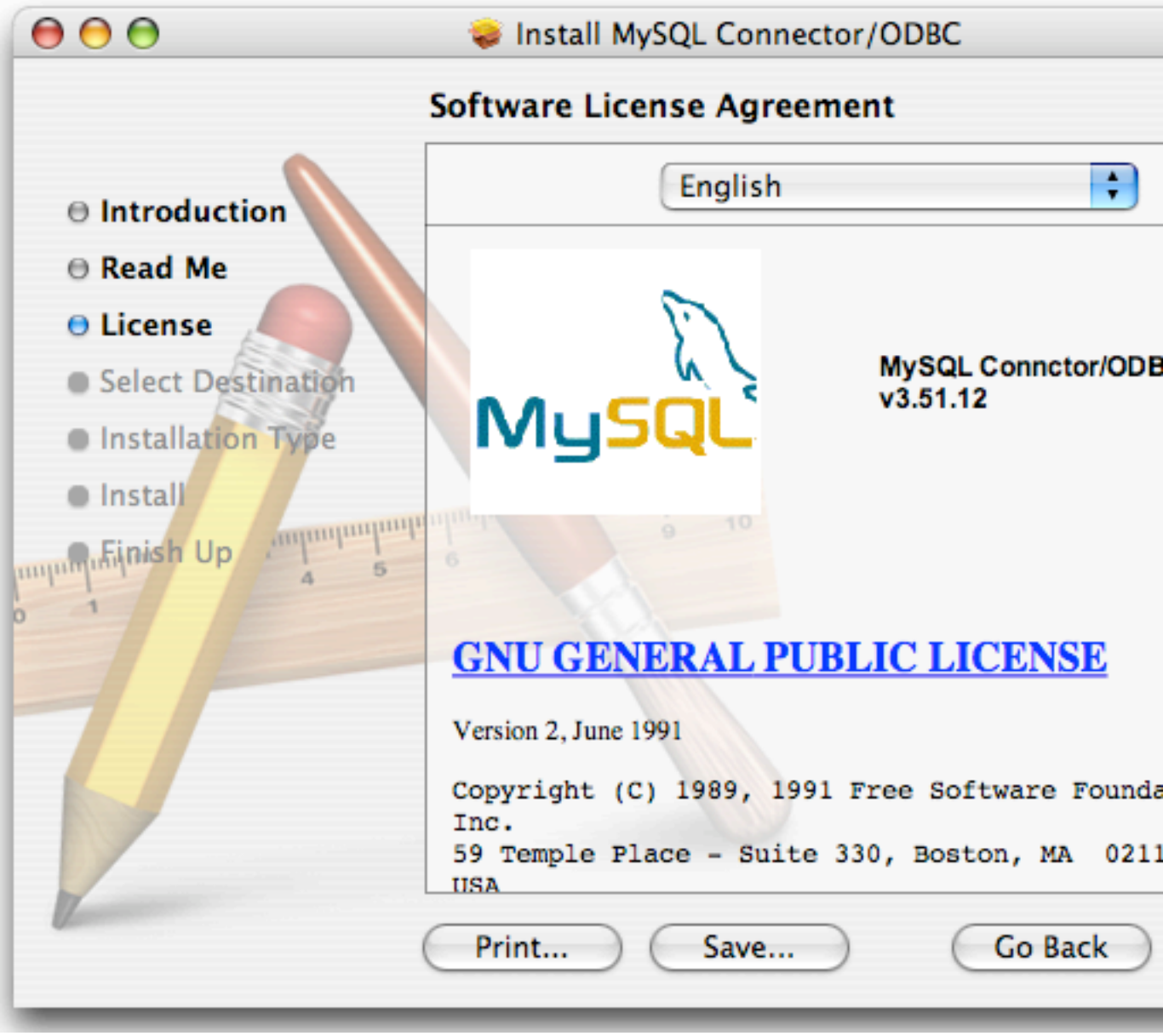
You can install Connector/ODBC on a Mac OS X or Mac OS X Server computer by using the binary distribution. The package is available as a compressed disk image (`.dmg`) file. To install Connector/ODBC on your computer using this method, follow these steps:

1. Download the file to your computer and double-click on the downloaded image file.
2. Within the disk image you will find an installer package (with the `.pkg` extension). Double click on this file to start the Mac OS X installer.
3. You will be presented with the installer welcome message. Click the Continue button to begin the installation process.

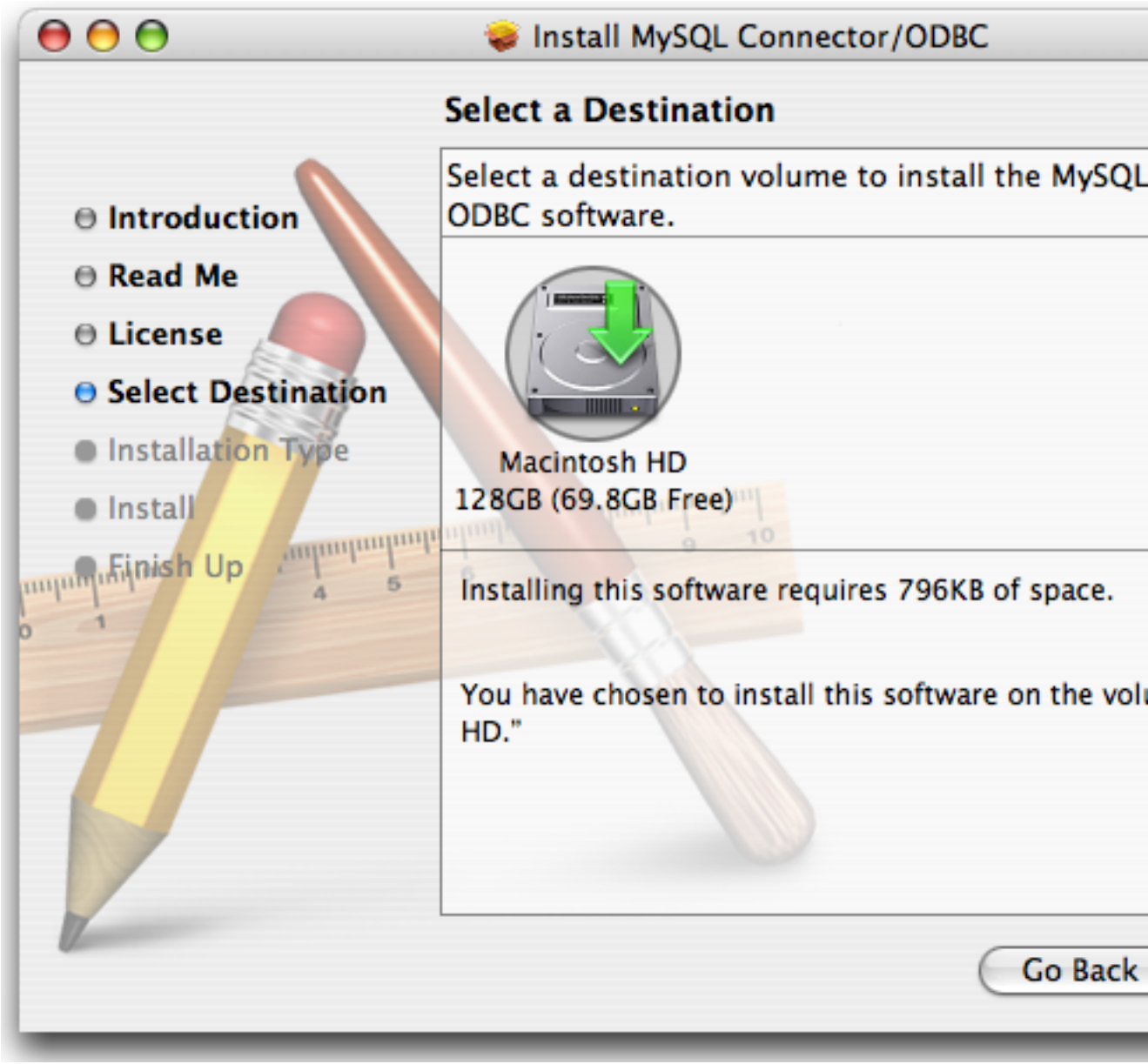


4. Please take the time to read the Important Information as it contains guidance on how to complete the installation process. Once you have read the notice and collected the necessary information, click Continue.

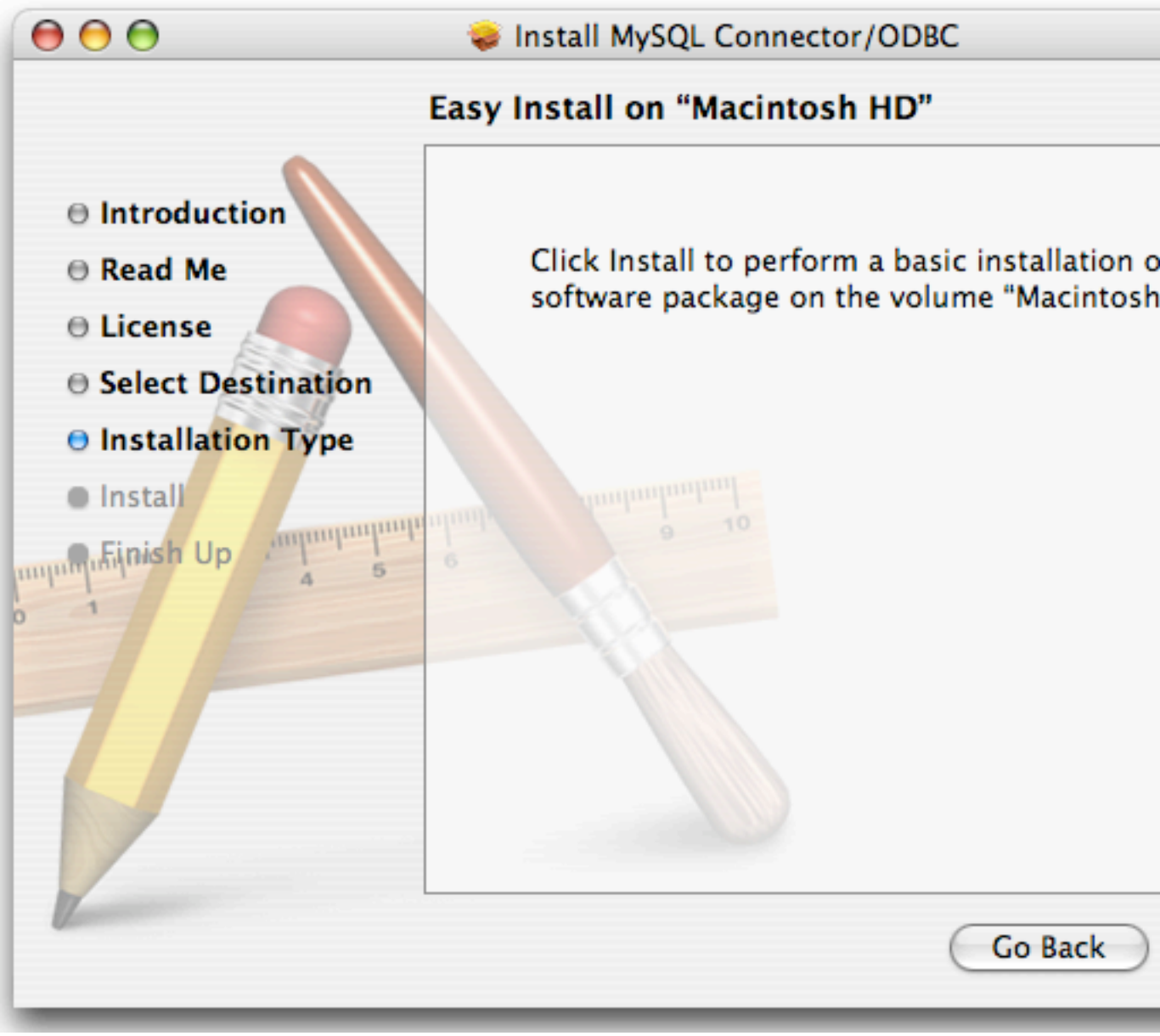
5. Connector/ODBC drivers are made available under the GNU General Public License. Please read the license if you are not familiar with it before continuing installation. Click Continue to approve the license (you will be asked to confirm that decision) and continue the installation.



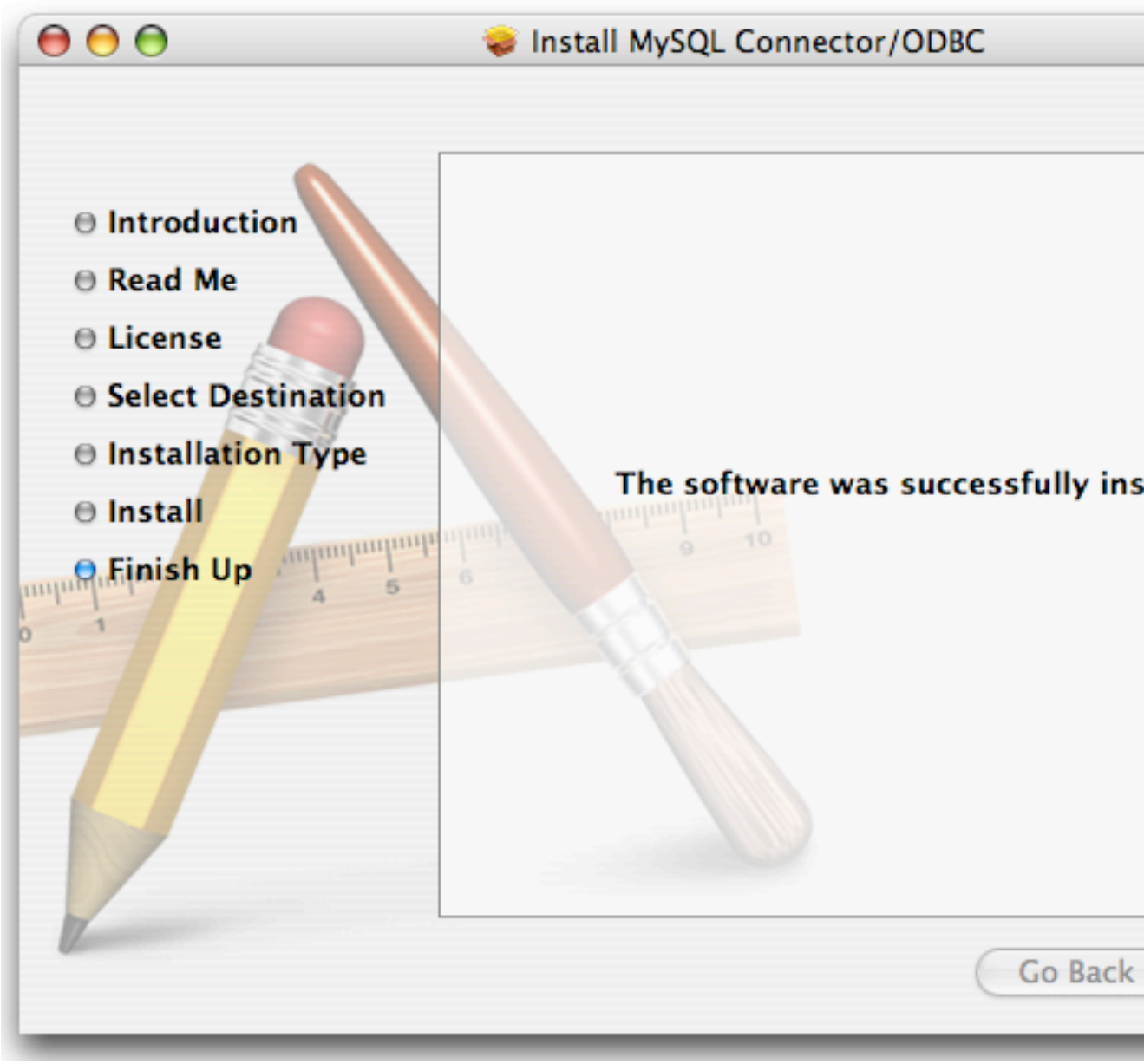
6. Choose a location to install the Connector/ODBC drivers and the ODBC Administrator application. You must install the files onto a drive with an operating system and you may be limited in the choices available. Select the drive you want to use, and then click Continue.



7. The installer will automatically select the files that need to be installed on your machine. Click Install to continue. The installer will copy the necessary files to your machine. A progress bar will be shown indicating the installation progress.



8. When installation has been completed you will get a window like the one shown below. Click Close to close and quit the installer.



25.1.2.4. Installing Connector/ODBC from a source distribution

Installing Connector/ODBC from a source distribution gives you greater flexibility in the contents and installation location of the Connector/ODBC components. It also enables you to build and install Connector/ODBC on platforms where a pre-compiled binary is not available.

Connector/ODBC sources are available either as a downloadable package, or through the revision control system used by the Connector/ODBC developers.

Installing Connector/ODBC from a Source Distribution on Windows

You should only need to install Connector/ODBC from source on Windows if you want to change or modify the source or installation. If you are unsure whether to install from source, please use the binary installation detailed in “Installing Connector/ODBC from a Binary Distribution on Windows”.

Installing Connector/ODBC from source on Windows requires a number of different tools and packages:

- MDAC, Microsoft Data Access SDK from http://www.microsoft.com/data/.
- Suitable C compiler, such as Microsoft Visual C++ or the C compiler included with Microsoft Visual Studio.
- Compatible `make` tool. Microsoft's `nmake` is used in the examples in this section.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit http://dev.mysql.com/downloads/.

Building Connector/ODBC 3.51

Connector/ODBC source distributions include `Makefiles` that require the `nmake` or other `make` utility. In the distribution, you can find `Makefile` for building the release version and `Makefile_debug` for building debugging versions of the driver libraries and DLLs.

To build the driver, use this procedure:

1. Download and extract the sources to a folder, then change directory into that folder. The following command assumes the folder is named `myodbc3-src`:

   ```
   C:\> cd myodbc3-src
   ```

2. Edit `Makefile` to specify the correct path for the MySQL client libraries and header files. Then use the following commands to build and install the release version:

   ```
   C:\> nmake -f Makefile
   C:\> nmake -f Makefile install
   ```

   `nmake -f Makefile` builds the release version of the driver and places the binaries in subdirectory called `Release`.

   `nmake -f Makefile install` installs (copies) the driver DLLs and libraries (`myodbc3.dll`, `myodbc3.lib`) to your system directory.

3. To build the debug version, use `Makefile_Debug` rather than `Makefile`, as shown below:

   ```
   C:\> nmake -f Makefile_debug
   C:\> nmake -f Makefile_debug install
   ```

4. You can clean and rebuild the driver by using:

   ```
   C:\> nmake -f Makefile clean
   C:\> nmake -f Makefile install
   ```

Note:

- Make sure to specify the correct MySQL client libraries and header files path in the Makefiles (set the `MYSQL_LIB_PATH` and `MYSQL_INCLUDE_PATH` variables). The default header file path is assumed to be `C:\mysql\include`. The default library path is assumed to be `C:\mysql\lib\opt` for release DLLs and `C:\mysql\lib\debug` for debug versions.
- For the complete usage of `nmake`, visit http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfRunningNMAKE.asp.

- If you are using the Subversion tree for compiling, all Windows-specific `Makefiles` are named as `Win_Makefile*`.

Testing

After the driver libraries are copied/installed to the system directory, you can test whether the libraries are properly built by using the samples provided in the `samples` subdirectory:

```
C:\> cd samples
C:\> nmake -f Makefile all
```

Building MyODBC 2.50

The MyODBC 2.50 source distribution includes VC workspace files. You can build the driver using these files (`.dsp` and `.dsw`) directly by loading them from Microsoft Visual Studio 6.0 or higher.

Installing Connector/ODBC from a Source Distribution on Unix

You need the following tools to build MySQL from source on Unix:

- A working ANSI C++ compiler. `gcc` 2.95.2 or later, `egcs` 1.0.2 or later or `egcs 2.91.66`, SGI C++, and SunPro C++ are some of the compilers that are known to work.
- A good `make` program. GNU `make` is always recommended and is sometimes required.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit http://dev.mysql.com/downloads/.

  If you have built your own MySQL server and/or client libraries from source then you must have used the `--enable-thread-safe-client` option to `configure` when the libraries were built.

  You should also ensure that the `libmysqlclient` library were built and installed as a shared library.
- A compatible ODBC manager must be installed. Connector/ODBC is known to work with the `iODBC` and `unixODBC` managers. See “ODBC Driver Managers”, for more information.
- If you are using a character set that isn't compiled into the MySQL client library then you need to install the MySQL character definitions from the `charsets` directory into *`SHAREDIR`* (by default, `/usr/local/mysql/share/mysql/charsets`). These should be in place if you have installed the MySQL server on the same machine. See Capítulo 10, *Soporte de conjuntos de caracteres*, for more information on character set support.

Once you have all the required files, unpack the source files to a separate directory, you then have to run `configure` and build the library using `make`.

Typical `configure` Options

The `configure` script gives you a great deal of control over how you configure your Connector/ODBC build. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. For a list of options and environment variables supported by `configure`, run this command:

```
shell> ./configure --help
```

Some of the more commonly used `configure` options are described here:

1. To compile Connector/ODBC, you need to supply the MySQL client include and library files path using the `--with-mysql-path=DIR` option, where *`DIR`* is the directory where MySQL is installed.

   MySQL compile options can be determined by running *`DIR`*`/bin/mysql_config`.

2. Supply the standard header and library files path for your ODBC Driver Manager (`iODBC` or `unixODBC`).

   - If you are using `iODBC` and `iODBC` is not installed in its default location (`/usr/local`), you might have to use the `--with-iodbc=DIR` option, where *`DIR`* is the directory where `iODBC` is installed.

     If the `iODBC` headers do not reside in *`DIR`*`/include`, you can use the `--with-iodbc-includes=INCDIR` option to specify their location.

     The applies to libraries. If they are not in *`DIR`*`/lib`, you can use the `--with-iodbc-libs=LIBDIR` option.

   - If you are using `unixODBC`, use the `--with-unixODBC=DIR` option (case sensitive) to make `configure` look for `unixODBC` instead of `iODBC` by default, *`DIR`* is the directory where `unixODBC` is installed.

     If the `unixODBC` headers and libraries aren't located in *`DIR`*`/include` and *`DIR`*`/lib`, use the `--with-unixODBC-includes=INCDIR` and `--with-unixODBC-libs=LIBDIR` options.

3. You might want to specify an installation prefix other than `/usr/local`. For example, to install the Connector/ODBC drivers in `/usr/local/odbc/lib`, use the `--prefix=/usr/local/odbc` option.

The final configuration command looks something like this:

```
shell> ./configure --prefix=/usr/local \
         --with-iodbc=/usr/local \
         --with-mysql-path=/usr/local/mysql
```

Additional configure Options

There are a number of other options that you need, or want, to set when configuring the Connector/ODBC driver before it is built.

- To link the driver with MySQL thread safe client libraries `libmysqlclient_r.so` or `libmysqlclient_r.a`, you must specify the following `configure` option:

  ```
  --enable-thread-safe
  ```

  and can be disabled (default) using

  ```
  --disable-thread-safe
  ```

  This option enables the building of the driver thread-safe library `libmyodbc3_r.so` from by linking with MySQL thread-safe client library `libmysqlclient_r.so` (The extensions are OS dependent).

  If the compilation with the thread-safe option fails, it may be because the correct thread-libraries on the system could not be located. You should set the value of `LIBS` to point to the correct thread library for your system.

  ```
  LIBS="-lpthread" ./configure ..
  ```

- You can enable or disable the shared and static versions of Connector/ODBC using these options:

```
--enable-shared[=yes/no]
--disable-shared
--enable-static[=yes/no]
--disable-static
```

- By default, all the binary distributions are built as non-debugging versions (configured with `--without-debug`).

  To enable debugging information, build the driver from source distribution and use the `--with-debug` option when you run `configure`.

- This option is available only for source trees that have been obtained from the Subversion repository. This option does not apply to the packaged source distributions.

  By default, the driver is built with the `--without-docs` option. If you would like the documentation to be built, then execute `configure` with:

```
--with-docs
```

Building and Compilation

To build the driver libraries, you have to just execute `make`.

```
shell> make
```

If any errors occur, correct them and continue the build process. If you aren't able to build, then send a detailed email to `<myodbc@lists.mysql.com>` for further assistance.

Building Shared Libraries

On most platforms, MySQL does not build or support `.so` (shared) client libraries by default. This is based on our experience of problems when building shared libraries.

In cases like this, you have to download the MySQL distribution and configure it with these options:

```
--without-server --enable-shared
```

To build shared driver libraries, you must specify the `--enable-shared` option for `configure`. By default, `configure` does not enable this option.

If you have configured with the `--disable-shared` option, you can build the `.so` file from the static libraries using the following commands:

```
shell> cd mysql-connector-odbc-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
          $CC -bundle -flat_namespace -undefined error \
          -o .libs/libmyodbc3-3.51.01.so \
          catalog.o connect.o cursor.o dll.o error.o execute.o \
          handle.o info.o misc.o myodbc3.o options.o prepare.o \
          results.o transact.o utility.o \
          -L/usr/local/mysql/lib/mysql/ \
          -L/usr/local/iodbc/lib/ \
```

```
          -lz -lc -lmysqlclient -liodbcinst
```

Make sure to change `-liodbcinst` to `-lodbcinst` if you are using `unixODBC` instead of `iODBC`, and configure the library paths accordingly.

This builds and places the `libmyodbc3-3.51.01.so` file in the `.libs` directory. Copy this file to the Connector/ODBC library installation directory (`/usr/local/lib` (or the `lib` directory under the installation directory that you supplied with the `--prefix`).

```
shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
          $CC -bundle -flat_namespace -undefined error
          -o .libs/libmyodbc3_r-3.51.01.so
          catalog.o connect.o cursor.o dll.o error.o execute.o
          handle.o info.o misc.o myodbc3.o options.o prepare.o
          results.o transact.o utility.o
          -L/usr/local/mysql/lib/mysql/
          -L/usr/local/iodbc/lib/
          -lz -lc -lmysqlclient_r -liodbcinst
```

Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

That command installs one of the following sets of libraries:

For Connector/ODBC 3.51:

- `libmyodbc3.so`
- `libmyodbc3-3.51.01.so`, where 3.51.01 is the version of the driver
- `libmyodbc3.a`

For thread-safe Connector/ODBC 3.51:

- `libmyodbc3_r.so`
- `libmyodbc3-3_r.51.01.so`
- `libmyodbc3_r.a`

For MyODBC 2.5.0:

- `libmyodbc.so`
- `libmyodbc-2.50.39.so`, where 2.50.39 is the version of the driver
- `libmyodbc.a`

For more information on build process, refer to the `INSTALL` file that comes with the source distribution. Note that if you are trying to use the `make` from Sun, you may end up with errors. On the other hand, GNU `gmake` should work fine on all platforms.

Testing Connector/ODBC on Unix

To run the basic samples provided in the distribution with the libraries that you built, use the following command:

```
shell> make test
```

Before running the tests, create the DSN 'myodbc3' in `odbc.ini` and set the environment variable `ODBCINI` to the correct `odbc.ini` file; and MySQL server is running. You can find a sample `odbc.ini` with the driver distribution.

You can even modify the `samples/run-samples` script to pass the desired DSN, UID, and PASSWORD values as the command-line arguments to each sample.

Building Connector/ODBC from Source on Mac OS X

To build the driver on Mac OS X (Darwin), make use of the following `configure` example:

```
shell> ./configure --prefix=/usr/local
          --with-unixODBC=/usr/local
          --with-mysql-path=/usr/local/mysql
          --disable-shared
          --enable-gui=no
          --host=powerpc-apple
```

The command assumes that the `unixODBC` and MySQL are installed in the default locations. If not, configure accordingly.

On Mac OS X, `--enable-shared` builds `.dylib` files by default. You can build `.so` files like this:

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
          $CC -bundle -flat_namespace -undefined error
          -o .libs/libmyodbc3-3.51.01.so *.o
          -L/usr/local/mysql/lib/
          -L/usr/local/iodbc/lib
          -liodbcinst -lmysqlclient -lz -lc
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
          $CC -bundle -flat_namespace -undefined error
          -o .libs/libmyodbc3-3.51.01.so *.o
          -L/usr/local/mysql/lib/
          -L/usr/local/iodbc/lib
          -liodbcinst -lmysqlclienti_r -lz -lc -lpthread
```

Make sure to change the `-liodbcinst` to `-lodbcinst` in case of using `unixODBC` instead of `iODBC` and configure the libraries path accordingly.

In Apple's version of GCC, both `cc` and `gcc` are actually symbolic links to `gcc3`.

Copy this library to the `$prefix/lib` directory and symlink to `libmyodbc3.so`.

You can cross-check the output shared-library properties using this command:

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

Building Connector/ODBC from Source on HP-UX

To build the driver on HP-UX 10.x or 11.x, make use of the following `configure` example:

If using `cc`:

```
shell> CC="cc" \
          CFLAGS="+z" \
          LDFLAGS="-Wl,+b:-Wl,+s" \
          ./configure --prefix=/usr/local
          --with-unixodbc=/usr/local
          --with-mysql-path=/usr/local/mysql/lib/mysql
          --enable-shared
          --enable-thread-safe
```

If using `gcc`:

```
shell> CC="gcc" \
          LDFLAGS="-Wl,+b:-Wl,+s" \
          ./configure --prefix=/usr/local
          --with-unixodbc=/usr/local
          --with-mysql-path=/usr/local/mysql
          --enable-shared
          --enable-thread-safe
```

Once the driver is built, cross-check its attributes using `chatr .libs/libmyodbc3.sl` to determine whether you need to have set the MySQL client library path using the `SHLIB_PATH` environment variable. For static versions, ignore all shared-library options and run `configure` with the `--disable-shared` option.

Building Connector/ODBC from Source on AIX

To build the driver on AIX, make use of the following `configure` example:

```
shell> ./configure --prefix=/usr/local
          --with-unixodbc=/usr/local
          --with-mysql-path=/usr/local/mysql
          --disable-shared
          --enable-thread-safe
```

NOTE: For more information about how to build and set up the static and shared libraries across the different platforms refer to ' Using static and shared libraries across platforms'.

Installing Connector/ODBC from the Development Source Tree

Caution: You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/ODBC up and running on your system, you should use a standard release distribution.

To be able to access the Connector/ODBC source tree, you must have Subversion installed. Subversion is freely available from http://subversion.tigris.org/.

To build from the source trees, you need the following tools:

- autoconf 2.52 (or newer)
- automake 1.4 (or newer)
- libtool 1.4 (or newer)
- m4

The most recent development source tree is available from our public Subversion trees at http://dev.mysql.com/tech-resources/sources.html.

To checkout out the Connector/ODBC sources, change to the directory where you want the copy of the Connector/ODBC tree to be stored, then use the following command:

```
shell> svn co http://svn.mysql.com/svnpublic/connector-odbc3
```

You should now have a copy of the entire Connector/ODBC source tree in the directory `connector-odbc3`. To build from this source tree on Unix or Linux follow these steps:

```
shell> cd connector-odbc3
shell> aclocal
shell> autoheader
shell> autoconf
shell> automake;
shell> ./configure  # Add your favorite options here
shell> make
```

For more information on how to build, refer to the `INSTALL` file located in the same directory. For more information on options to `configure`, see "Typical `configure` Options"

When the build is done, run `make install` to install the Connector/ODBC 3.51 driver on your system.

If you have gotten to the `make` stage and the distribution does not compile, please report it to `<myodbc@lists.mysql.com>`.

On Windows, make use of Windows Makefiles `WIN-Makefile` and `WIN-Makefile_debug` in building the driver. For more information, see "Installing Connector/ODBC from a Source Distribution on Windows".

After the initial checkout operation to get the source tree, you should run `svn update` periodically update your source according to the latest version.

25.1.3. Connector/ODBC Configuration

Before you connect to a MySQL database using the Connector/ODBC driver you must configure an ODBC *Data Source Name*. The DSN associates the various configuration parameters required to communicate with a database to a specific name. You use the DSN in an application to communicate with the database, rather than specifying individual parameters within the application itself. DSN information can be user specific, system specific, or provided in a special file. ODBC data source names are configured in different ways, depending on your platform and ODBC driver.

25.1.3.1. Data Source Names

A Data Source Name associates the configuration parameters for communicating with a specific database. Generally a DSN consists of the following parameters:

- Name
- Hostname

- Database Name
- Login
- Password

In addition, different ODBC drivers, including Connector/ODBC, may accept additional driver-specific options and parameters.

There are three types of DSN:

- A *System DSN* is a global DSN definition that is available to any user and application on a particular system. A System DSN can normally only be configured by a systems administrator, or by a user who has specific permissions that let them create System DSNs.
- A *User DSN* is specific to an individual user, and can be used to store database connectivity information that the user regularly uses.
- A *File DSN* uses a simple file to define the DSN configuration. File DSNs can be shared between users and machines and are therefore more practical when installing or deploying DSN information as part of an application across many machines.

DSN information is stored in different locations depending on your platform and environment.

25.1.3.2. Configuring a Connector/ODBC DSN on Windows

The `ODBC Data Source Administrator` within Windows enables you to create DSNs, check driver installation and configure ODBC systems such as tracing (used for debugging) and connection pooling.

Different editions and versions of Windows store the `ODBC Data Source Administrator` in different locations depending on the version of Windows that you are using.

To open the `ODBC Data Source Administrator` in Windows Server 2003:

1. On the `Start` menu, choose `Administrative Tools`, and then click `Data Sources (ODBC)`.

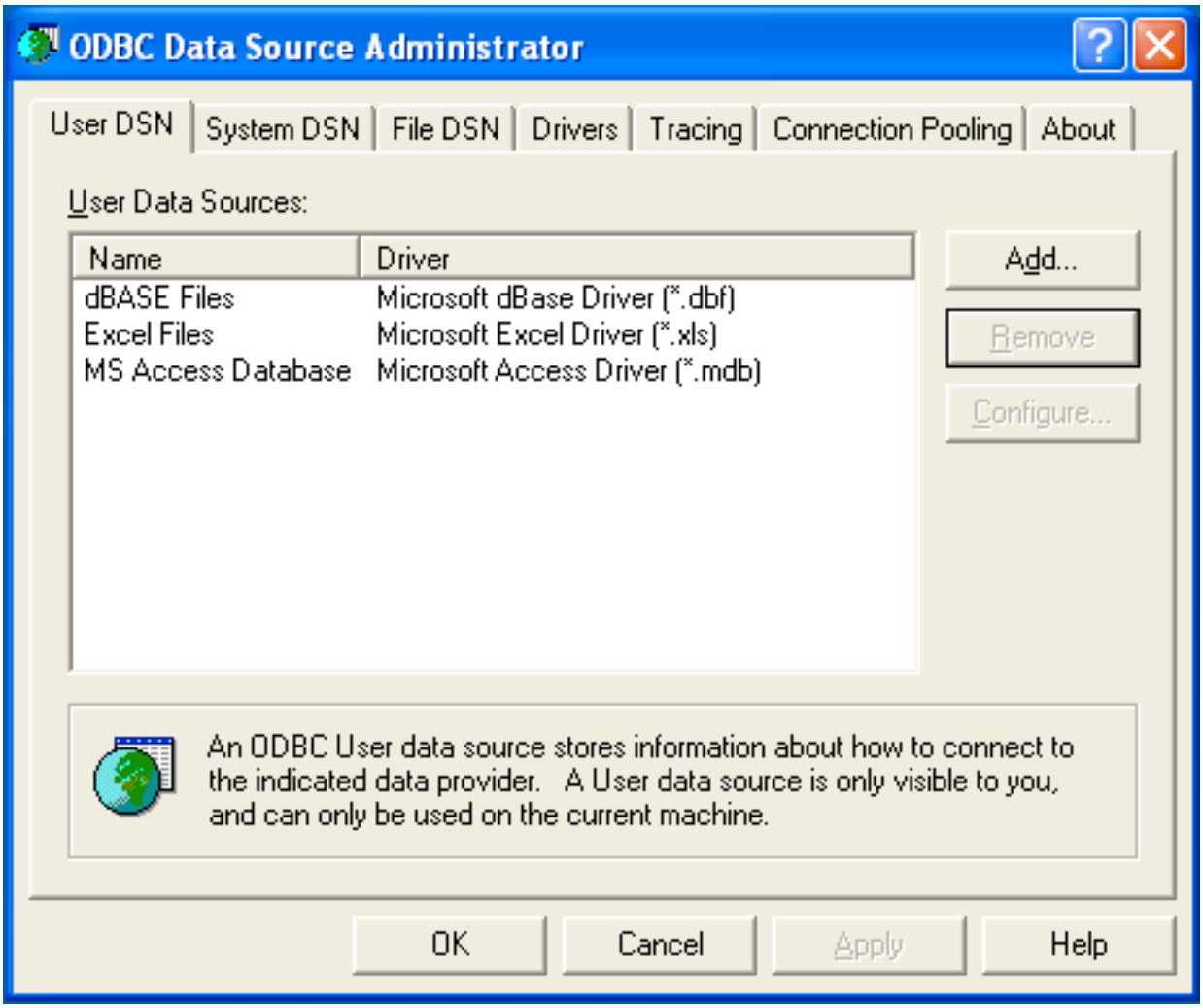
To open the `ODBC Data Source Administrator` in Windows 2000 Server or Windows 2000 Professional:

1. On the `Start` menu, choose `Settings`, and then click `Control Panel`.
2. In `Control Panel`, click `Administrative Tools`.
3. In `Administrative Tools`, click `Data Sources (ODBC)`.

To open the `ODBC Data Source Administrator` on Windows XP:

1. On the `Start` menu, click `Control Panel`.
2. In the `Control Panel` when in `Category View` click `Performance and Maintenance` and then click `Administrative Tools`.. If you are viewing the `Control Panel` in `Classic View`, click `Administrative Tools`.
3. In `Administrative Tools`, click `Data Sources (ODBC)`.

Irrespective of your Windows version, you should be presented the `ODBC Data Source Administrator` window:



Within Windows XP, you can add the `Administrative Tools` folder to your Start menu to make it easier to locate the ODBC Data Source Administrator. To do this:

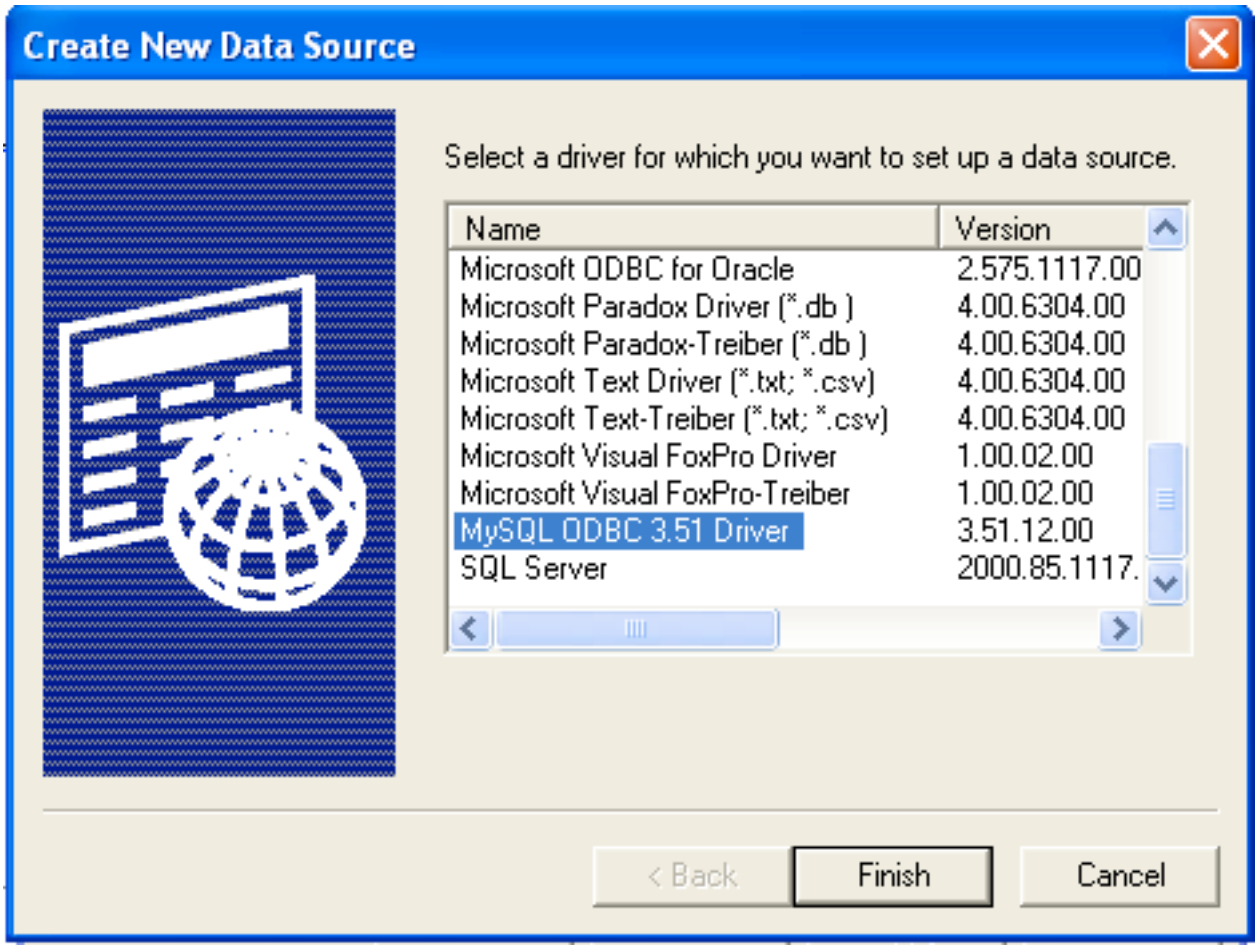
1. Right click on the Start menu.
2. Select `Properties`.
3. Click Customize....
4. Select the Advanced tab.
5. Within `Start menu items`, within the `System Administrative Tools` section, select `Display on the All Programs menu`.

Within both Windows Server 2003 and Windows XP you may want to permanently add the `ODBC Data Source Administrator` to your Start menu. To do this, locate the `Data Sources (ODBC)` icon using the methods shown, then right-click on the icon and then choose Pin to Start Menu.

Adding a Connector/ODBC DSN on Windows

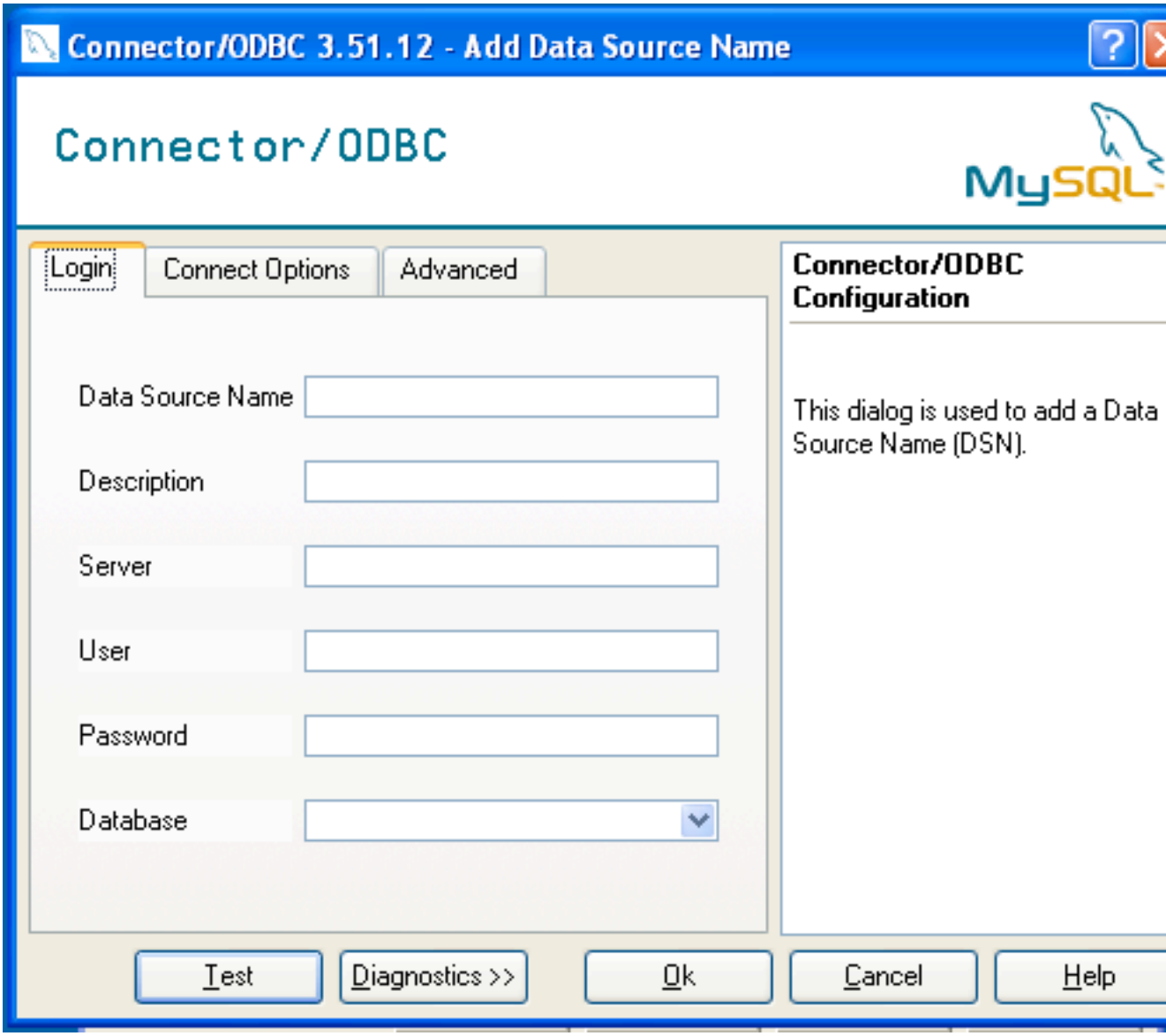
To add and configure a new Connector/ODBC data source on Windows, use the `ODBC Data Source Administrator`:

1. Open the `ODBC Data Source Administrator`.
2. To create a System DSN (which will be available to all users) , select the `System DSN` tab. To create a User DSN, which will be unique only to the current user, click the Add... button.
3. You will need to select the ODBC driver for this DSN.



   Select `MySQL ODBC 3.51 Driver`, then click `Finish`.

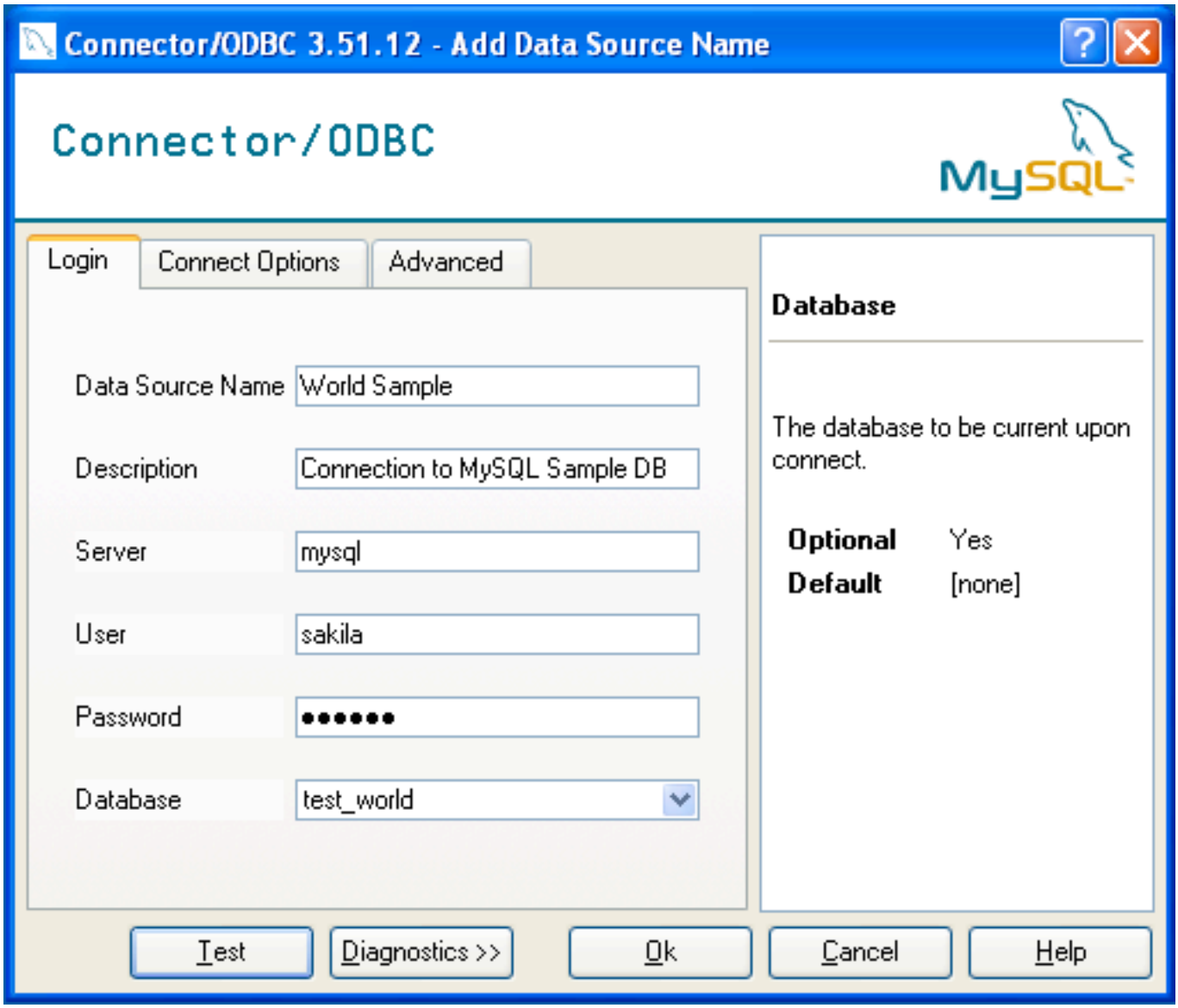
4. You now need to configure the specific fields for the DSN you are creating through the `Add Data Source Name` dialog.



In the `Data Source Name` box, enter the name of the data source you want to access. It can be any valid name that you choose.

5. In the `Description` box, enter some text to help identify the connection.
6. In the `Server` field, enter the name of the MySQL server host that you want to access. By default, it is `localhost`.
7. In the `User` field, enter the user name to use for this connection.
8. In the `Password` field, enter the corresponding password for this connection.
9. The `Database` popup should automatically populate with the list of databases that the user has permissions to access.
10. Click OK to save the DSN.

A completed DSN configuration may look like this:



Checking Connector/ODBC DSN Configuration on Windows

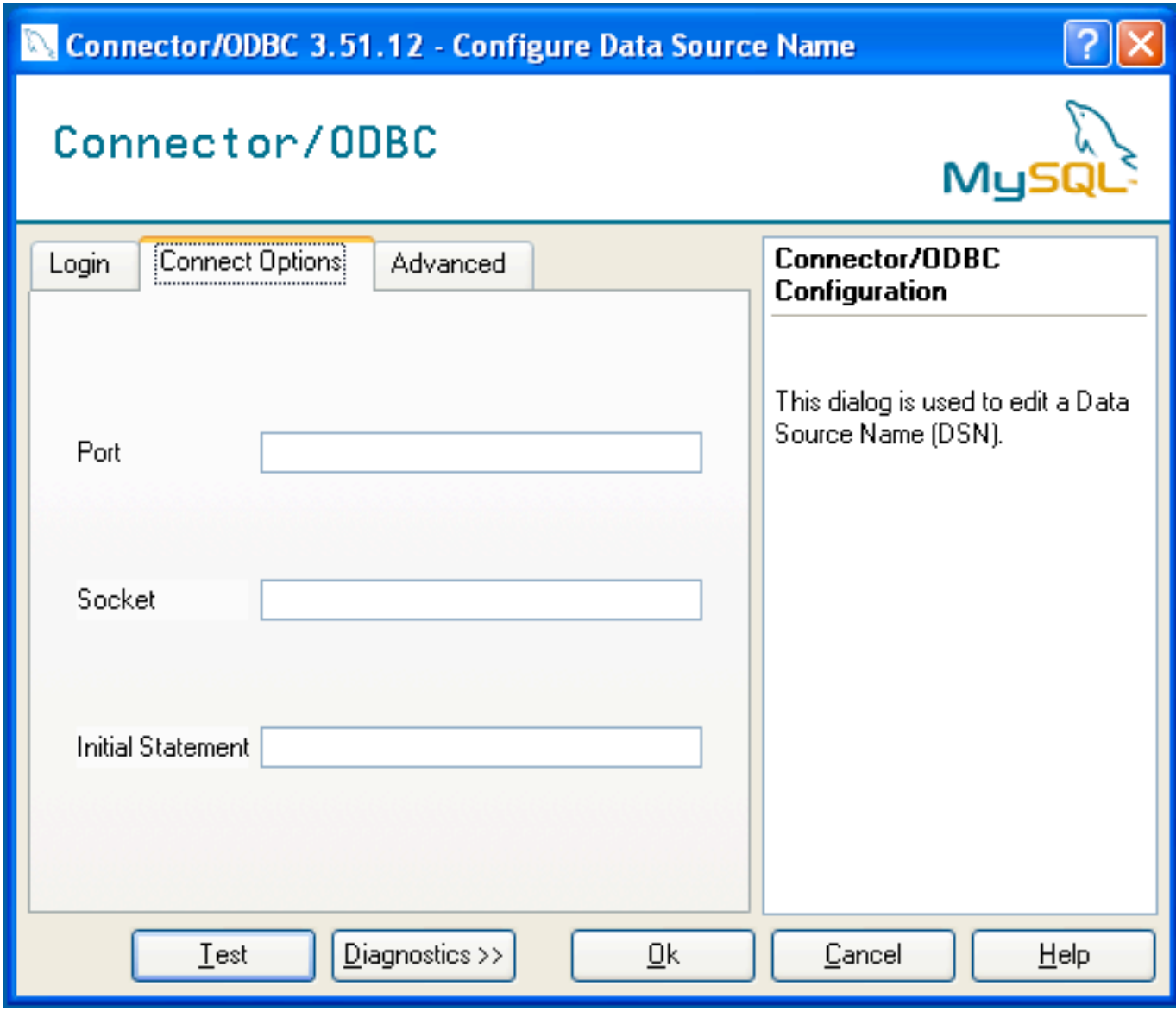
You can verify the connection using the parameters you have entered by clicking the Test button. If the connection could be made successfully, you will be notified with a `Success; connection was made!` dialog.

If the connection failed, you can obtain more information on the test and why it may have failed by clicking the Diagnostics... button to show additional error messages.

Connector/ODBC DSN Configuration Options

You can configure a number of options for a specific DSN by using either the Connect Options or Advanced tabs in the DSN configuration dialog.

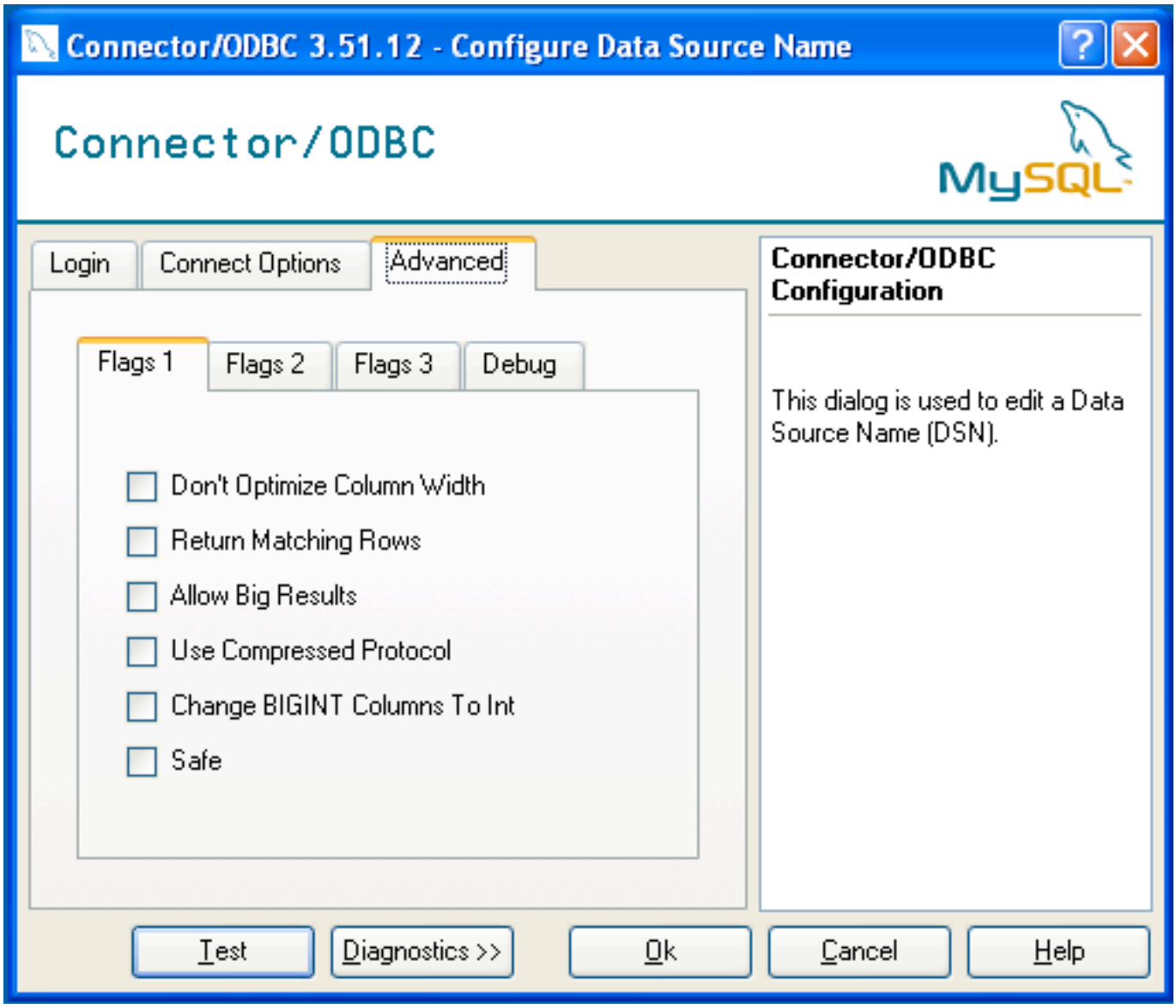
The Connection Options dialog can be seen below.



The three options you can configure are:

- `Port` sets the TCP/IP port number to use when communicating with MySQL. Communication with MySQL uses port 3306 by default. If your server is configured to use a different TCP/IP port, you must specify that port number here.
- `Socket` sets the name or location of a specific socket or Windows pipe to use when communicating with MySQL.
- `Initial Statement` defines an SQL statement that will be executed when the connection to MySQL is opened. You can use this to set MySQL options for your connection, such as setting the default character set or database to use during your connection.

The Advanced tab enables you to configure Connector/ODBC connection parameters. Refer to Sección 25.1.3.5, “Connector/ODBC Connection Parameters”, for information about the meaning of these options.



Errors and Debugging

This section answers Connector/ODBC connection-related questions.

- **While configuring a Connector/ODBC DSN, a `Could Not Load Translator or Setup Library` error occurs**

  For more information, refer to MS KnowledgeBase Article(Q260558). Also, make sure you have the latest valid `ctl3d32.dll` in your system directory.

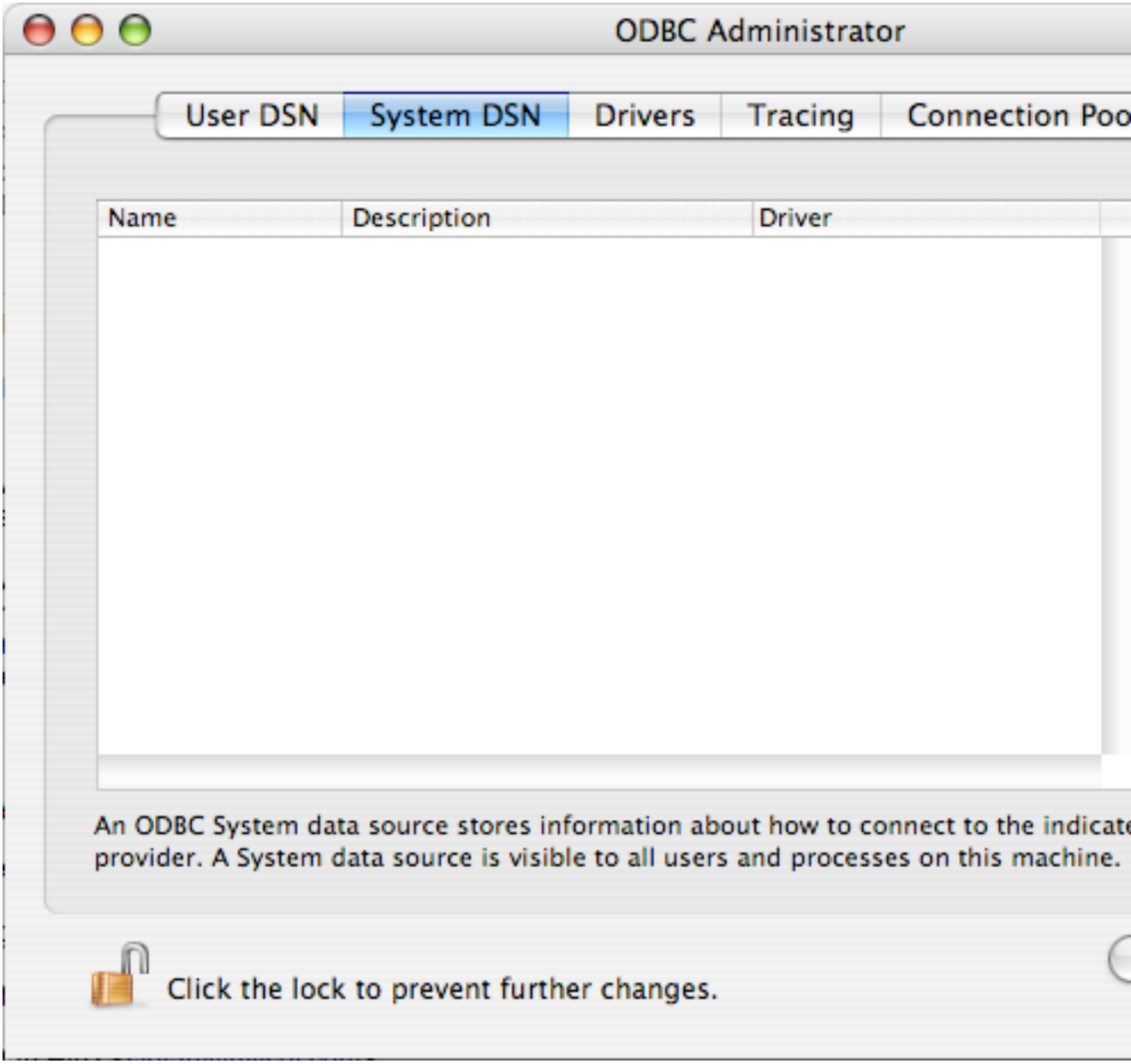
- On Windows, the default `myodbc3.dll` is compiled for optimal performance. If you want to debug Connector/ODBC 3.51 (for example, to enable tracing), you should instead use `myodbc3d.dll`. To install this file, copy `myodbc3d.dll` over the installed `myodbc3.dll` file. Make sure to revert back to the release version of the driver DLL once you are done with the debugging because the debug version may cause performance issues. Note that the `myodbc3d.dll` isn't included in Connector/ODBC 3.51.07 through 3.51.11. If you are using one of these versions, you should copy that DLL from a previous version (for example, 3.51.06).

For MyODBC 2.50, `myodbc.dll` and `myodbcd.dll` are used instead.

25.1.3.3. Configuring a Connector/ODBC DSN on Mac OS X

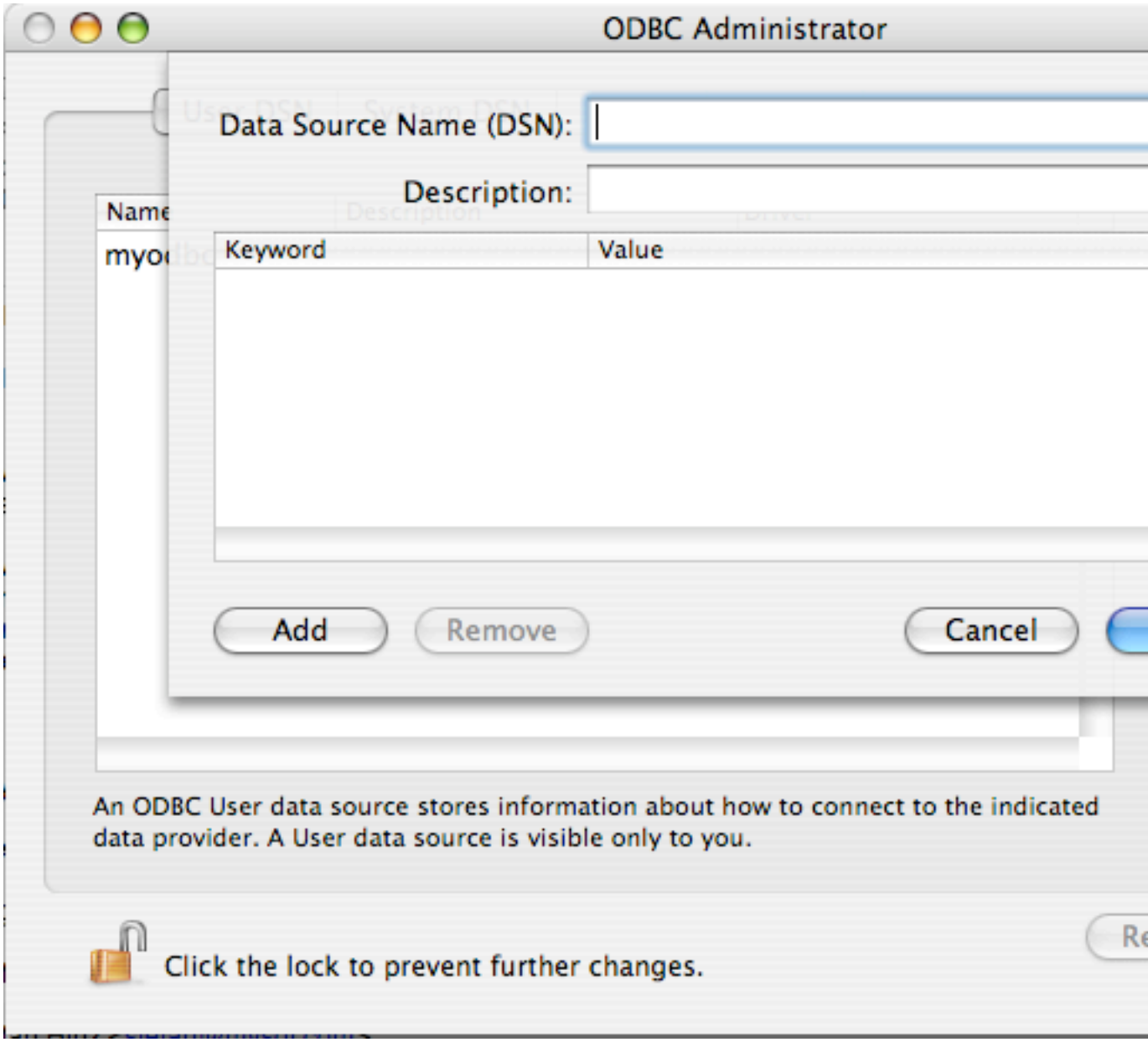
To configure a DSN on Mac OS X you should use the ODBC Administrator. If you have Mac OS X 10.2 or earlier, refer to Sección 25.1.3.4, “Configuring a Connector/ODBC DSN on Unix”. Select whether you want to create a User DSN or a System DSN. If you want to add a System DSN, you may need to authenticate with the system. You must click the padlock and enter a user and password with administrator privileges.

1. Open the ODBC Administrator from the `Utilities` folder in the `Applications` folder.



2. On the User DSN or System DSN panel, click Add.
3. Select the Connector/ODBC driver and click OK.

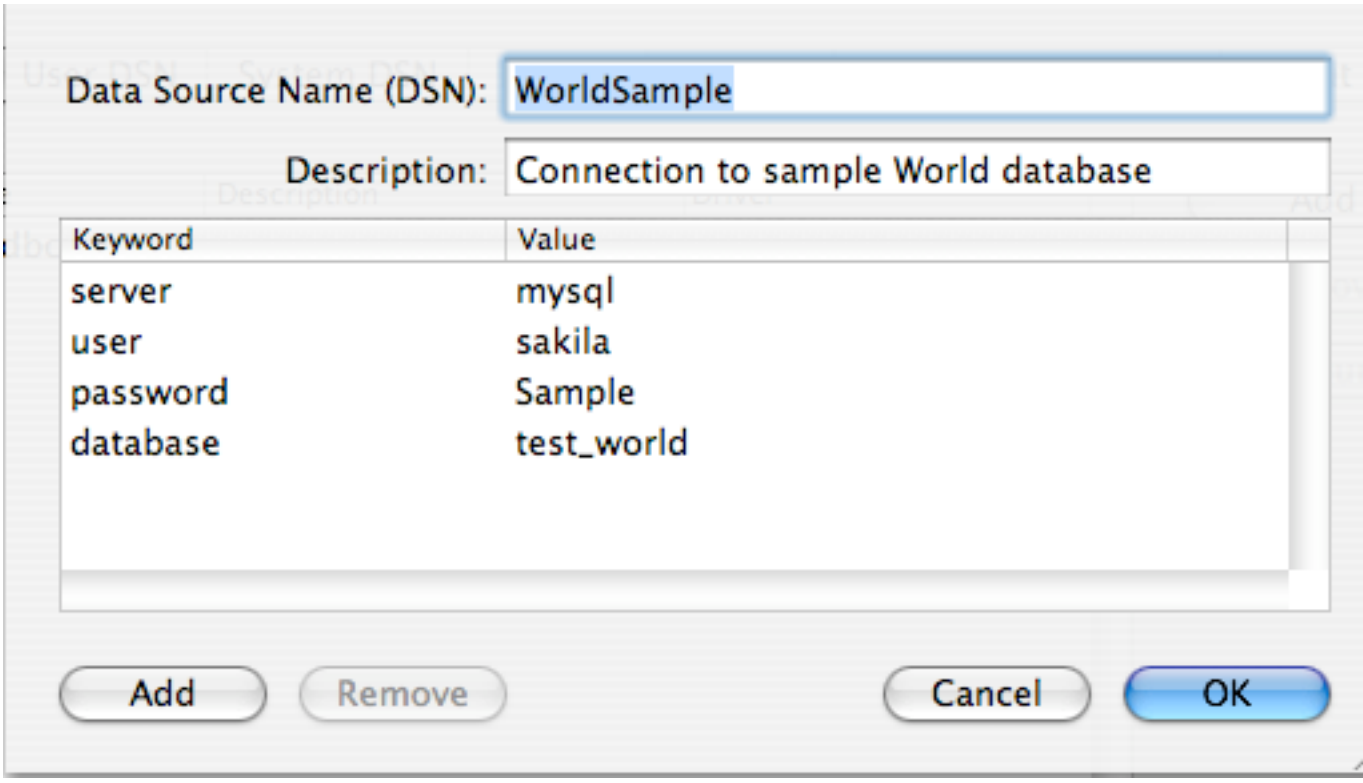
4. You will be presented with the `Data Source Name` dialog. Enter The `Data Source Name` and an optional `Description` for the DSN.



5. Click Add to add a new keyword/value pair to the panel. You should configure at least four pairs to specify the `server`, `username`, `password` and `database` connection parameters. See Sección 25.1.3.5, “Connector/ODBC Connection Parameters”.

6. Click OK to add the DSN to the list of configured data source names.

A completed DSN configuration may look like this:



You can configure additional ODBC options to your DSN by adding further keyword/value pairs and setting the corresponding values. See Sección 25.1.3.5, “Connector/ODBC Connection Parameters”.

25.1.3.4. Configuring a Connector/ODBC DSN on Unix

On `Unix`, you configure DSN entries directly in the `odbc.ini` file. Here is a typical `odbc.ini` file that configures `myodbc` and `myodbc3` as the DSN names for MyODBC 2.50 and Connector/ODBC 3.51, respectively:

```
;
;  odbc.ini configuration for Connector/ODBC and Connector/ODBC 3.51 drivers
;

[ODBC Data Sources]
myodbc      = MyODBC 2.50 Driver DSN
myodbc3     = MyODBC 3.51 Driver DSN

[myodbc]
Driver       = /usr/local/lib/libmyodbc.so
Description  = MyODBC 2.50 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =

[myodbc3]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = Connector/ODBC 3.51 Driver DSN
SERVER       = localhost
PORT         =
```

```
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =

[Default]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = Connector/ODBC 3.51 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =
```

Refer to the Sección 25.1.3.5, “Connector/ODBC Connection Parameters”, for the list of connection parameters that can be supplied.

Note: If you are using `unixODBC`, you can use the following tools to set up the DSN:

- ODBCConfig GUI tool(HOWTO: ODBCConfig)
- odbcinst

In some cases when using `unixODBC`, you might get this error:

```
Data source name not found and no default driver specified
```

If this happens, make sure the `ODBCINI` and `ODBCSYSINI` environment variables are pointing to the right `odbc.ini` file. For example, if your `odbc.ini` file is located in `/usr/local/etc`, set the environment variables like this:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

25.1.3.5. Connector/ODBC Connection Parameters

You can specify the parameters in the following tables for Connector/ODBC when configuring a DSN. Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see the table for information on which options relate to which fields and checkboxes. On Unix and Mac OS X, use the parameter name and value as the keyword/value pair in the DSN configuration. Alternatively, you can set these parameters within the `InConnectionString` argument in the `SQLDriverConnect()` call.

Parameter	Default Value	Comment
`user`	ODBC (on Windows)	The username used to connect to MySQL.
`server`	`localhost`	The hostname of the MySQL server.
`database`		The default database.
`option`	0	Options that specify how Connector/ODBC should work. See below.
`port`	3306	The TCP/IP port to use if `server` is not `localhost`.
`stmt`		A statement to execute when connecting to MySQL.

password		The password for the user account on server.
socket		The Unix socket file or Windows named pipe to connect to if server is localhost.

The option argument is used to tell Connector/ODBC that the client isn't 100% ODBC compliant. On Windows, you normally select options by toggling the checkboxes in the connection screen, but you can also select them in the option argument. The following options are listed in the order in which they appear in the Connector/ODBC connect screen:

Value	Windows Checkbox	Description
1	Don't Optimized Column Width	The client can't handle that Connector/ODBC returns the real width of a column.
2	Return Matching Rows	The client can't handle that MySQL returns the true value of affected rows. If this flag is set, MySQL returns “found rows” instead. You must have MySQL 3.21.14 or newer to get this to work.
4	Trace Driver Calls To myodbc.log	Make a debug log in C:\myodbc.log on Windows, or /tmp/myodbc.log on Unix variants.
8	Allow Big Results	Don't set any packet limit for results and parameters.
16	Don't Prompt Upon Connect	Don't prompt for questions even if driver would like to prompt.
32	Enable Dynamic Cursor	Enable or disable the dynamic cursor support. (Not allowed in Connector/ODBC 2.50.)
64	Ignore # in Table Name	Ignore use of database name in db_name.tbl_name.col_name.
128	User Manager Cursors	Force use of ODBC manager cursors (experimental).
256	Don't Use Set Locale	Disable the use of extended fetch (experimental).
512	Pad Char To Full Length	Pad CHAR columns to full column length.
1024	Return Table Names for SQLDescribeCol	SQLDescribeCol() returns fully qualified column names.
2048	Use Compressed Protocol	Use the compressed client/server protocol.
4096	Ignore Space After Function Names	Tell server to ignore space after function name and before ‘(’ (needed by PowerBuilder). This makes all function names keywords.
8192	Force Use of Named Pipes	Connect with named pipes to a mysqld server running on NT.
16384	Change BIGINT Columns to Int	Change BIGINT columns to INT columns (some applications can't handle BIGINT).
32768	No Catalog (exp)	Return 'user' as Table_qualifier and Table_owner from SQLTables (experimental).
65536	Read Options From my.cnf	Read parameters from the [client] and [odbc] groups from my.cnf.
131072	Safe	Add some extra safety checks (should not be needed but...).
262144	Disable transaction	Disable transactions.
524288	Save queries to myodbc.sql	Enable query logging to c:\myodbc.sql(/tmp/myodbc.sql) file. (Enabled only in debug mode.)
1048576	Don't Cache Result (forward only cursors)	Do not cache the results locally in the driver, instead read from server (mysql_use_result()). This works only for forward-

		only cursors. This option is very important in dealing with large tables when you don't want the driver to cache the entire result set.
2097152	Force Use Of Forward Only Cursors	Force the use of `Forward-only` cursor type. In case of applications setting the default static/dynamic cursor type, and one wants the driver to use non-cache result sets, then this option ensures the forward-only cursor behavior.
4194304	Enable auto-reconnect.	Enables auto-reconnection functionality. You should not use this option with transactions, since a auto reconnection during a incomplete transaction may cause corruption. Note that an auto-reconnected connection will not inherit the same settings and environment as the original. This option was enabled in Connector/ODBC 3.5.13.
8388608	Flag Auto Is Null	When set, this option causes the connection to set the `SQL_AUTO_IS_NULL` option to 1. This disables the standard behavior, but may enable older applications to correctly identify `AUTO_INCREMENT` values. For more information, see Capítulo 12, *Funciones y operadores*. This option was enabled in Connector/ODBC 3.5.13.

To select multiple options, add together their values. For example, setting `option` to 12 (4+8) gives you debugging without packet limits.

The following table shows some recommended `option` values for various configurations:

Configuration	Option Value
Microsoft Access, Visual Basic	3
Driver trace generation (Debug mode)	4
Microsoft Access (with improved DELETE queries)	35
Large tables with too many rows	2049
Sybase PowerBuilder	135168
Query log generation (Debug mode)	524288
Generate driver trace as well as query log (Debug mode)	524292
Large tables with no-cache results	3145731

25.1.3.6. Connecting Without a Predefined DSN

You can connect to the MySQL server using SQLDriverConnect, by specifying the `DRIVER` name field. Here are the connection strings for Connector/ODBC using DSN-Less connections:

For MyODBC 2.50:

```
ConnectionString = "DRIVER={MySQL};\
                    SERVER=localhost;\
                    DATABASE=test;\
                    USER=venu;\
                    PASSWORD=venu;\
                    OPTION=3;"
```

For Connector/ODBC 3.51:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
                   SERVER=localhost;\
                   DATABASE=test;\
                   USER=venu;\
                   PASSWORD=venu;\
                   OPTION=3;"
```

If your programming language converts backslash followed by whitespace to a space, it is preferable to specify the connection string as a single long string, or to use a concatenation of multiple strings that does not add spaces in between. For example:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"
                   "SERVER=localhost;"
                   "DATABASE=test;"
                   "USER=venu;"
                   "PASSWORD=venu;"
                   "OPTION=3;"
```

Note. Note that on Mac OS X you may need to specify the full path to the Connector/ODBC driver library.

Refer to the Sección 25.1.3.5, “Connector/ODBC Connection Parameters”, for the list of connection parameters that can be supplied.

25.1.3.7. ODBC Connection Pooling

Connection pooling enables the ODBC driver to re-use existing connections to a given database from a pool of connections, instead of opening a new connection each time the database is accessed. By enabling connection pooling you can improve the overall performance of your application by lowering the time taken to open a connection to a database in the connection pool.

For more information about connection pooling: http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470.

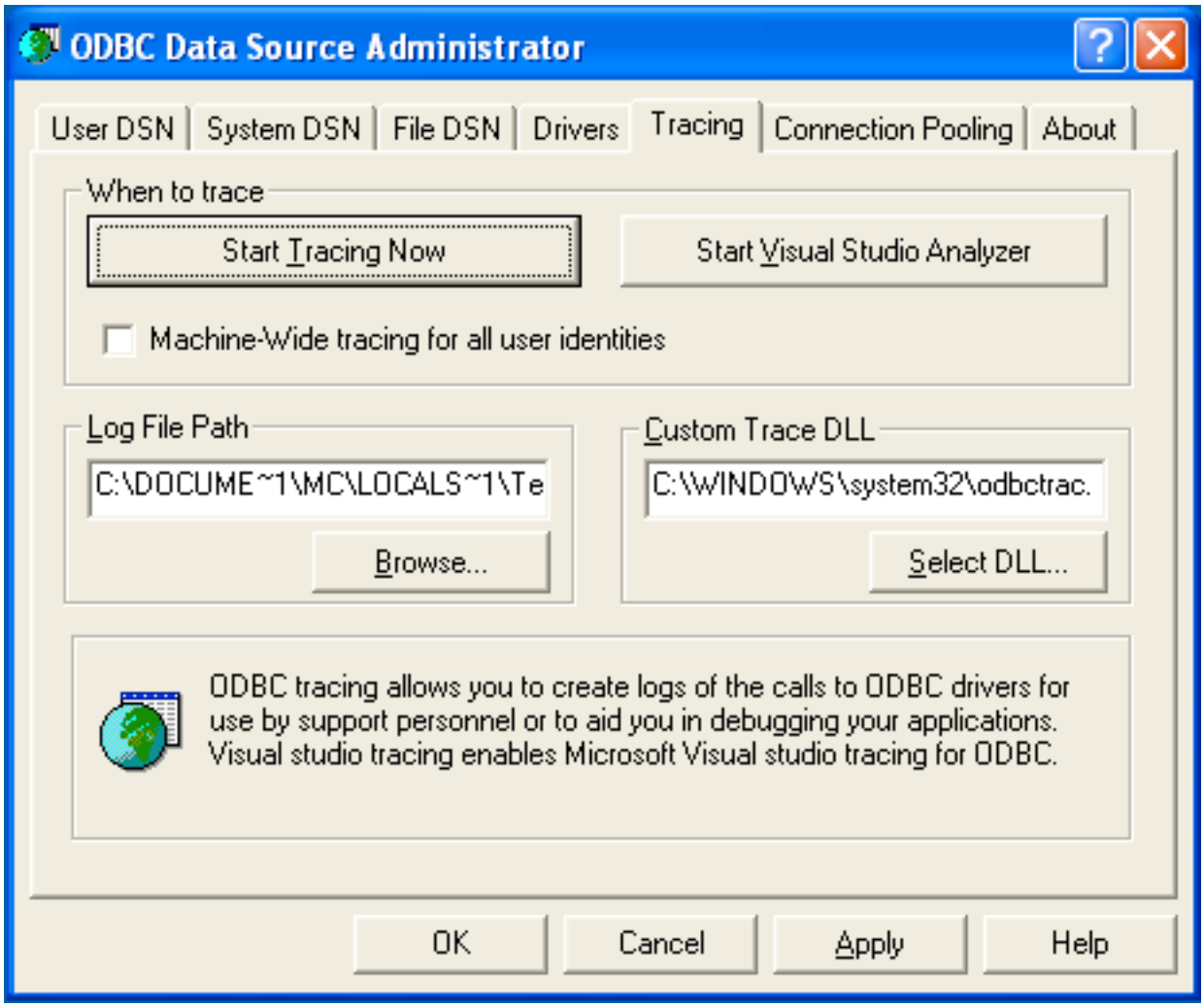
25.1.3.8. Getting an ODBC Trace File

If you encounter difficulties or problems with Connector/ODBC, you should start by making a log file from the `ODBC Manager` and Connector/ODBC. This is called *tracing*, and is enabled through the ODBC Manager. The procedure for this differs for Windows, Mac OS X and Unix.

Enabling ODBC Tracing on Windows

To enable the trace option on Windows:

1. The `Tracing` tab of the ODBC Data Source Administrator dialog box enables you to configure the way ODBC function calls are traced.



2. When you activate tracing from the `Tracing` tab, the `Driver Manager` logs all ODBC function calls for all subsequently run applications.

3. ODBC function calls from applications running before tracing is activated are not logged. ODBC function calls are recorded in a log file you specify.

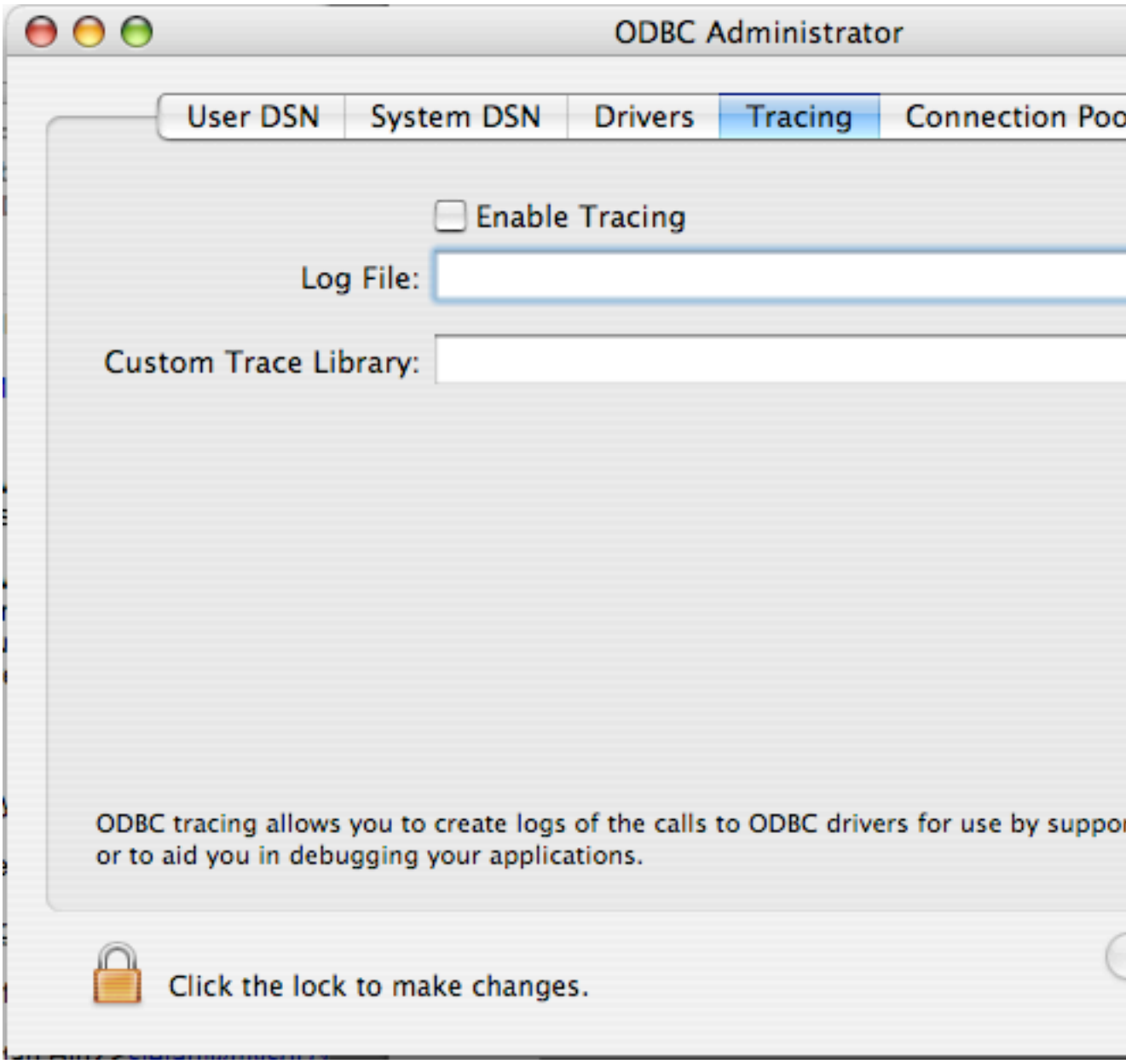
4. Tracing ceases only after you click `Stop Tracing Now`. Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.

Enabling ODBC Tracing on Mac OS X

To enable the trace option on Mac OS X 10.3 or later you should use the `Tracing` tab within ODBC Administrator .

1. Open the ODBC Administrator.

2. Select the `Tracing` tab.



3. Select the `Enable Tracing` checkbox.
4. Enter the location where you want to save the Tracing log. If you want to append information to an existing log file, click the Choose... button.

Enabling ODBC Tracing on Unix

To enable the trace option on Mac OS X 10.2 (or earlier) or Unix you must add the `trace` option to the ODBC configuration:

1. On Unix, you need to explicitly set the `Trace` option in the `ODBC.INI` file.

   Set the tracing `ON` or `OFF` by using `TraceFile` and `Trace` parameters in `odbc.ini` as shown below:

```
TraceFile  = /tmp/odbc.trace
Trace      = 1
```

`TraceFile` specifies the name and full path of the trace file and `Trace` is set to `ON` or `OFF`. You can also use `1` or `YES` for `ON` and `0` or `NO` for `OFF`. If you are using `ODBCConfig` from `unixODBC`, then follow the instructions for tracing `unixODBC` calls at HOWTO-ODBCConfig.

Enabling a Connector/ODBC Log

To generate a Connector/ODBC log, do the following:

1. Within Windows, enable the `Trace Connector/ODBC` option flag in the Connector/ODBC connect/configure screen. The log is written to file `C:\myodbc.log`. If the trace option is not remembered when you are going back to the above screen, it means that you are not using the `myodbcd.dll` driver, see "Errors and Debugging".

   On Mac OS X, Unix, or if you are using DSN-Less connection, then you need to supply `OPTION=4` in the connection string or set the corresponding keyword/value pair in the DSN.

2. Start your application and try to get it to fail. Then check the Connector/ODBC trace file to find out what could be wrong.

If you need help determining what is wrong, see Sección 25.1.7.1, "Connector/ODBC Community Support".

25.1.4. Connector/ODBC Examples

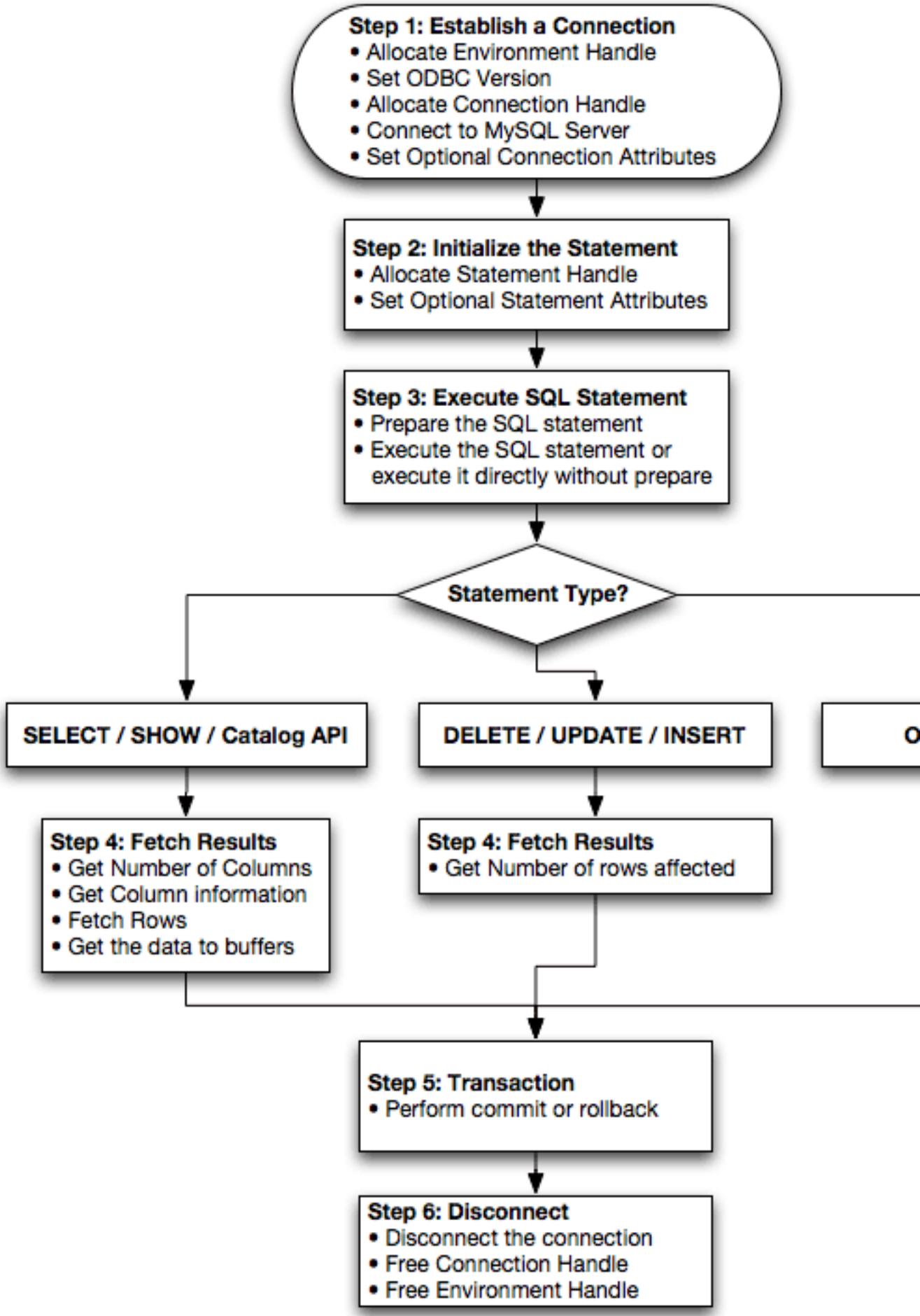
Once you have configured a DSN to provide access to a database, how you access and use that connection is dependent on the application or programming language. As ODBC is a standardized interface, any application or language that supports ODBC can use the DSN and connect to the configured database.

25.1.4.1. Basic Connector/ODBC Application Steps

Interacting with a MySQL server from an applications using the Connector/ODBC typically involves the following operations:

- Configure the Connector/ODBC DSN
- Connect to MySQL server
- Initialization operations
- Execute SQL statements
- Retrieve results
- Perform Transactions
- Disconnect from the server

Most applications use some variation of these steps. The basic application steps are shown in the following diagram:


Step 1: Establish a Connection
• Allocate Environment Handle
• Set ODBC Version
• Allocate Connection Handle
• Connect to MySQL Server
• Set Optional Connection Attributes
Step 2: Initialize the Statement
• Allocate Statement Handle
• Set Optional Statement Attributes
Step 3: Execute SQL Statement
• Prepare the SQL statement
• Execute the SQL statement or execute it directly without prepare
Statement Type?
SELECT / SHOW / Catalog API
DELETE / UPDATE / INSERT
Step 4: Fetch Results
• Get Number of Columns
• Get Column information
• Fetch Rows
• Get the data to buffers
Step 4: Fetch Results
• Get Number of rows affected
Step 5: Transaction
• Perform commit or rollback
Step 6: Disconnect
• Disconnect the connection
• Free Connection Handle
• Free Environment Handle

25.1.4.2. Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC

A typical installation situation where you would install Connector/ODBC is when you want to access a database on a Linux or Unix host from a Windows machine.

As an example of the process required to set up access between two machines, the steps below take you through the basic steps. These instructions assume that you want to connect to system ALPHA from system BETA with a username and password of `myuser` and `mypassword`.

On system ALPHA (the MySQL server) follow these steps:

1. Start the MySQL server.
2. Use `GRANT` to set up an account with a username of `myuser` that can connect from system BETA using a password of `myuser` to the database `test`:

   ```
   GRANT ALL ON test.* to 'myuser'@'BETA' IDENTIFIED BY 'mypassword';
   ```

   For more information about MySQL privileges, refer to Sección 5.7, “Gestión de la cuenta de usuario MySQL”.

On system BETA (the Connector/ODBC client), follow these steps:

1. Configure a Connector/ODBC DSN using parameters that match the server, database and authentication information that you have just configured on system ALPHA.

Parameter	Value	Comment
DSN	remote_test	A name to identify the connection.
SERVER	ALPHA	The address of the remote server.
DATABASE	test	The name of the default database.
USER	myuser	The username configured for access to this database.
PASSWORD	mypassword	The password for `myuser`.

2. Using an ODBC-capable application, such as Microsoft Office, connect to the MySQL server using the DSN you have just created. If the connection fails, use tracing to examine the connection process. See Sección 25.1.3.8, “Getting an ODBC Trace File”, for more information.

25.1.4.3. Connector/ODBC and Third-Party ODBC Tools

Once you have configured your Connector/ODBC DSN, you can access your MySQL database through any application that supports the ODBC interface, including programming languages and third-party applications. This section contains guides and help on using Connector/ODBC with various ODBC-compatible tools and applications, including Microsoft Word, Microsoft Excel and Adobe/Macromedia ColdFusion.

Applications Tested with Connector/ODBC

Connector/ODBC has been tested with the following applications:

Publisher	Application	Notes
Adobe	ColdFusion	Formerly Macromedia ColdFusion
Borland	C++ Builder	
	Builder 4	

	Delphi	
Business Objects	Crystal Reports	
Claris	Filemaker Pro	
Corel	Paradox	
Computer Associates	Visual Objects	Also known as CAVO
	AllFusion ERwin Data Modeler	
Gupta	Team Developer	Previously known as Centura Team Developer; Gupta SQL/Windows
Gensym	G2-ODBC Bridge	
Inline	iHTML	
Lotus	Notes	Versions 4.5 and 4.6
Microsoft	Access	
	Excel	
	Visio Enterprise	
	Visual C++	
	Visual Basic	
	ODBC.NET	Using C#, Visual Basic, C++
	FoxPro	
	Visual Interdev	
OpenOffice.org	OpenOffice.org	
Perl	DBD::ODBC	
Pervasive Software	DataJunction	
Sambar Technologies	Sambar Server	
SPSS	SPSS	
SoftVelocity	Clarion	
SQLExpress	SQLExpress for Xbase++	
Sun	StarOffice	
SunSystems	Vision	
Sybase	PowerBuilder	
	PowerDesigner	
theKompany.com	Data Architect	

If you know of any other applications that work with Connector/ODBC, please send mail to <myodbc@lists.mysql.com> about them.

Using Connector/ODBC with Microsoft Word or Excel

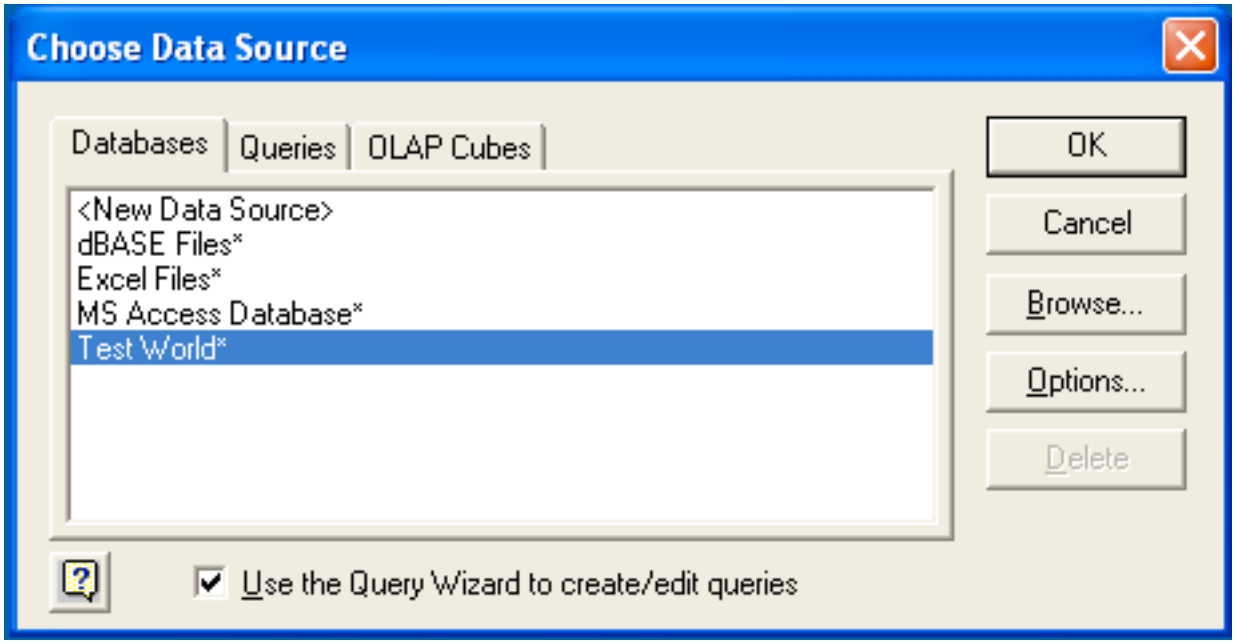
You can use Microsoft Word and Microsoft Excel to access information from a MySQL database using Connector/ODBC. Within Microsoft Word, this facility is most useful when importing data for mailmerge, or for tables and data to be included in reports. Within Microsoft Excel, you can execute queries on your MySQL server and import the data directly into an Excel Worksheet, presenting the data as a series of rows and columns.

With both applications, data is accessed and imported into the application using Microsoft Query , which enables you to execute a query though an ODBC source. You use Microsoft Query to build the SQL statement to be executed, selecting the tables, fields, selection criteria and sort order. For example, to insert information from a table in the World test database into an Excel spreadsheet, using the DSN samples shown in Sección 25.1.3, “Connector/ODBC Configuration”:

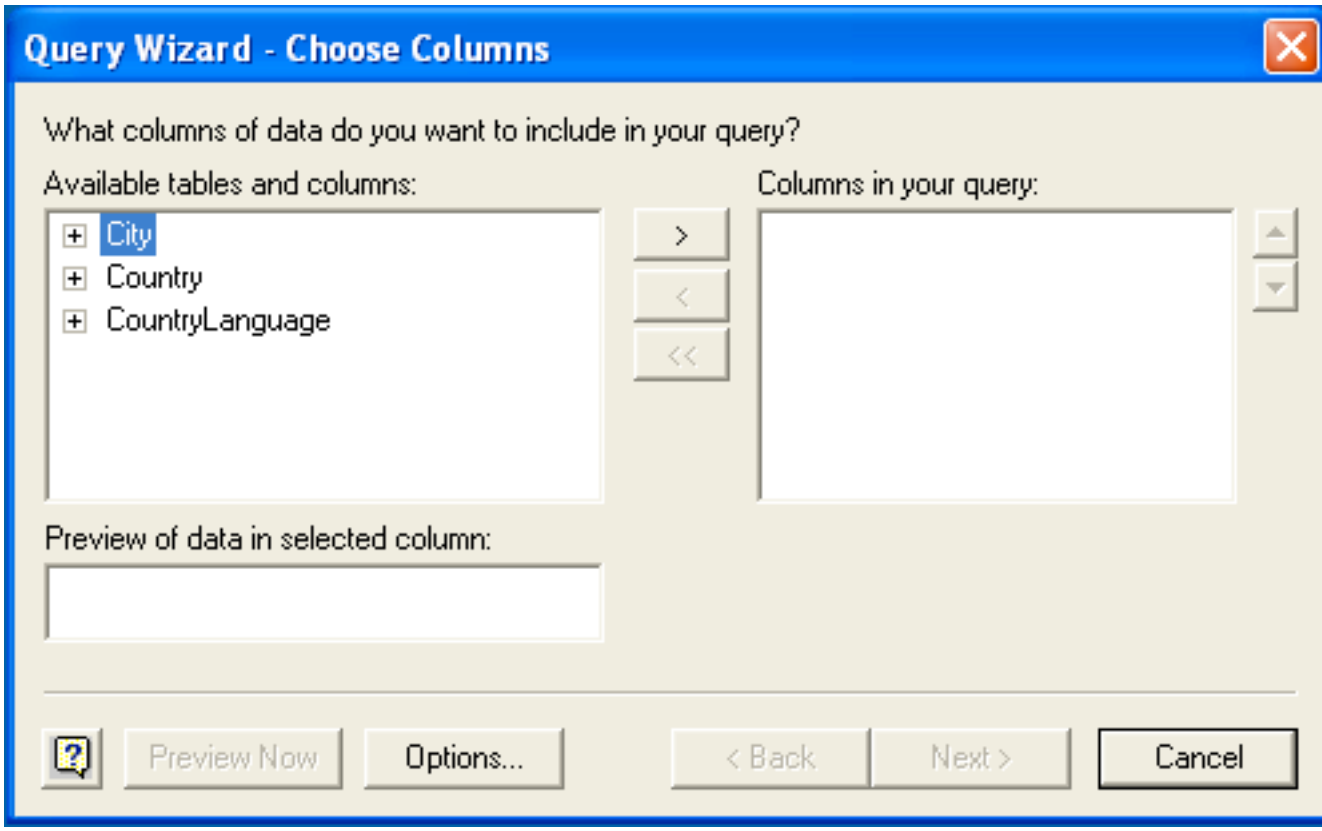
1. Create a new Worksheet.

2. From the `Data` menu, choose `Import External Data`, and then select `New Database Query`.

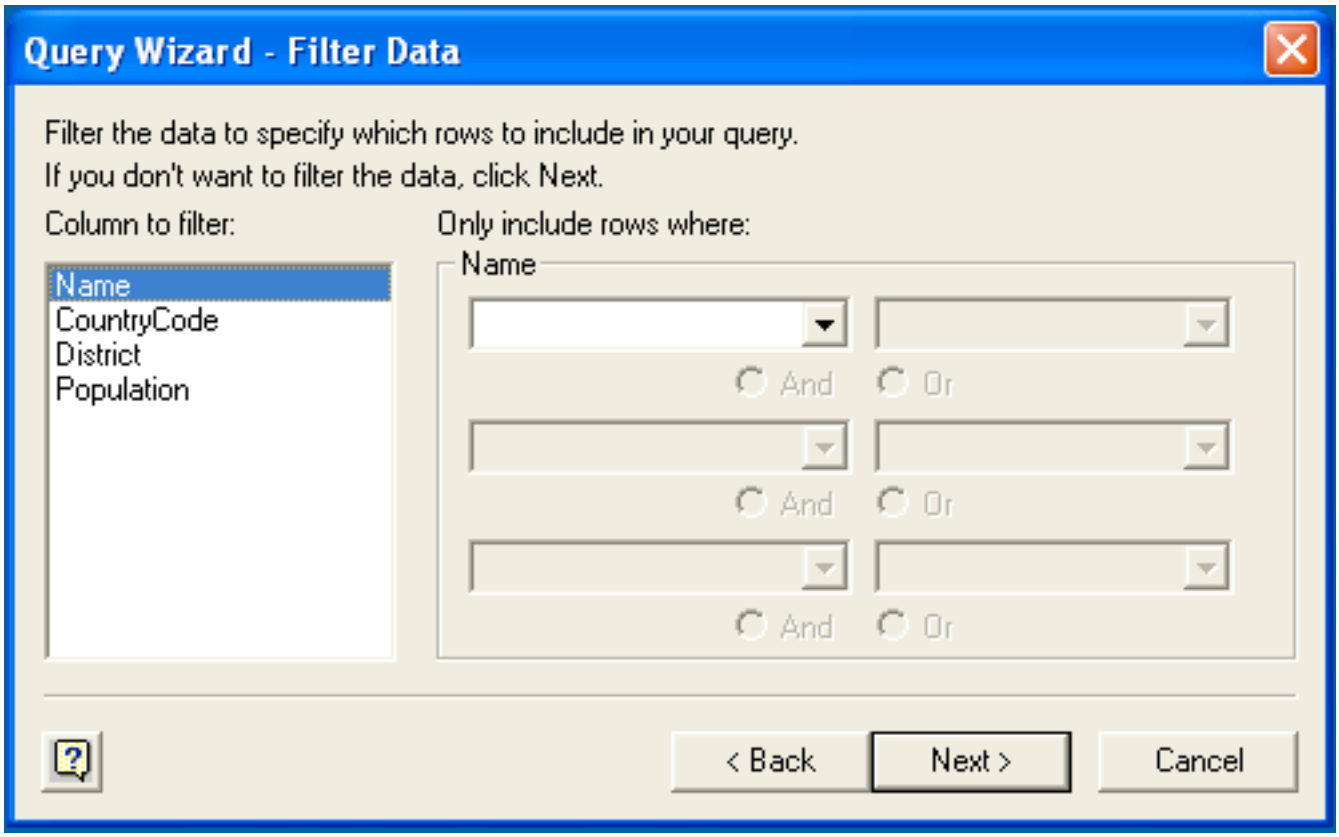
3. Microsoft Query will start. First, you need to choose the data source, by selecting an existing Data Source Name.



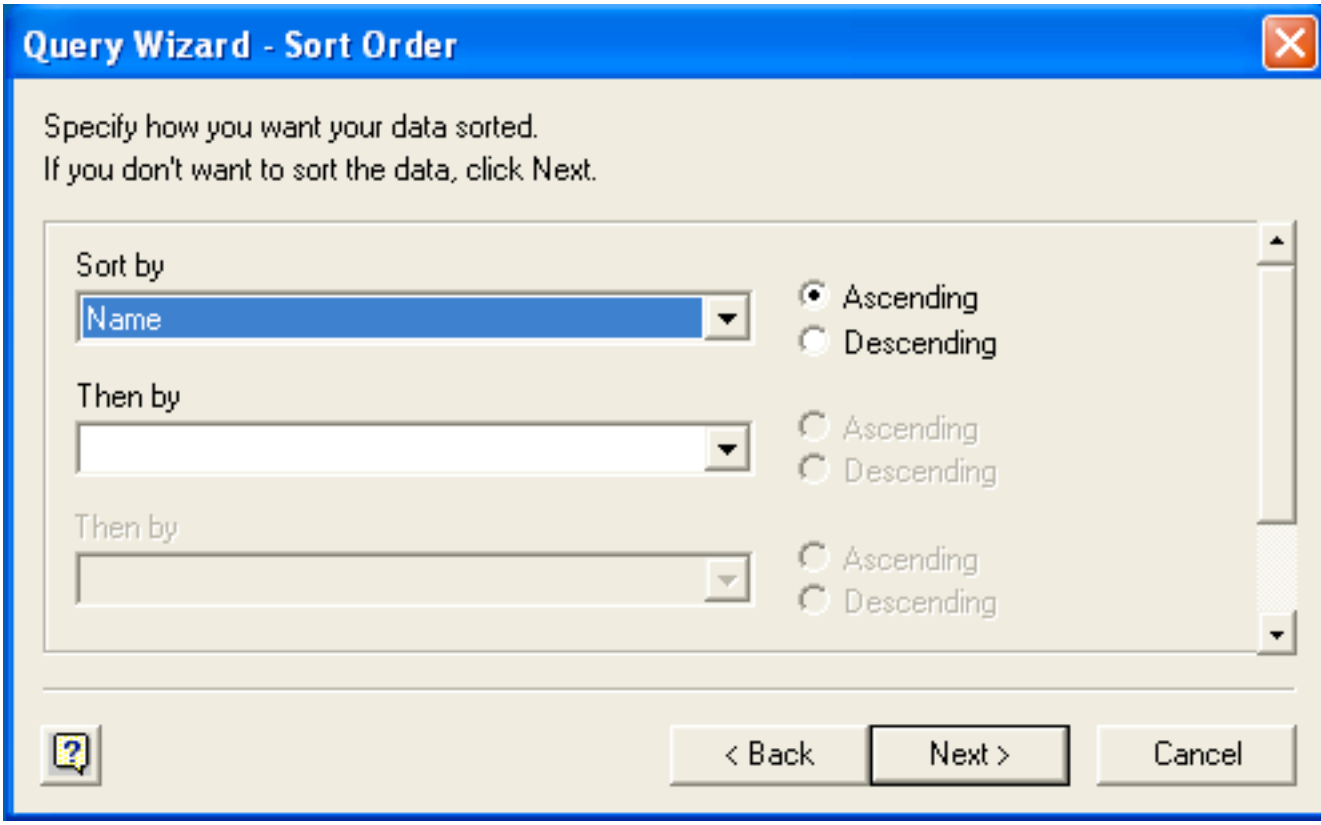
4. Within the `Query Wizard`, you must choose the columns that you want to import. The list of tables available to the user configured through the DSN is shown on the left, the columns that will be added to your query are shown on the right. The columns you choose are equivalent to those in the first section of a `SELECT` query. Click Next to continue.



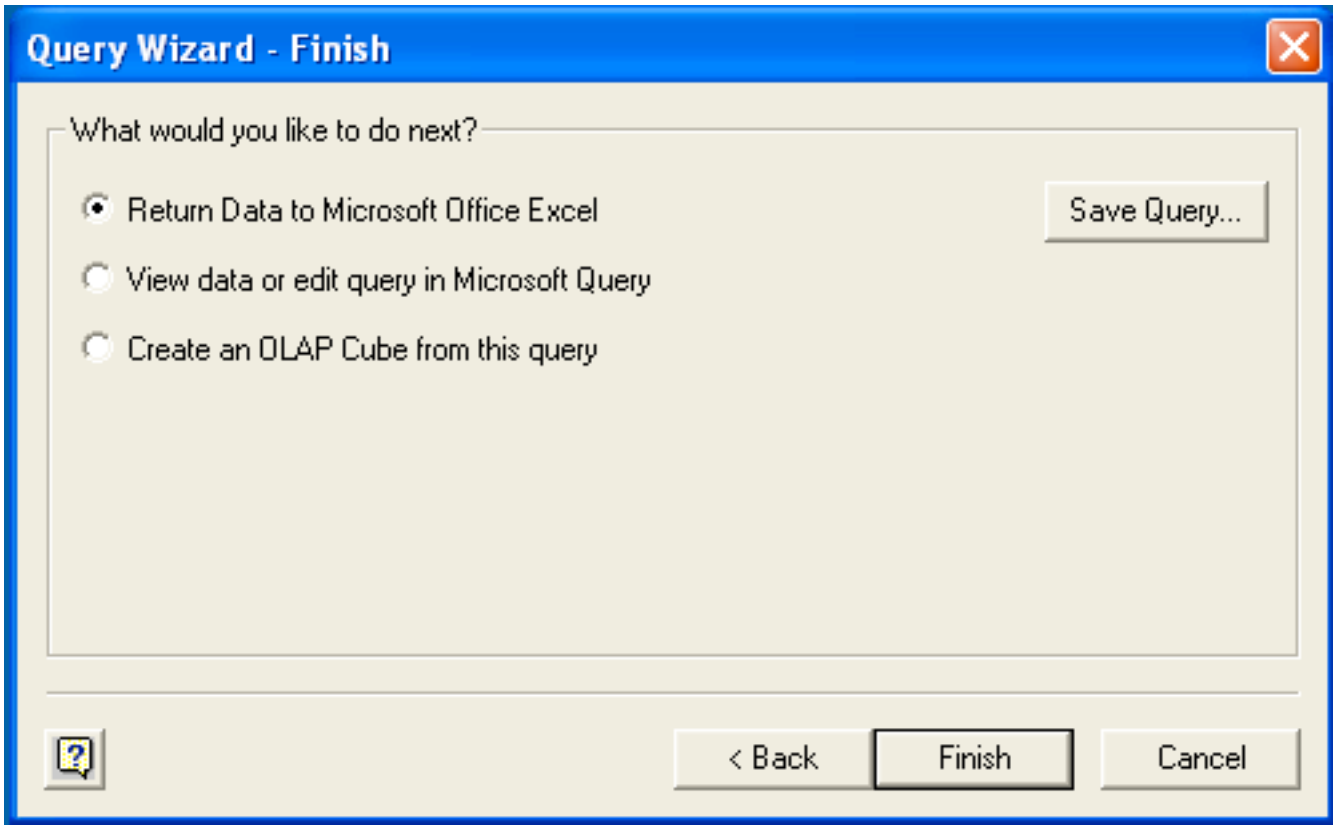
5. You can filter rows from the query (the equivalent of a `WHERE` clause) using the `Filter Data` dialog. Click Next to continue.



6. Select an (optional) sort order for the data. This is equivalent to using a `ORDER BY` clause in your SQL query. You can select up to three fields for sorting the information returned by the query. Click Next to continue.



7. Select the destination for your query. You can select to return the data Microsoft Excel, where you can choose a worksheet and cell where the data will be inserted; you can continue to view the query and results within Microsoft Query, where you can edit the SQL query and further filter and sort the information returned; or you can create an OLAP Cube from the query, which can then be used directly within Microsoft Excel. Click Finish.



The same process can be used to import data into a Word document, where the data will be inserted as a table. This can be used for mail merge purposes (where the field data is read from a Word table), or where you want to include data and reports within a report or other document.

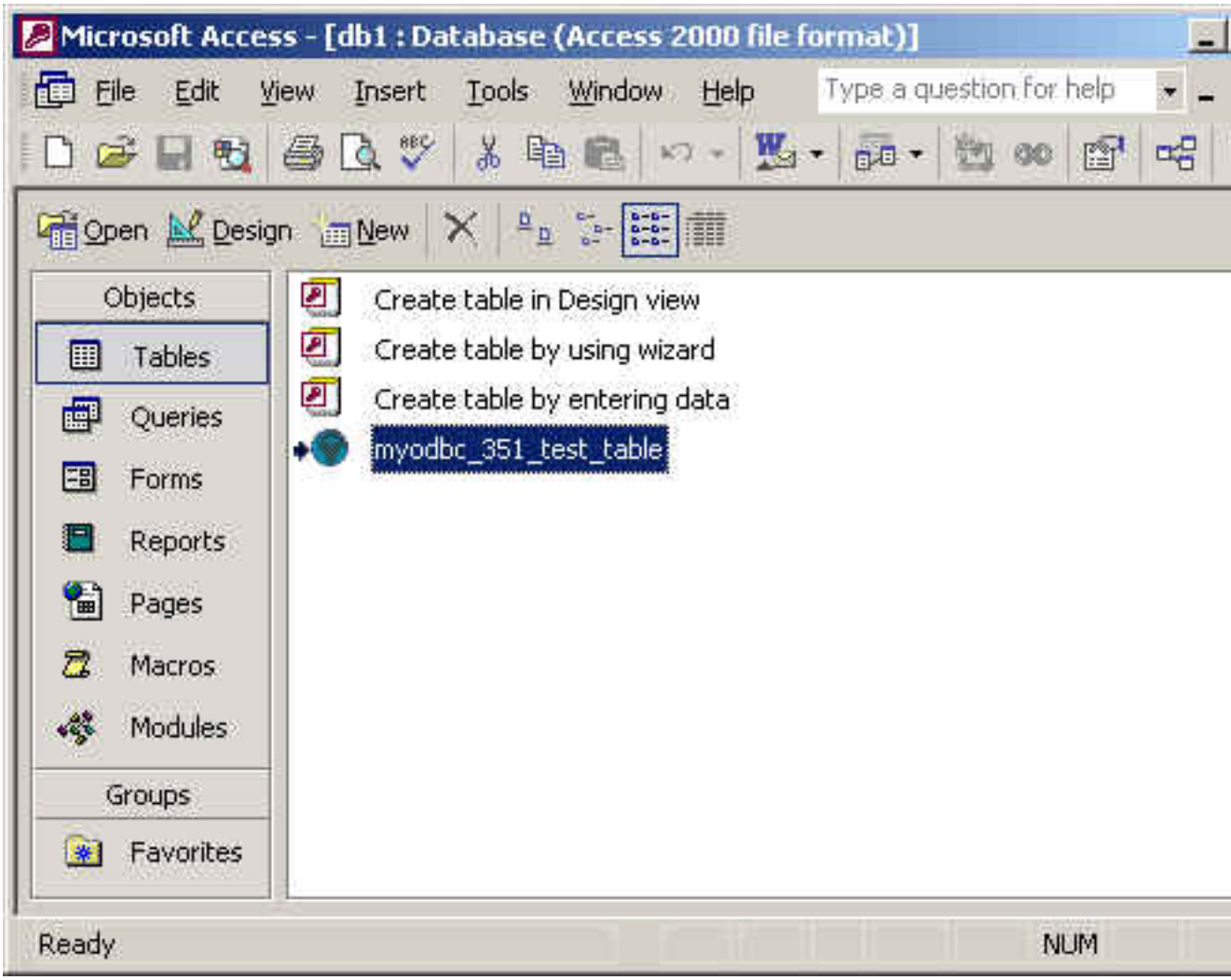
Using Connector/ODBC and Microsoft Access

You can use MySQL database with Microsoft Access using Connector/ODBC. The MySQL database can be used as an import source, an export source, or as a linked table for direct use within an Access application, so you can use Access as the front-end interface to a MySQL database.

Exporting Access Data to MySQL

To export a table of data from an Access database to MySQL, follow these instructions:

1. When you open an Access database or an Access project, a Database window appears. It displays shortcuts for creating new database objects and opening existing objects.



2. Click the name of the `table` or `query` you want to export, and then in the `File` menu, select `Export`.

3. In the `Export Object Type` *`Object name`* `To` dialog box, in the `Save As Type` box, select `ODBC Databases ()` as shown here:

4. In the `Export` dialog box, enter a name for the file (or use the suggested name), and then select `OK`.

5. The Select Data Source dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the File Data Source or Machine Data Source tab, and then double-click the Connector/ODBC or Connector/ODBC 3.51 data source that you want to export to. To

define a new data source for Connector/ODBC, please Sección 25.1.3.2, “Configuring a Connector/ODBC DSN on Windows”.

Microsoft Access connects to the MySQL Server through this data source and exports new tables and or data.

Importing MySQL Data to Access

To import or link a table or tables from MySQL to Access, follow these instructions:

1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the `File` menu, point to `Get External Data`, and then click `Import`. To link tables, on the File menu, point to `Get External Data`, and then click `Link Tables`.
3. In the `Import` (or `Link`) dialog box, in the Files Of Type box, select `ODBC Databases ()`. The Select Data Source dialog box lists the defined data sources The Select Data Source dialog box is displayed; it lists the defined data source names.
4. If the ODBC data source that you selected requires you to log on, enter your login ID and password (additional information might also be required), and then click `OK`.
5. Microsoft Access connects to the MySQL server through `ODBC data source` and displays the list of tables that you can `import` or `link`.
6. Click each table that you want to `import` or `link`, and then click `OK`. If you're linking a table and it doesn't have an index that uniquely identifies each record, Microsoft Access displays a list of the fields in the linked table. Click a field or a combination of fields that uniquely identifies each record, and then click `OK`.

Linking MySQL Data to Access Tables

Use the following procedure to view or to refresh links when the structure or location of a linked table has changed. The Linked Table Manager lists the paths to all currently linked tables.

To view or refresh links:

1. Open the database that contains links to tables.
2. On the `Tools` menu, point to `Add-ins` (`Database Utilities` in Access 2000 or newer), and then click `Linked Table Manager`.
3. Select the check box for the tables whose links you want to refresh.
4. Click OK to refresh the links.

Microsoft Access confirms a successful refresh or, if the table wasn't found, displays the `Select New Location of` <table name> dialog box in which you can specify its the table's new location. If several selected tables have moved to the new location that you specify, the Linked Table Manager searches that location for all selected tables, and updates all links in one step.

To change the path for a set of linked tables:

1. Open the database that contains links to tables.
2. On the `Tools` menu, point to `Add-ins` (`Database Utilities` in Access 2000 or newer), and then click `Linked Table Manager`.

3. Select the `Always Prompt For A New Location` check box.
4. Select the check box for the tables whose links you want to change, and then click `OK`.
5. In the `Select New Location of` <table name> dialog box, specify the new location, click `Open`, and then click `OK`.

25.1.4.4. Using Connector/ODBC with Crystal Reports

Crystal Reports can use an ODBC DSN to connect to a database from which you to extract data and information for reporting purposes.

> **Nota**
>
> There is a known issue with certain versions of Crystal Reports where the application is unable to open and browse tables and fields through an ODBC connection. Before using Crystal Reports with MySQL, please ensure that you have update to the latest version, including any outstanding service packs and hotfixes. For more information on this issue, see the Business) Objects Knowledgebase for more information.

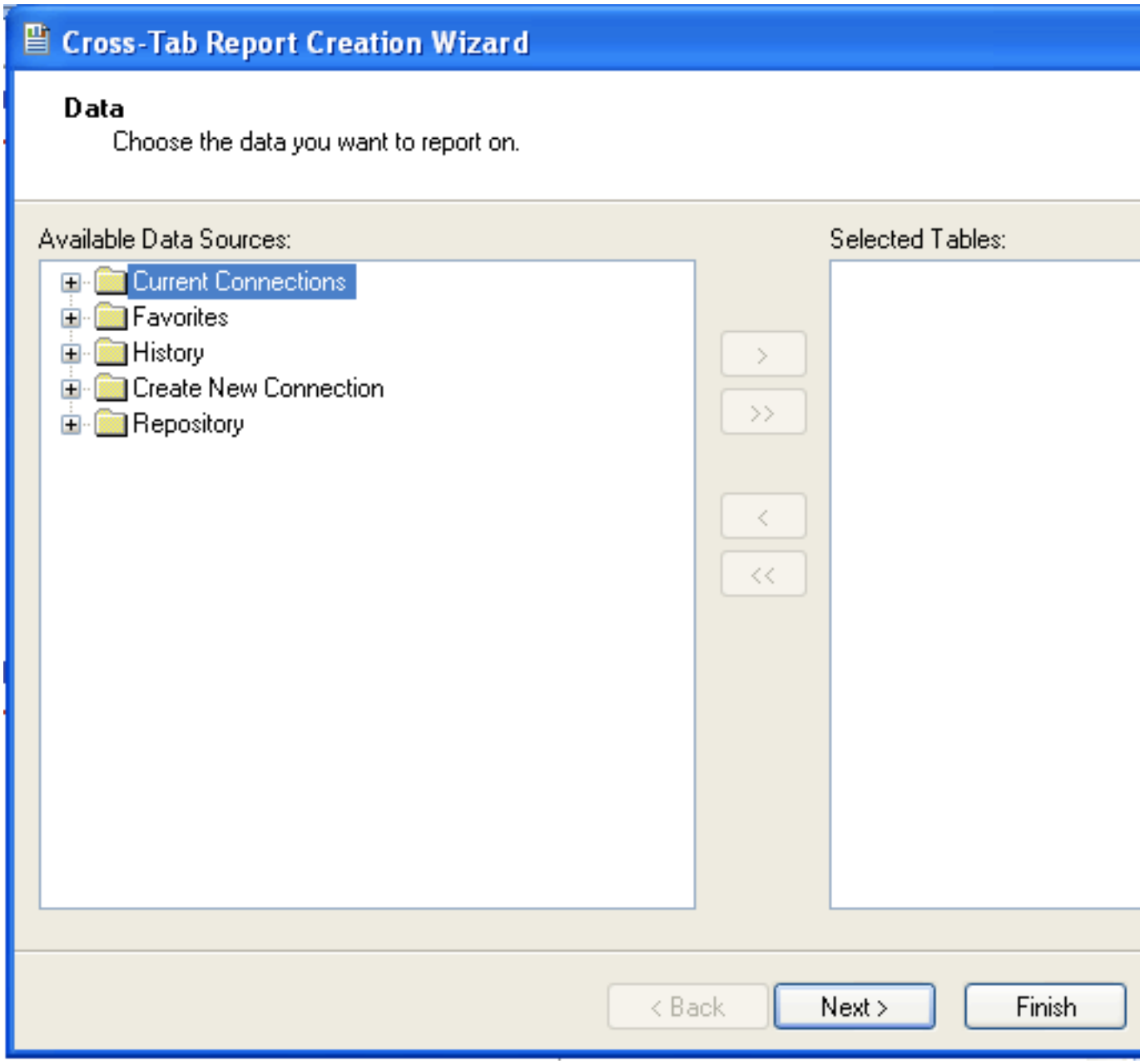
For example, to create a simple crosstab report within Crystal Reports XI, you should follow these steps:

1. Create a DSN using the `Data Sources (ODBC)` tool. You can either specify a complete database, including username and password, or you can build a basic DSN and use Crystal Reports to set the username and password.

   For the purposes of this example, a DSN that provides a connection to an instance of the MySQL Sakila sample database has been created.

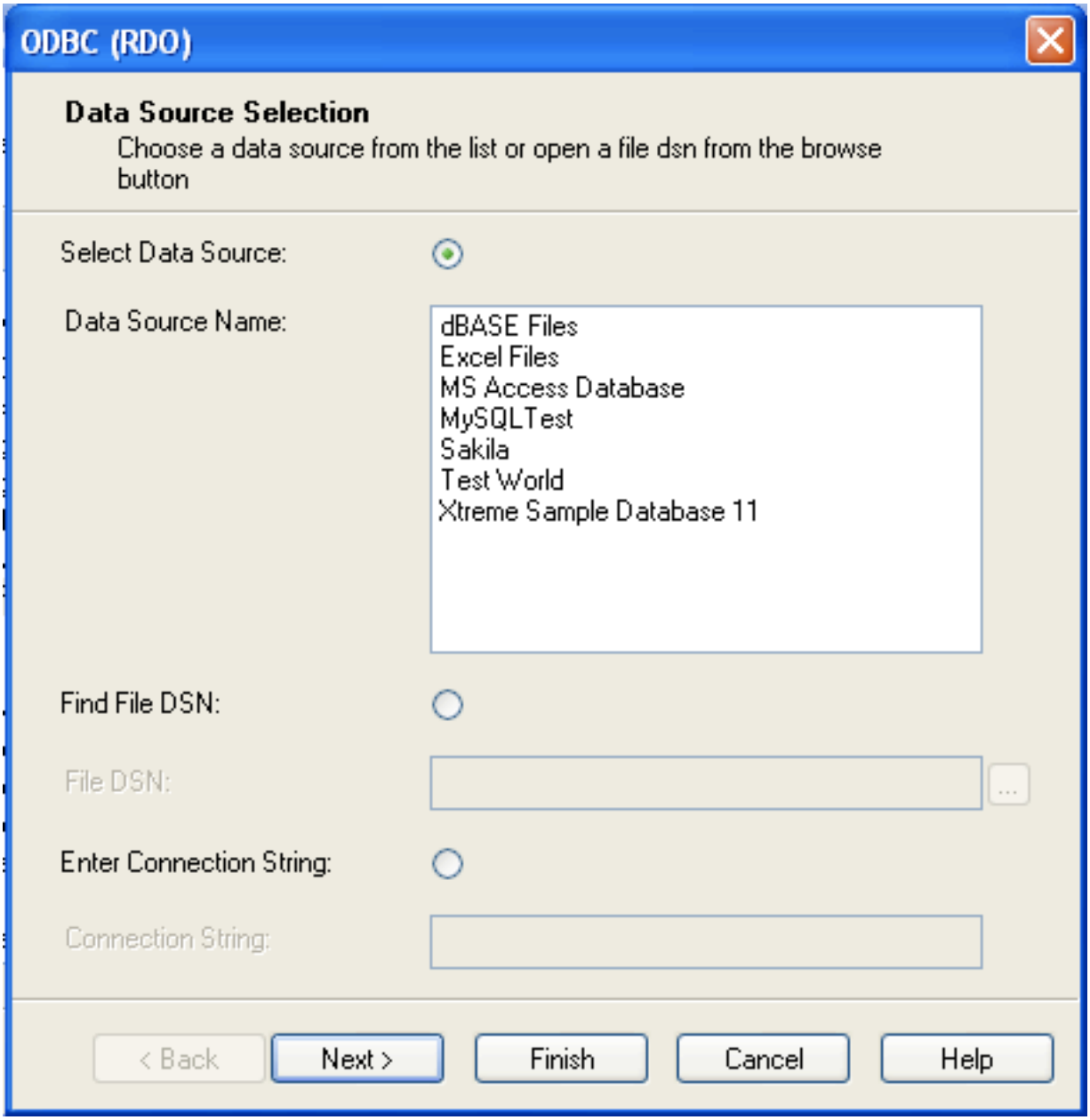
2. Open Crystal Reports and create a new project, or an open an existing reporting project into which you want to insert data from your MySQL data source.
3. Start the Cross-Tab Report Wizard, either by clicking on the option on the Start Page. Expand the **Create New Connection** folder, then expand the **ODBC (RDO)** folder to obtain a list of ODBC data sources.

   You will be asked to select a data source.



4. When you first expand the **ODBC (RDO)** folder you will be presented the Data Source Selection screen. From here you can select either a pre-configured DSN, open a file-based DSN or enter and manual connection string. For this example, the **Sakila** DSN will be used.

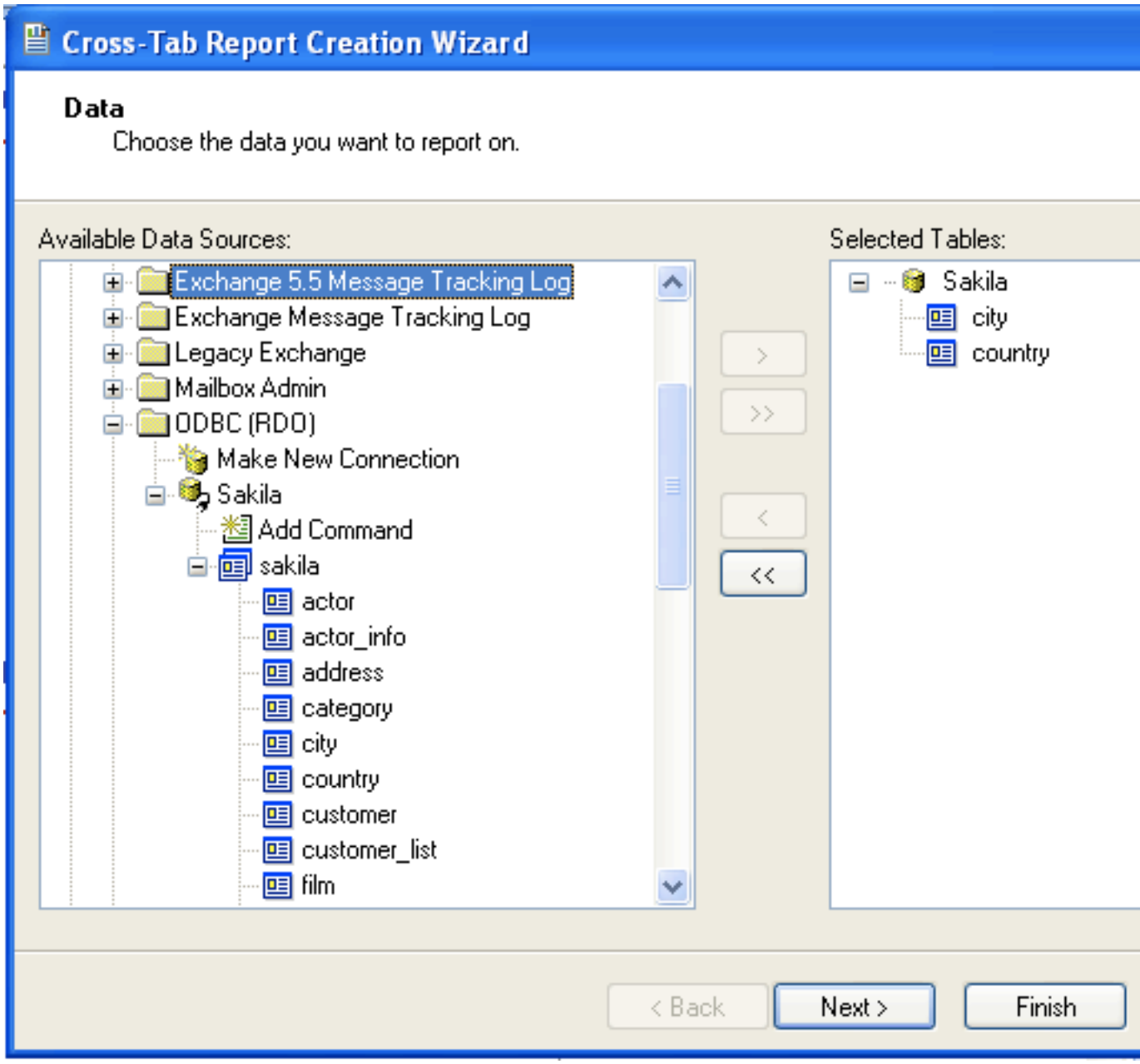
   If the DSN contains a username/password combination, or you want to use different authentication credentials, click Next to enter the username and password that you want to use. Otherwise, click Finish to continue the data source selection wizard.



5. You will be returned the Cross-Tab Report Creation Wizard. You now need to select the database and tables that you want to include in your report. For our example, we will expand the selected Sakila database. Click the `city` table and use the > button to add the table to the report. Then repeat the action with the `country` table. Alternatively you can select multiple tables and add them to the report.

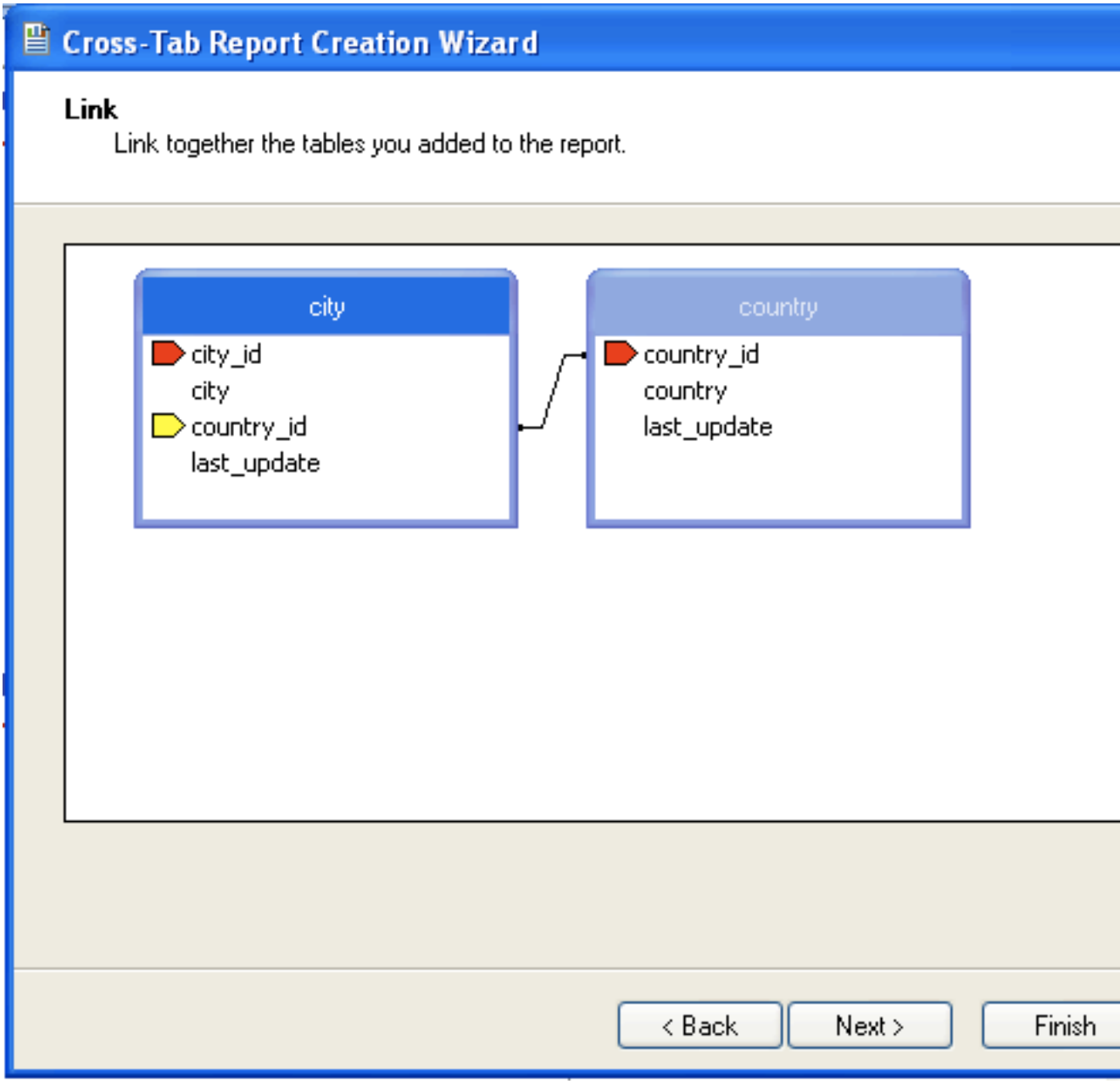
   Finally, you can select the parent **Sakila** resource and add of the tables to the report.

   Once you have selected the tables you want to include, click Next to continue.



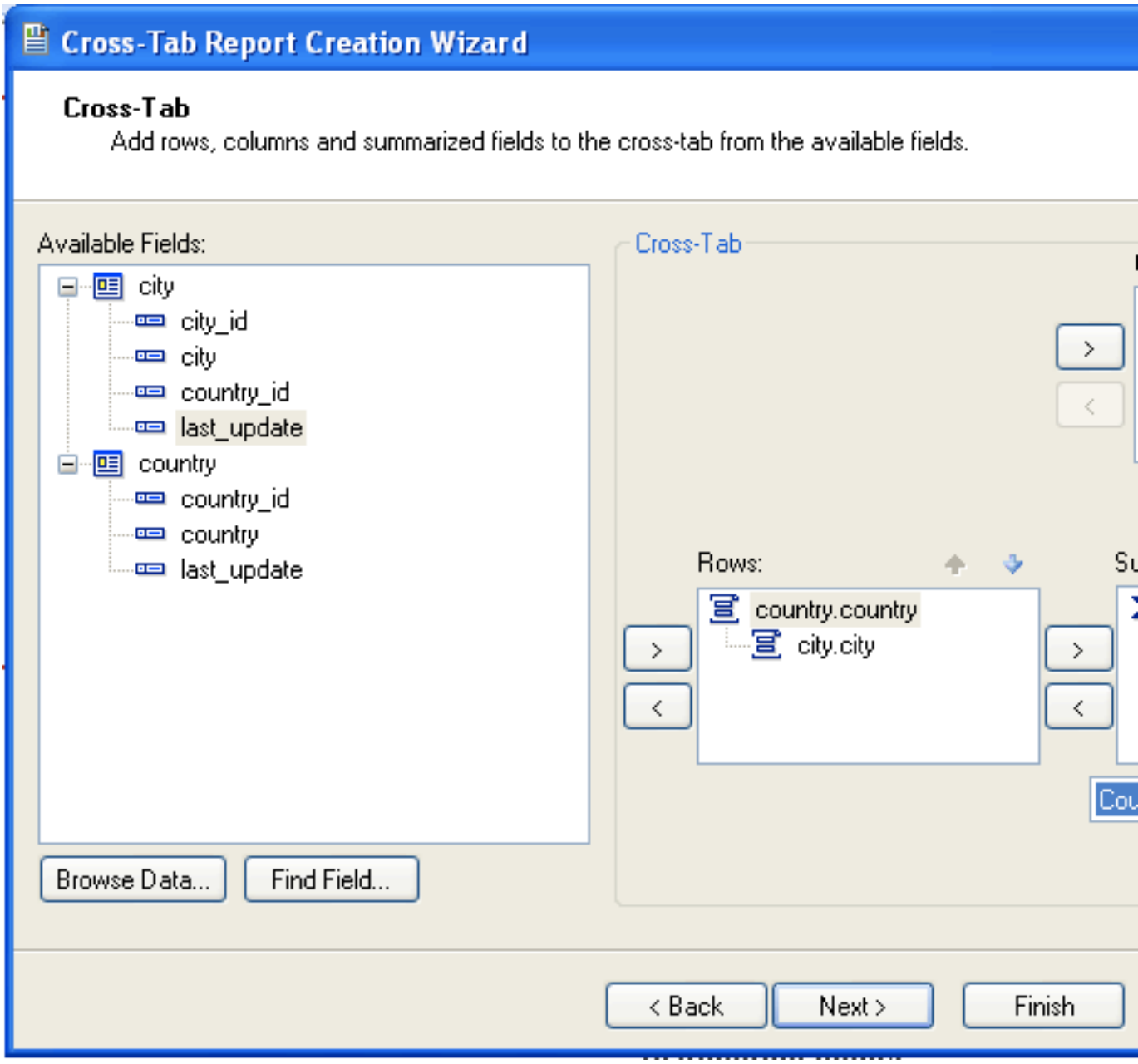
6. Crystal Reports will now read the table definitions and automatically identify the links between the tables. The identification of links between tables enables Crystal Reports to automatically lookup and summarize information based on all the tables in the database according to your query. If Crystal Reports is unable to perform the linking itself, you can manually create the links between fields in the tables you have selected.

   Click Next to continue the process.



7. You can now select the columns and rows that you wish to include within the Cross-Tab report. Drag and drop or use the > buttons to add fields to each area of the report. In the example shown, we will report on cities, organized by country, incorporating a count of the number of cities within each country. If you want to browse the data, select a field and click the Browse Data... button.

   Click Next to create a graph of the results. Since we are not creating a graph from this data, click Finish to generate the report.



8. The finished report will be shown, a sample of the output from the Sakila sample database is shown below.

		Total
Total		600
Afghanistan	Total	1
	Kabul	1
Algeria	Total	3
	Batna	1
	Bchar	1
	Skikda	1
American Samoa	Total	1
	Tafuna	1
Angola	Total	2
	Benguela	1
	Namibe	1
Anguilla	Total	1
	South Hill	1
Argentina	Total	13
	Almirante Brow	1

Once the ODBC connection has been opened within Crystal Reports, you can browse and add any fields within the available tables into your reports.

25.1.4.5. Connector/ODBC Programming Examples

With a suitable ODBC Manager and the my Connector/ODBC driver installed, any programming language or environment that can support ODBC should be able to connect to a MySQL database through Connector/ODBC.

This includes, but is certainly not limited to, Microsoft support languages (including Visual Basic, C# and interfaces such as ODBC.NET), Perl (through the DBI module, and the DBD::ODBC driver).

Using Connector/ODBC with Visual Basic Using ADO, DAO and RDO

This section contains simple examples of the use of MySQL ODBC 3.51 Driver with ADO, DAO and RDO.

ADO: `rs.addNew`, `rs.delete`, and `rs.update`

The following ADO (ActiveX Data Objects) example creates a table `my_ado` and demonstrates the use of `rs.addNew`, `rs.delete`, and `rs.update`.

```
Private Sub myodbc_ado_Click()

Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fld As ADODB.Field
Dim sql As String

'connect to MySQL server using MySQL ODBC 3.51 Driver
Set conn = New ADODB.Connection
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"

conn.Open

'create table
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20)," _
& "txt text, dt date, tm time, ts timestamp)"

'direct insert
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"

Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer

'fetch the initial table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close

'rs insert
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close

'rs update
```

```
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close

'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close

'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
conn.Close
End Sub
```

DAO: `rs.addNew`, `rs.update`, and Scrolling

The following DAO (Data Access Objects) example creates a table `my_dao` and demonstrates the use of `rs.addNew`, `rs.update`, and result set scrolling.

```
Private Sub myodbc_dao_Click()

Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String

'connect to MySQL using MySQL ODBC 3.51 Driver
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

'Create table my_dao
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute

Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMENT PRIMARY KEY, " _
& "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")
```

```
queryDef.Execute

'Insert new records using rs.addNew
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer

For i = 10 To 15
rs.AddNew
rs!Name = "insert record" & i
rs!Id2 = i
rs.Update
Next i
rs.Close

'rs update..
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close

'fetch the table back...
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend

'rs Scrolling
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print str

rs.MoveLast
str = " LAST ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print str

rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print str

'free all resources
rs.Close
queryDef.Close
conn.Close
ws.Close

End Sub
```

RDO: `rs.addNew` and `rs.update`

The following RDO (Remote Data Objects) example creates a table `my_rdo` and demonstrates the use of `rs.addNew` and `rs.update`.

```
Dim rs As rdoResultset
Dim cn As New rdoConnection
Dim cl As rdoColumn
Dim SQL As String

'cn.Connect = "DSN=test;"
cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"
```

```
cn.CursorDriver = rdUseOdbc
cn.EstablishConnection rdDriverPrompt


'drop table my_rdo
SQL = "drop table if exists my_rdo"
cn.Execute SQL, rdExecDirect

'create table my_rdo
SQL = "create table my_rdo(id int, name varchar(20))"
cn.Execute SQL, rdExecDirect

'insert - direct
SQL = "insert into my_rdo values (100,'venu')"
cn.Execute SQL, rdExecDirect

SQL = "insert into my_rdo values (200,'MySQL')"
cn.Execute SQL, rdExecDirect

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close

'rs update
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close

'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
Do Until rs.EOF
For Each cl In rs.rdoColumns
Debug.Print cl.Value,
Next
rs.MoveNext
Debug.Print
Loop
Debug.Print "Row count="; rs.RowCount

'close
rs.Close
cn.Close

End Sub
```

Using Connector/ODBC with .NET

This section contains simple examples that demonstrate the use of Connector/ODBC drivers with ODBC.NET.

Using Connector/ODBC with ODBC.NET and C# (C sharp)

The following sample creates a table `my_odbc_net` and demonstrates its use in C#.

```
/**
 * @sample    : mycon.cs
 * @purpose   : Demo sample for ODBC.NET using Connector/ODBC
 * @author    : Venu, <myodbc@lists.mysql.com>
 *
 * (C) Copyright MySQL AB, 1995-2006
 *
 **/

/* build command
 *
 *  csc /t:exe
 *      /out:mycon.exe mycon.cs
 *      /r:Microsoft.Data.Odbc.dll
 */

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3
{
  class mycon
  {
    static void Main(string[] args)
    {
      try
        {
          //Connection string for MyODBC 2.50
          /*string MyConString = "DRIVER={MySQL};" +
            "SERVER=localhost;" +
            "DATABASE=test;" +
            "UID=venu;" +
            "PASSWORD=venu;" +
            "OPTION=3";
          */
          //Connection string for Connector/ODBC 3.51
          string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
            "SERVER=localhost;" +
            "DATABASE=test;" +
            "UID=venu;" +
            "PASSWORD=venu;" +
            "OPTION=3";

          //Connect to MySQL using Connector/ODBC
          OdbcConnection MyConnection = new OdbcConnection(MyConString);
          MyConnection.Open();

          Console.WriteLine("\n !!! success, connected successfully !!!\n");

          //Display connection information
          Console.WriteLine("Connection Information:");
          Console.WriteLine("\tConnection String:" +
                            MyConnection.ConnectionString);
          Console.WriteLine("\tConnection Timeout:" +
                            MyConnection.ConnectionTimeout);
          Console.WriteLine("\tDatabase:" +
                            MyConnection.Database);
```

```
Console.WriteLine("\tDataSource:" +
                  MyConnection.DataSource);
Console.WriteLine("\tDriver:" +
                  MyConnection.Driver);
Console.WriteLine("\tServerVersion:" +
                  MyConnection.ServerVersion);

//Create a sample table
OdbcCommand MyCommand =
  new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",
                  MyConnection);
MyCommand.ExecuteNonQuery();
MyCommand.CommandText =
  "CREATE TABLE my_odbc_net(id int, name varchar(20), idb bigint)";
MyCommand.ExecuteNonQuery();

//Insert
MyCommand.CommandText =
  "INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
Console.WriteLine("INSERT, Total rows affected:" +
                  MyCommand.ExecuteNonQuery());;

//Insert
MyCommand.CommandText =
  "INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
Console.WriteLine("INSERT, Total rows affected:" +
                  MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
  "INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
Console.WriteLine("INSERT, Total rows affected:" +
                  MyCommand.ExecuteNonQuery());

//Update
MyCommand.CommandText =
  "UPDATE my_odbc_net SET id=999 WHERE id=20";
Console.WriteLine("Update, Total rows affected:" +
                  MyCommand.ExecuteNonQuery());

//COUNT(*)
MyCommand.CommandText =
  "SELECT COUNT(*) as TRows FROM my_odbc_net";
Console.WriteLine("Total Rows:" +
                  MyCommand.ExecuteScalar());

//Fetch
MyCommand.CommandText = "SELECT * FROM my_odbc_net";
OdbcDataReader MyDataReader;
MyDataReader =  MyCommand.ExecuteReader();
while (MyDataReader.Read())
  {
    if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
      //Supported only by Connector/ODBC 3.51
      Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
                        MyDataReader.GetString(1) + " " +
                        MyDataReader.GetInt64(2));
    }
    else {
      //BIGINTs not supported by Connector/ODBC
      Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
                        MyDataReader.GetString(1) + " " +
                        MyDataReader.GetInt32(2));
    }
  }

//Close all resources
```

```
          MyDataReader.Close();
          MyConnection.Close();
        }
      catch (OdbcException MyOdbcException) //Catch any ODBC exception ..
        {
          for (int i=0; i < MyOdbcException.Errors.Count; i++)
            {
              Console.Write("ERROR #" + i + "\n" +
                            "Message: " +
                            MyOdbcException.Errors[i].Message + "\n" +
                            "Native: " +
                            MyOdbcException.Errors[i].NativeError.ToString() + "\n" +
                            "Source: " +
                            MyOdbcException.Errors[i].Source + "\n" +
                            "SQL: " +
                            MyOdbcException.Errors[i].SQLState + "\n");
            }
        }
    }
  }
}
```

Using Connector/ODBC with ODBC.NET and Visual Basic

The following sample creates a table `my_vb_net` and demonstrates the use in VB.

```
' @sample    : myvb.vb
' @purpose   : Demo sample for ODBC.NET using Connector/ODBC
' @author    : Venu, <myodbc@lists.mysql.com>
'
' (C) Copyright MySQL AB, 1995-2006
'
'


'
' build command
'
' vbc /target:exe
'     /out:myvb.exe
'     /r:Microsoft.Data.Odbc.dll
'     /r:System.dll
'     /r:System.Data.dll
'

Imports Microsoft.Data.Odbc
Imports System

Module myvb
  Sub Main()
    Try

      'Connector/ODBC 3.51 connection string
      Dim MyConString As String = "DRIVER={MySQL ODBC 3.51 Driver};" & _
      "SERVER=localhost;" & _
      "DATABASE=test;" & _
      "UID=venu;" & _
      "PASSWORD=venu;" & _
      "OPTION=3;"

      'Connection
      Dim MyConnection As New OdbcConnection(MyConString)
      MyConnection.Open()

      Console.WriteLine("Connection State::" & MyConnection.State.ToString)

      'Drop
      Console.WriteLine("Dropping table")
```

```
    Dim MyCommand As New OdbcCommand()
    MyCommand.Connection = MyConnection
    MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"
    MyCommand.ExecuteNonQuery()

    'Create
    Console.WriteLine("Creating....")
    MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name varchar(30))"
    MyCommand.ExecuteNonQuery()

    'Insert
    MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"
    Console.WriteLine("INSERT, Total rows affected:" & _
    MyCommand.ExecuteNonQuery())

    'Insert
    MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
    Console.WriteLine("INSERT, Total rows affected:" & _
    MyCommand.ExecuteNonQuery())

    'Insert
    MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
    Console.WriteLine("INSERT, Total rows affected:" & _
    MyCommand.ExecuteNonQuery())

    'Insert
    MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
    Console.WriteLine("INSERT, Total rows affected:" & _
                      MyCommand.ExecuteNonQuery())

    'Update
    MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"
    Console.WriteLine("Update, Total rows affected:" & _
    MyCommand.ExecuteNonQuery())

    'COUNT(*)
    MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"
    Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

    'Select
    Console.WriteLine("Select * FROM my_vb_net")
    MyCommand.CommandText = "SELECT * FROM my_vb_net"
    Dim MyDataReader As OdbcDataReader
    MyDataReader = MyCommand.ExecuteReader
    While MyDataReader.Read
      If MyDataReader("name") Is DBNull.Value Then
        Console.WriteLine("id = " & _
        CStr(MyDataReader("id")) & "  name = " & _
        "NULL")
      Else
        Console.WriteLine("id = " & _
        CStr(MyDataReader("id")) & "  name = " & _
        CStr(MyDataReader("name")))
      End If
    End While

    'Catch ODBC Exception
  Catch MyOdbcException As OdbcException
    Dim i As Integer
    Console.WriteLine(MyOdbcException.ToString)

    'Catch program exception
  Catch MyException As Exception
    Console.WriteLine(MyException.ToString)
  End Try
End Sub
```

25.1.5. Connector/ODBC Reference

This section provides reference material for the Connector/ODBC API, showing supported functions and methods, supported MySQL column types and the corresponding native type in Connector/ODBC, and the error codes returned by Connector/ODBC when a fault occurs.

25.1.5.1. Connector/ODBC API Reference

This section summarizes ODBC routines, categorized by functionality.

For the complete ODBC API reference, please refer to the ODBC Programer's Reference at http://msdn.microsoft.com/library/en-us/odbc/htm/odbcabout_this_manual.asp.

An application can call `SQLGetInfo` function to obtain conformance information about Connector/ODBC. To obtain information about support for a specific function in the driver, an application can call `SQLGetFunctions`.

Note: For backward compatibility, the Connector/ODBC 3.51 driver supports all deprecated functions.

The following tables list Connector/ODBC API calls grouped by task:

Connecting to a data source:

Function name	Connector/ODBC 2.50	Connector/ODBC 3.51	Standard	Purpose
`SQLAllocHandle`	No	Yes	ISO 92	Obtains an environment, connection, statement, or descriptor handle.
`SQLConnect`	Yes	Yes	ISO 92	Connects to a specific driver by data source name, user ID, and password.
`SQLDriverConnect`	Yes	Yes	ODBC	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
`SQLAllocEnv`	Yes	Yes	Deprecated	Obtains an environment handle allocated from driver.
`SQLAllocConnect`	Yes	Yes	Deprecated	Obtains a connection handle

Obtaining information about a driver and data source:

Function name	Connector/ODBC 2.50	Connector/ODBC 3.51	Standard	Purpose
`SQLDataSources`	No	No	ISO 92	Returns the list of available data sources, handled by the Driver Manager
`SQLDrivers`	No	No	ODBC	Returns the list of installed drivers and their attributes, handles by Driver Manager
`SQLGetInfo`	Yes	Yes	ISO 92	Returns information about a specific driver and data source.
`SQLGetFunctions`	Yes	Yes	ISO 92	Returns supported driver functions.

SQLGetTypeInfo	Yes	Yes	ISO 92	Returns information about supported data types.

Setting and retrieving driver attributes:

Function name	Connector/ODBC 2.50	Connector/ODBC 3.51	Standard	Purpose
SQLSetConnectAttr	No	Yes	ISO 92	Sets a connection attribute.
SQLGetConnectAttr	No	Yes	ISO 92	Returns the value of a connection attribute.
SQLSetConnectOption	Yes	Yes	Deprecated	Sets a connection option
SQLGetConnectOption	Yes	Yes	Deprecated	Returns the value of a connection option
SQLSetEnvAttr	No	Yes	ISO 92	Sets an environment attribute.
SQLGetEnvAttr	No	Yes	ISO 92	Returns the value of an environment attribute.
SQLSetStmtAttr	No	Yes	ISO 92	Sets a statement attribute.
SQLGetStmtAttr	No	Yes	ISO 92	Returns the value of a statement attribute.
SQLSetStmtOption	Yes	Yes	Deprecated	Sets a statement option
SQLGetStmtOption	Yes	Yes	Deprecated	Returns the value of a statement option

Preparing SQL requests:

Function name	Connector/ODBC 2.50	Connector/ODBC 3.51	Standard	Purpose
SQLAllocStmt	Yes	Yes	Deprecated	Allocates a statement handle
SQLPrepare	Yes	Yes	ISO 92	Prepares an SQL statement for later execution.
SQLBindParameter	Yes	Yes	ODBC	Assigns storage for a parameter in an SQL statement.
SQLGetCursorName	Yes	Yes	ISO 92	Returns the cursor name associated with a statement handle.
SQLSetCursorName	Yes	Yes	ISO 92	Specifies a cursor name.
SQLSetScrollOptions	Yes	Yes	ODBC	Sets options that control cursor behavior.

Submitting requests:

Function name	Connector/ODBC 2.50	Connector/ODBC 3.51	Standard	Purpose
SQLExecute	Yes	Yes	ISO 92	Executes a prepared statement.
SQLExecDirect	Yes	Yes	ISO 92	Executes a statement
SQLNativeSql	Yes	Yes	ODBC	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	Yes	Yes	ODBC	Returns the description for a specific parameter in a statement.
SQLNumParams	Yes	Yes	ISO 92	Returns the number of parameters in a statement.

SQLParamData	Yes	Yes	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)
SQLPutData	Yes	Yes	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)

Retrieving results and information about results:

	Connector/ODBC			
Function name	**2.50**	**3.51**	**Standard**	**Purpose**
SQLRowCount	Yes	Yes	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols	Yes	Yes	ISO 92	Returns the number of columns in the result set.
SQLDescribeCol	Yes	Yes	ISO 92	Describes a column in the result set.
SQLColAttribute	No	Yes	ISO 92	Describes attributes of a column in the result set.
SQLColAttributes	Yes	Yes	Deprecated	Describes attributes of a column in the result set.
SQLFetch	Yes	Yes	ISO 92	Returns multiple result rows.
SQLFetchScroll	No	Yes	ISO 92	Returns scrollable result rows.
SQLExtendedFetch	Yes	Yes	Deprecated	Returns scrollable result rows.
SQLSetPos	Yes	Yes	ODBC	Positions a cursor within a fetched block of data and allows an application to refresh data in the rowset or to update or delete data in the result set.
SQLBulkOperations	No	Yes	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Retrieving error or diagnostic information:

	Connector/ODBC			
Function name	**2.50**	**3.51**	**Standard**	**Purpose**
SQLError	Yes	Yes	Deprecated	Returns additional error or status information
SQLGetDiagField	Yes	Yes	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
SQLGetDiagRec	Yes	Yes	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).

Obtaining information about the data source's system tables (catalog functions) item:

	Connector/ODBC			
Function name	**2.50**	**3.51**	**Standard**	**Purpose**
SQLColumnPrivileges	Yes	Yes	ODBC	Returns a list of columns and associated privileges for one or more tables.

SQLColumns	Yes	Yes	X/Open	Returns the list of column names in specified tables.
SQLForeignKeys	Yes	Yes	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Yes	Yes	ODBC	Returns the list of column names that make up the primary key for a table.
SQLSpecialColumns	Yes	Yes	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Yes	Yes	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Yes	Yes	ODBC	Returns a list of tables and the privileges associated with each table.
SQLTables	Yes	Yes	X/Open	Returns the list of table names stored in a specific data source.

Performing transactions:

	Connector/ODBC			
Function name	**2.50**	**3.51**	**Standard**	**Purpose**
SQLTransact	Yes	Yes	Deprecated	Commits or rolls back a transaction
SQLEndTran	No	Yes	ISO 92	Commits or rolls back a transaction.

Terminating a statement:

	Connector/ODBC			
Function name	**2.50**	**3.51**	**Standard**	**Purpose**
SQLFreeStmt	Yes	Yes	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCloseCursor	Yes	Yes	ISO 92	Closes a cursor that has been opened on a statement handle.
SQLCancel	Yes	Yes	ISO 92	Cancels an SQL statement.

Terminating a connection:

	Connector/ODBC			
Function name	**2.50**	**3.51**	**Standard**	**Purpose**
SQLDisconnect	Yes	Yes	ISO 92	Closes the connection.
SQLFreeHandle	No	Yes	ISO 92	Releases an environment, connection, statement, or descriptor handle.
SQLFreeConnect	Yes	Yes	Deprecated	Releases connection handle

SQLFreeEnv	Yes	Yes	Deprecated	Releases an environment handle

25.1.5.2. Connector/ODBC Data Types

The following table illustrates how driver maps the server data types to default SQL and C data types:

Native Value	SQL Type	C Type
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT
double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
text	SQL_VARCHAR	SQL_C_CHAR

varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

25.1.5.3. Connector/ODBC Error Codes

The following tables lists the error codes returned by the driver apart from the server errors.

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
500	01000	01000	General warning
501	01004	01004	String data, right truncated
502	01S02	01S02	Option value changed
503	01S03	01S03	No rows updated/deleted
504	01S04	01S04	More than one row updated/deleted
505	01S06	01S06	Attempt to fetch before the result set returned the first row set
506	07001	07002	SQLBindParameter not used for all parameters
507	07005	07005	Prepared statement not a cursor-specification
508	07009	07009	Invalid descriptor index
509	08002	08002	Connection name in use
510	08003	08003	Connection does not exist
511	24000	24000	Invalid cursor state
512	25000	25000	Invalid transaction state
513	25S01	25S01	Transaction state unknown
514	34000	34000	Invalid cursor name
515	S1000	HY000	General driver defined error
516	S1001	HY001	Memory allocation error
517	S1002	HY002	Invalid column number
518	S1003	HY003	Invalid application buffer type
519	S1004	HY004	Invalid SQL data type
520	S1009	HY009	Invalid use of null pointer
521	S1010	HY010	Function sequence error
522	S1011	HY011	Attribute can not be set now
523	S1012	HY012	Invalid transaction operation code
524	S1013	HY013	Memory management error
525	S1015	HY015	No cursor name available
526	S1024	HY024	Invalid attribute value
527	S1090	HY090	Invalid string or buffer length

528	S1091	HY091	Invalid descriptor field identifier
529	S1092	HY092	Invalid attribute/option identifier
530	S1093	HY093	Invalid parameter number
531	S1095	HY095	Function type out of range
532	S1106	HY106	Fetch type out of range
533	S1117	HY117	Row value out of range
534	S1109	HY109	Invalid cursor position
535	S1C00	HYC00	Optional feature not implemented
0	21S01	21S01	Column count does not match value count
0	23000	23000	Integrity constraint violation
0	42000	42000	Syntax error or access violation
0	42S02	42S02	Base table or view not found
0	42S12	42S12	Index not found
0	42S21	42S21	Column already exists
0	42S22	42S22	Column not found
0	08S01	08S01	Communication link failure

25.1.6. Connector/ODBC Notes and Tips

Here are some common notes and tips for using Connector/ODBC within different environments, applications and tools. The notes provided here are based on the experiences of Connector/ODBC developers and users.

25.1.6.1. Connector/ODBC General Functionality

This section provides help with common queries and areas of functionality in MySQL and how to use them with Connector/ODBC.

Obtaining Auto-Increment Values

Obtaining the value of column that uses `AUTO_INCREMENT` after an `INSERT` statement can be achieved in a number of different ways. To obtain the value immediately after an `INSERT`, use a `SELECT` query with the `LAST_INSERT_ID()` function.

For example, using Connector/ODBC you would execute two separate statements, the `INSERT` statement and the `SELECT` query to obtain the auto-increment value.

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

If you do not require the value within your application, but do require the value as part of another `INSERT`, the entire process can be handled by executing the following statements:

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

Certain ODBC applications (including Delphi and Access) may have trouble obtaining the auto-increment value using the previous examples. In this case, try the following statement as an alternative:

```
SELECT * FROM tbl WHERE auto IS NULL;
```

See Sección 24.2.13.3, “Cómo obtener el ID único del último registro insertado”.

Dynamic Cursor Support

Support for the `dynamic cursor` is provided in Connector/ODBC 3.51, but dynamic cursors are not enabled by default. You can enable this function within Windows by selecting the `Enable Dynamic Cursor` checkbox within the ODBC Data Source Administrator.

On other platforms, you can enable the dynamic cursor by adding `32` to the `OPTION` value when creating the DSN.

Connector/ODBC Performance

The Connector/ODBC driver has been optimized to provide very fast performance. If you experience problems with the performance of Connector/ODBC, or notice a large amount of disk activity for simple queries, there are a number of aspects you should check:

- Ensure that `ODBC Tracing` is not enabled. With tracing enabled, a lot of information is recorded in the tracing file by the ODBC Manager. You can check, and disable, tracing within Windows using the Tracing panel of the ODBC Data Source Administrator. Within Mac OS X, check the Tracing panel of ODBC Administrator. See Sección 25.1.3.8, “Getting an ODBC Trace File”.
- Make sure you are using the standard version of the driver, and not the debug version. The debug version includes additional checks and reporting measures.
- Disable the Connector/ODBC driver trace and query logs. These options are enabled for each DSN, so make sure to examine only the DSN that you are using in your application. Within Windows, you can disable the Connector/ODBC and query logs by modifying the DSN configuration. Within Mac OS X and Unix, ensure that the driver trace (option value 4) and query logging (option value 524288) are not enabled.

Setting ODBC Query Timeout in Windows

For more information on how to set the query timeout on Microsoft Windows when executing queries through an ODBC connection, read the Microsoft knowledgebase document at http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756.

25.1.6.2. Connector/ODBC Application Specific Tips

Most programs should work with Connector/ODBC, but for each of those listed here, there are specific notes and tips to improve or enhance the way you work with Connector/ODBC and these applications.

With all applications you should ensure that you are using the latest Connector/ODBC drivers, ODBC Manager and any supporting libraries and interfaces used by your application. For example, on Windows, using the latest version of Microsoft Data Access Components (MDAC) will improve the compatibility with ODBC in general, and with the Connector/ODBC driver.

Using Connector/ODBC with Microsoft Applications

The majority of Microsoft applications have been tested with Connector/ODBC, including Microsoft Office, Microsoft Access and the various programming languages supported within ASP and Microsoft Visual Studio.

If you have problem with Connector/ODBC and your program also works with OLEDB, you should try the OLEDB driver.

Microsoft Access

To improve the integration between Microsoft Access and MySQL through Connector/ODBC:

- For all versions of Access, you should enable the Connector/ODBC `Return matching rows` option. For Access 2.0, you should additionally enable the `Simulate ODBC 1.0` option.
- You should have a `TIMESTAMP` column in all tables that you want to be able to update. For maximum portability, don't use a length specification in the column declaration (which is unsupported within MySQL in versions earlier than 4.1).
- You should have a primary key in each MySQL table you want to use with Access. If not, new or updated rows may show up as `#DELETED#`.
- Use only `DOUBLE` float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you can't find or update rows.
- If you are using Connector/ODBC to link to a table that has a `BIGINT` column, the results are displayed as `#DELETED#`. The work around solution is:
  - Have one more dummy column with `TIMESTAMP` as the data type.
  - Select the `Change BIGINT columns to INT` option in the connection dialog in ODBC DSN Administrator.
  - Delete the table link from Access and re-create it.

  Old records may still display as `#DELETED#`, but newly added/updated records are displayed properly.
- If you still get the error `Another user has changed your data` after adding a `TIMESTAMP` column, the following trick may help you:

  Don't use a `table` data sheet view. Instead, create a form with the fields you want, and use that `form` data sheet view. You should set the `DefaultValue` property for the `TIMESTAMP` column to `NOW()`. It may be a good idea to hide the `TIMESTAMP` column from view so your users are not confused.
- In some cases, Access may generate SQL statements that MySQL can't understand. You can fix this by selecting `"Query|SQLSpecific|Pass-Through"` from the Access menu.
- On Windows NT, Access reports `BLOB` columns as `OLE OBJECTS`. If you want to have `MEMO` columns instead, you should change `BLOB` columns to `TEXT` with `ALTER TABLE`.
- Access can't always handle the MySQL `DATE` column properly. If you have a problem with these, change the columns to `DATETIME`.
- If you have in Access a column defined as `BYTE`, Access tries to export this as `TINYINT` instead of `TINYINT UNSIGNED`. This gives you problems if you have values larger than 127 in the column.
- If you have very large (long) tables in Access, it might take a very long time to open them. Or you might run low on virtual memory and eventually get an `ODBC Query Failed` error and the table cannot open. To deal with this, select the following options:
  - Return Matching Rows (2)
  - Allow BIG Results (8).

  These add up to a value of 10 (`OPTION=10`).

Some external articles and tips that may be useful when using Access, ODBC and Connector/ODBC:

- Read How to Trap ODBC Login Error Messages in Access

- Optimizing Access ODBC Applications
  - Optimizing for Client/Server Performance
  - Tips for Converting Applications to Using ODBCDirect
  - Tips for Optimizing Queries on Attached SQL Tables
- For a list of tools that can be used with Access and ODBC data sources, refer to converters section for list of available tools.

Microsoft Excel and Column Types

If you have problems importing data into Microsoft Excel, particularly numerical, date, and time values, this is probably because of a bug in Excel, where the column type of the source data is used to determine the data type when that data is inserted into a cell within the worksheet. The result is that Excel incorrectly identifies the content and this affects both the display format and the data when it is used within calculations.

To address this issue, use the `CONCAT()` function in your queries. The use of `CONCAT()` forces Excel to treat the value as a string, which Excel will then parse and usually correctly identify the embedded information.

However, even with this option, some data may be incorrectly formatted, even though the source data remains unchanged. Use the `Format Cells` option within Excel to change the format of the displayed information.

Microsoft Visual Basic

To be able to update a table, you must define a primary key for the table.

Visual Basic with ADO can't handle big integers. This means that some queries like `SHOW PROCESSLIST` do not work properly. The fix is to use `OPTION=16384` in the ODBC connect string or to select the `Change BIGINT columns to INT` option in the Connector/ODBC connect screen. You may also want to select the `Return matching rows` option.

Microsoft Visual InterDev

If you have a `BIGINT` in your result, you may get the error `[Microsoft][ODBC Driver Manager] Driver does not support this parameter`. Try selecting the `Change BIGINT columns to INT` option in the Connector/ODBC connect screen.

Visual Objects

You should select the `Don't optimize column widths` option.

Microsoft ADO

When you are coding with the ADO API and Connector/ODBC, you need to pay attention to some default properties that aren't supported by the MySQL server. For example, using the `CursorLocation Property` as `adUseServer` returns a result of –1 for the `RecordCount Property`. To have the right value, you need to set this property to `adUseClient`, as shown in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long
```

```
myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close
```

Another workaround is to use a `SELECT COUNT(*)` statement for a similar query to get the correct row count.

To find the number of rows affected by a specific SQL statement in ADO, use the `RecordsAffected` property in the ADO execute method. For more information on the usage of execute method, refer to http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/mdmthcnnexecute.asp.

For information, see ActiveX Data Objects(ADO) Frequently Asked Questions.

Using Connector/ODBC with Active Server Pages (ASP)

You should select the `Return matching rows` option in the DSN.

For more information about how to access MySQL via ASP using Connector/ODBC, refer to the following articles:

- Using MyODBC To Access Your MySQL Database Via ASP
- ASP and MySQL at DWAM.NT

A Frequently Asked Questions list for ASP can be found at http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp.

Using Connector/ODBC with Visual Basic (ADO, DAO and RDO) and ASP

Some articles that may help with Visual Basic and ASP:

- MySQL BLOB columns and Visual Basic 6 by Mike Hillyer (`<mike@openwin.org>`).
- How to map Visual basic data type to MySQL types by Mike Hillyer (`<mike@openwin.org>`).

Using Connector/ODBC with Borland Applications

With all Borland applications where the Borland Database Engine (BDE) is used, follow these steps to improve compatibility:

- Update to BDE 3.2 or newer.
- Enable the `Don't optimize column widths` option in the DSN.
- Enabled the `Return matching rows` option in the DSN.

Using Connector/ODBC with Borland Builder 4

When you start a query, you can use the `Active` property or the `Open` method. Note that `Active` starts by automatically issuing a `SELECT * FROM ...` query. That may not be a good thing if your tables are large.

Using Connector/ODBC with Delphi

Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for Connector/ODBC. The BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton <bryan@flesherfab.com> for this):

```
fReg:= TRegistry.Create;
fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
fReg.WriteString('Database', 'Documents');
fReg.WriteString('Description', ' ');
fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', ' ');
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;

Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');

AliasEditor.Add('DocumentsFab','MySQL',Memo1.Lines);
```

Using Connector/ODBC with C++ Builder

Tested with BDE 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to recognize primary keys, only the index named `PRIMARY`, although this has not been a problem.

Using Connector/ODBC with ColdFusion

The following information is taken from the ColdFusion documentation:

Use the following information to configure ColdFusion Server for Linux to use the `unixODBC` driver with Connector/ODBC for MySQL data sources. Allaire has verified that MyODBC 2.50.26 works with MySQL 3.22.27 and ColdFusion for Linux. (Any newer version should also work.) You can download Connector/ODBC at http://dev.mysql.com/downloads/connector/odbc/.

ColdFusion version 4.5.1 allows you to us the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion version 4.5.1. Before the MySQL driver appears in the ODBC data sources drop-down list, you must build and copy the Connector/ODBC driver to `/opt/coldfusion/lib/libmyodbc.so`.

The Contrib directory contains the program `mydsn-xxx.zip` which allows you to build and remove the DSN registry file for the Connector/ODBC driver on ColdFusion applications.

For more information and guides on using ColdFusion and Connector/ODBC, see the following external sites:

- Troubleshooting Data Sources and Database Connectivity for Unix Platforms.

Using Connector/ODBC with OpenOffice

Open Office (http://www.openoffice.org) How-to: MySQL + OpenOffice. How-to: OpenOffice + MyODBC + unixODBC.

Using Connector/ODBC with Sambar Server

Sambar Server (http://www.sambarserver.info) How-to: MyODBC + SambarServer + MySQL.

Using Connector/ODBC with Pervasive Software DataJunction

You have to change it to output `VARCHAR` rather than `ENUM`, as it exports the latter in a manner that causes MySQL problems.

Using Connector/ODBC with SunSystems Vision

You should select the `Return matching rows` option.

25.1.6.3. Connector/ODBC Errors and Resolutions

The following section details some common errors and their suggested fix or alternative solution. If you are still experiencing problems, use the Connector/ODBC mailing list; see Sección 25.1.7.1, “Connector/ODBC Community Support”.

Many problems can be resolved by upgrading your Connector/ODBC drivers to the latest available release. On Windows, you should also make sure that you have the latest versions of the Microsoft Data Access Components (MDAC) installed.

Questions

- 25.1.6.3.1: [1261] Are MyODBC 2.50 applications compatible with Connector/ODBC 3.51?
- 25.1.6.3.2: [1261] I have installed Connector/ODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation completed successfully, but the Connector/ODBC driver does not appear in `ODBC Data Source Administrator`.
- 25.1.6.3.3: [1261] When connecting or using the Test button in `ODBC Data Source Administrator` I get error 10061 (Cannot connect to server)
- 25.1.6.3.4: [1261] The following error is reported when using transactions: `Transactions are not enabled`
- 25.1.6.3.5: [1262] The following error is reported when I submit a query: `Cursor not found`
- 25.1.6.3.6: [1262] Access reports records as `#DELETED#` when inserting or updating records in linked tables.
- 25.1.6.3.7: [1262] How do I handle Write Conflicts or Row Location errors?
- 25.1.6.3.8: [1263] Exporting data from Access 97 to MySQL reports a `Syntax Error`.
- 25.1.6.3.9: [1263] Exporting data from Microsoft DTS to MySQL reports a `Syntax Error`.
- 25.1.6.3.10: [1263] Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception.

- 25.1.6.3.11: [1263] Using `SELECT COUNT(*) FROM tbl_name` within Visual Basic and ASP returns an error.
- 25.1.6.3.12: [1263] Using the `AppendChunk()` or `GetChunk()` ADO methods, the `Multiple-step operation generated errors. Check each status value` error is returned.
- 25.1.6.3.13: [1263] Access Returns `Another user had modified the record that you have modified` while editing records on a Linked Table.
- 25.1.6.3.14: [1263] When linking an application directly to the Connector/ODBC library under Unix/Linux, the application crashes.
- 25.1.6.3.15: [1263] Applications in the Microsoft Office suite are unable to update tables that have `DATE` or `TIMESTAMP` columns.

Questions and Answers

25.1.6.3.1: Are MyODBC 2.50 applications compatible with Connector/ODBC 3.51?

Applications based on MyODBC 2.50 should work fine with Connector/ODBC 3.51 and later versions. If you find something is not working with the latest version of Connector/ODBC which previously worked under an earlier version, please file a bug report. See Sección 25.1.7.2, “How to Report Connector/ODBC Problems or Bugs”.

25.1.6.3.2: I have installed Connector/ODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation completed successfully, but the Connector/ODBC driver does not appear in `ODBC Data Source Administrator`.

This is not a bug, but is related to the way Windows x64 editions operate with the ODBC driver. On Windows x64 editions, the Connector/ODBC driver is installed in the `%SystemRoot%\SysWOW64` folder. However, the default `ODBC Data Source Administrator` that is available through the `Administrative Tools` or `Control Panel` in Windows x64 Editions is located in the `%SystemRoot%\system32` folder, and only searches this folder for ODBC drivers.

On Windowx x64 editions, you should use the ODBC administration tool located at `%SystemRoot%\SysWOW64\odbcad32.exe`, this will correctly locate the installed Connector/ODBC drivers and enable you to create a Connector/ODBC DSN.

This issue was originally reported as Bug #20301.

25.1.6.3.3: When connecting or using the `Test` button in `ODBC Data Source Administrator` I get error 10061 (Cannot connect to server)

This error can be raised by a number of different issues, including server problems, network problems, and firewall and port blocking problems. For more information, see Sección A.2.2, “`Can't connect to [local] MySQL server`”.

25.1.6.3.4: The following error is reported when using transactions: `Transactions are not enabled`

This error indicates that you are trying to use transactions with a MySQL table that does not support transactions. Transactions are supported within MySQL when using the `InnoDB` database engine. In versions of MySQL before Mysql 5.1 you may also use the `BDB` engine.

You should check the following before continuing:

- Verify that your MySQL server supports a transactional database engine. Use `SHOW ENGINES` to obtain a list of the available engine types.

- Verify that the tables you are updating use a transaction database engine.
- Ensure that you have not enabled the `disable transactions` option in your DSN.

25.1.6.3.5: The following error is reported when I submit a query: `Cursor not found`

This occurs because the application is using the old MyODBC 2.50 version, and it did not set the cursor name explicitly through SQLSetCursorName. The fix is to upgrade to Connector/ODBC 3.51 version.

25.1.6.3.6: Access reports records as `#DELETED#` when inserting or updating records in linked tables.

If the inserted or updated records are shown as `#DELETED#` in the access, then:

- If you are using Access 2000, you should get and install the newest (version 2.6 or higher) Microsoft MDAC (`Microsoft Data Access Components`) from http://www.microsoft.com/data/. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified. Another way to work around this bug is to upgrade to MyODBC 2.50.33 or higher and MySQL 3.23.x or higher, which together provide a workaround for the problem.

  You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found at http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114. This fixes some cases where columns are marked as `#DELETED#` in Access.

  Note: If you are using MySQL 3.22, you must apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and up to work around this problem.
- For all versions of Access, you should enable the Connector/ODBC `Return matching rows` option. For Access 2.0, you should additionally enable the `Simulate ODBC 1.0` option.
- You should have a timestamp in all tables that you want to be able to update.
- You should have a primary key in the table. If not, new or updated rows may show up as `#DELETED#`.
- Use only `DOUBLE` float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you can't find or update rows.
- If you are using Connector/ODBC to link to a table that has a `BIGINT` column, the results are displayed as `#DELETED`. The work around solution is:
  - Have one more dummy column with `TIMESTAMP` as the data type.
  - Select the `Change BIGINT columns to INT` option in the connection dialog in ODBC DSN Administrator.
  - Delete the table link from Access and re-create it.

  Old records still display as `#DELETED#`, but newly added/updated records are displayed properly.

25.1.6.3.7: How do I handle Write Conflicts or Row Location errors?

If you see the following errors, select the `Return Matching Rows` option in the DSN configuration dialog, or specify `OPTION=2`, as the connection parameter:

```
Write Conflict. Another user has changed your data.

Row cannot be located for updating. Some values may have been changed
since it was last read.
```

25.1.6.3.8: Exporting data from Access 97 to MySQL reports a `Syntax Error`.

This error is specific to Access 97 and versions of Connector/ODBC earlier than 3.51.02. Update to the latest version of the Connector/ODBC driver to resolve this problem.

25.1.6.3.9: Exporting data from Microsoft DTS to MySQL reports a `Syntax Error`.

This error occurs only with MySQL tables using the `TEXT` or `VARCHAR` data types. You can fix this error by upgrading your Connector/ODBC driver to version 3.51.02 or higher.

25.1.6.3.10: Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception.

You can get the patch that addresses this problem from http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243.

25.1.6.3.11: Using `SELECT COUNT(*) FROM tbl_name` within Visual Basic and ASP returns an error.

This error occurs because the `COUNT(*)` expression is returning a `BIGINT`, and ADO can't make sense of a number this big. Select the `Change BIGINT columns to INT` option (option value 16384).

25.1.6.3.12: Using the `AppendChunk()` or `GetChunk()` ADO methods, the `Multiple-step operation generated errors. Check each status value` error is returned.

The `GetChunk()` and `AppendChunk()` methods from ADO doesn't work as expected when the cursor location is specified as `adUseServer`. On the other hand, you can overcome this error by using `adUseClient`.

A simple example can be found from http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

25.1.6.3.13: Access Returns `Another user had modified the record that you have modified` while editing records on a Linked Table.

In most cases, this can be solved by doing one of the following things:

- Add a primary key for the table if one doesn't exist.
- Add a timestamp column if one doesn't exist.
- Only use double-precision float fields. Some programs may fail when they compare single-precision floats.

If these strategies don't help, you should start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a Connector/ODBC log to help you figure out why things go wrong. For instructions, see Sección 25.1.3.8, “Getting an ODBC Trace File”.

25.1.6.3.14: When linking an application directly to the Connector/ODBC library under Unix/Linux, the application crashes.

Connector/ODBC 3.51 under Unix/Linux is not compatible with direct application linking. You must use a driver manager, such as iODBC or unixODBC to connect to an ODBC source.

25.1.6.3.15: Applications in the Microsoft Office suite are unable to update tables that have `DATE` or `TIMESTAMP` columns.

This is a known issue with Connector/ODBC. You must ensure that the field has a default value (rather than `NULL` and that the default value is non-zeo (i.e. the default value is not `0000-00-00 00:00:00`).

25.1.7. Connector/ODBC Support

There are many different places where you can get support for using Connector/ODBC. You should always try the Connector/ODBC Mailing List or Connector/ODBC Forum. See Sección 25.1.7.1, “Connector/ODBC Community Support”, for help before reporting a specific bug or issue to MySQL.

25.1.7.1. Connector/ODBC Community Support

MySQL AB provides assistance to the user community by means of its mailing lists. For Connector/ODBC-related issues, you can get help from experienced users by using the `<myodbc@lists.mysql.com>` mailing list. Archives are available online at http://lists.mysql.com/myodbc.

For information about subscribing to MySQL mailing lists or to browse list archives, visit http://lists.mysql.com/. See Sección 1.6.1.1, “Las listas de correo de MySQL”.

Community support from experienced users is also available through the ODBC Forum. You may also find help from other users in the other MySQL Forums, located at http://forums.mysql.com. See Sección 1.6.3, “Soporte por parte de la comunidad en los foros de MySQL”.

25.1.7.2. How to Report Connector/ODBC Problems or Bugs

If you encounter difficulties or problems with Connector/ODBC, you should start by making a log file from the `ODBC Manager` (the log you get when requesting logs from `ODBC ADMIN`) and Connector/ODBC. The procedure for doing this is described in Sección 25.1.3.8, “Getting an ODBC Trace File”.

Check the Connector/ODBC trace file to find out what could be wrong. You should be able to determine what statements were issued by searching for the string `>mysql_real_query` in the `myodbc.log` file.

You should also try issuing the statements from the `mysql` client program or from `admndemo`. This helps you determine whether the error is in Connector/ODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (maximum 40 rows) to the `myodbc` mailing list. See Sección 1.6.1.1, “Las listas de correo de MySQL”. Please never send the whole Connector/ODBC or ODBC log file!

You should ideally include the following information with the email:

- Operating system and version
- Connector/ODBC version
- ODBC Driver Manager type and version
- MySQL server version
- ODBC trace from Driver Manager
- Connector/ODBC log file from Connector/ODBC driver
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

Also, before posting the bug, check the MyODBC mailing list archive at http://lists.mysql.com/myodbc.

If you are unable to find out what's wrong, the last option is to create an archive in `tar` or Zip format that contains a Connector/ODBC trace file, the ODBC log file, and a `README` file that explains the problem. You can send this to ftp://ftp.mysql.com/pub/mysql/upload/. Only MySQL engineers have access to the files you upload, and we are very discreet with the data.

If you can create a program that also demonstrates the problem, please include it in the archive as well.

If the program works with another SQL server, you should include an ODBC log file where you perform exactly the same SQL statements so that we can compare the results between the two systems.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

25.1.7.3. How to Submit a Connector/ODBC Patch

You can send a patch or suggest a better solution for any existing code or problems by sending a mail message to `<myodbc@lists.mysql.com>`.

25.1.7.4. Connector/ODBC Change History

The Connector/ODBC Change History (Changelog) is located with the main Changelog for MySQL. See Sección C.2, “Cambios en MySQL Connector/ODBC (MyODBC)”.

25.1.7.5. Credits

These are the developers that have worked on the Connector/ODBC and Connector/ODBC 3.51 Drivers from MySQL AB.

- Michael (Monty) Widenius
- Venu Anuganti
- Peter Harvey

25.2. MySQL Connector/NET

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. Connector/NET is a fully managed ADO.NET driver written in 100% pure C#.

Connector/NET includes full support for:

- MySQL 5.0 features (such as stored procedures)
- MySQL 4.1 features (server-side prepared statements, Unicode, and shared memory access, and so forth)
- Large-packet support for sending and receiving rows and BLOBs up to 2 gigabytes in size.
- Protocol compression which allows for compressing the data stream between the client and server.
- Support for connecting using TCP/IP sockets, named pipes, or shared memory on Windows.
- Support for connecting using TCP/IP sockets or Unix sockets on Unix.
- Support for the Open Source Mono framework developed by Novell.
- Fully managed, does not utilize the MySQL client library.

This document is intended as a user's guide to Connector/NET and includes a full syntax reference. Syntax information is also included within the `Documentation.chm` file included with the Connector/NET distribution.

If you are using MySQL 5.0 or later, and Visual Studio as your development environment, you may want also want to use the MySQL Visual Studio Plugin. The plugin acts as a DDEX (Data Designer Extensibility) provider, enabling you to use the data design tools within Visual Studio to manipulate the schema and objects within a MySQL database. For more information, see Sección 25.3, “MySQL Visual Studio Plugin”.

25.2.1. Connector/NET Versions

There is currently one version of Connector/NET available:

- Connector/NET 1.0 includes support for MySQL 4.0, and MySQL 5.0 features, and full compatibility with the ADO.NET driver interface.

> **Nota**
>
> Version numbers for MySQL products are formatted as X.X.X. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.XX.XX. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

25.2.2. Connector/NET Installation

Connector/NET runs on any platform that supports the .NET framework. The .NET framework is primarily supported on recent versions of Microsoft Windows, and is supported on Linux through the Open Source Mono framework (see http://www.mono-project.com).

Connector/NET is available for download from http://dev.mysql.com/downloads/connector/net/1.0.html.

25.2.2.1. Installing Connector/NET on Windows

On Windows, installation is supported either through a binary installation process or by downloading a Zip file with the Connector/NET components.

Before installing, you should ensure that your system is up to date, including installing the latest version of the .NET Framework.

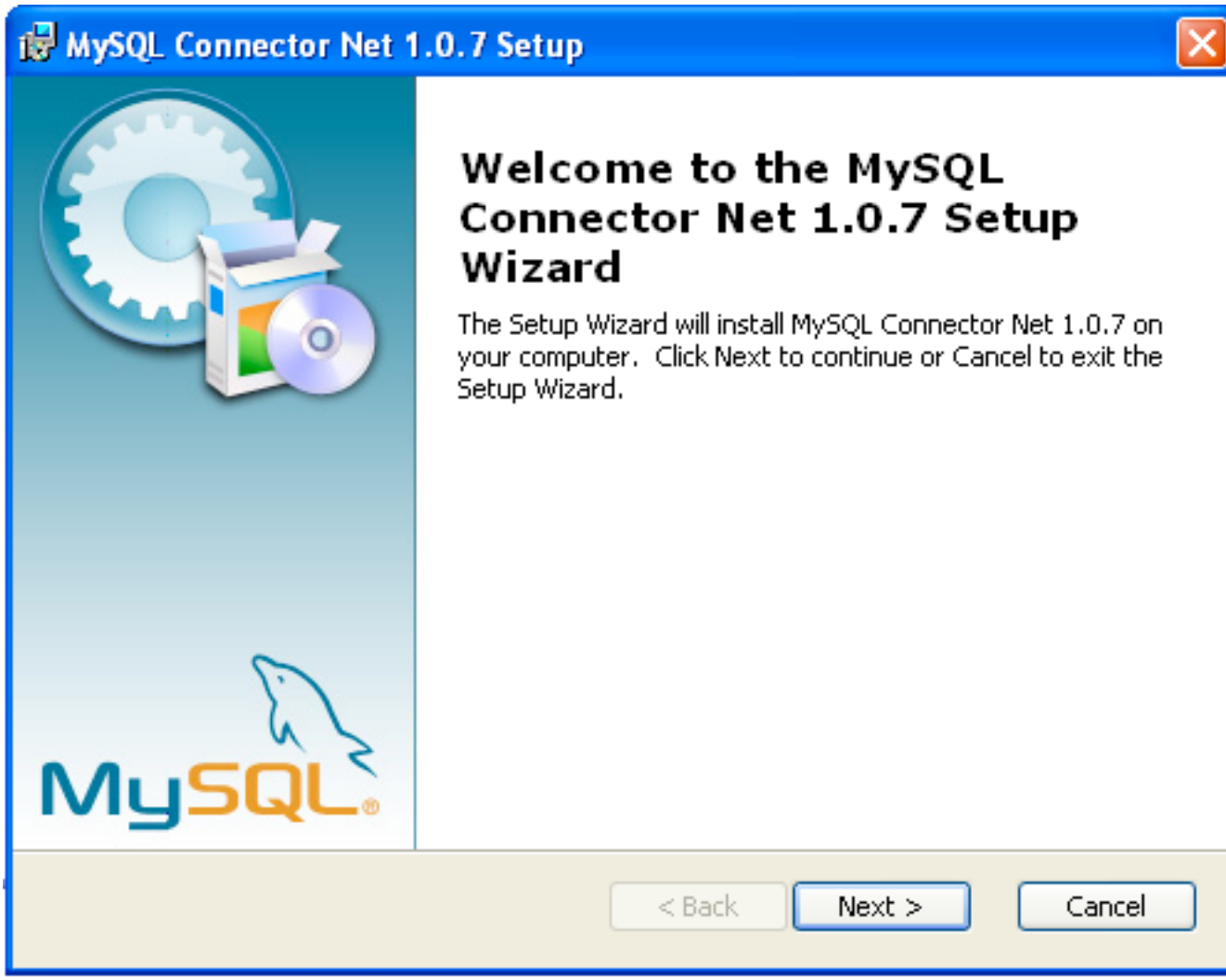
Installing Connector/NET using the Installer

Using the installer is the most straightforward method of installing Connector/NET on Windows and the installed components include the source code, test code and full reference documentation.

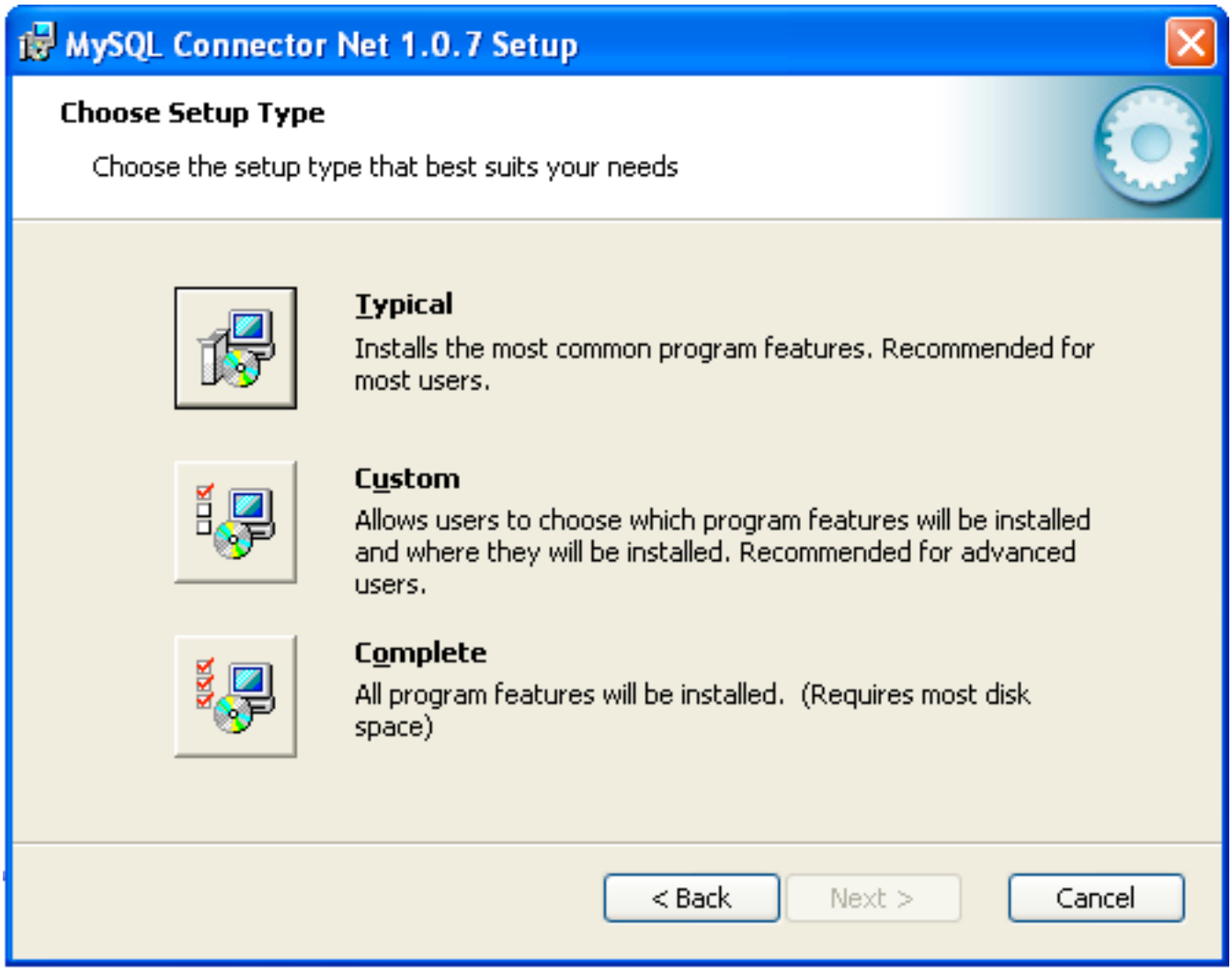
Connector/NET is installed through the use of a Windows Installer (`.msi`) installation package, which can be used to install Connector/NET on all Windows operating systems. The MSI package in contained within a ZIP archive named `mysql-connector-net-`*`version`*`.zip`, where *`version`* indicates the Connector/NET version.

To install Connector/NET:

1. Double click on the MSI installer file extracted from the Zip you downloaded. Click Next to start the installation.



2. You must choose the type of installation that you want to perform.



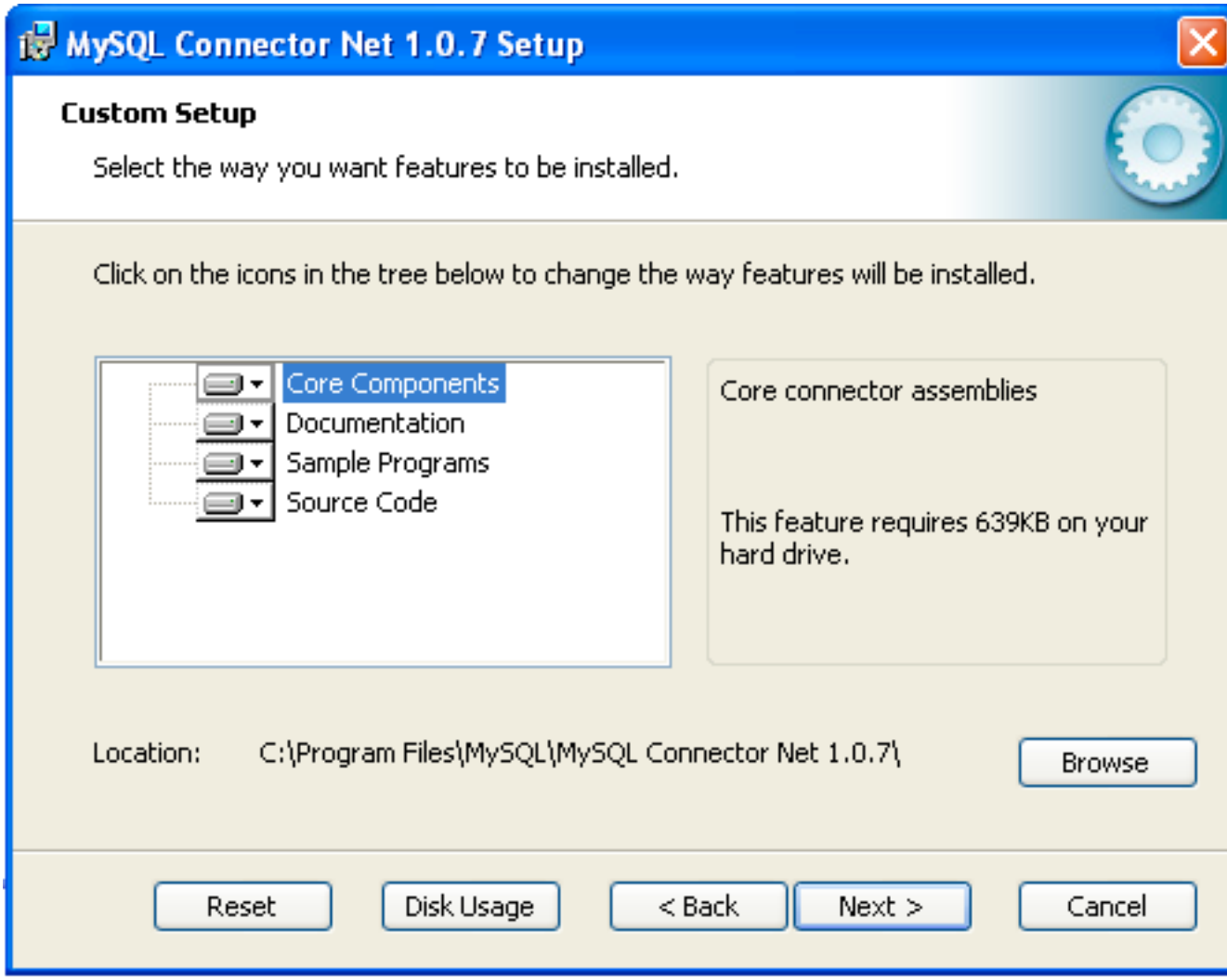
For most situations, the Typical installation will be suitable. Click the Typical button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the Complete button and proceed to step 5. If you want to customize your installation, including choosing the components to install and some installation options, click the Custom button and proceed to Step 3.

The Connector/NET installer will register the connector within the Global Assembly Cache (GAC) - this will make the Connector/NET component available to all applications, not just those where you explicitly reference the Connector/NET component. The installer will also create the necessary links in the Start menu to the documentation and release notes.

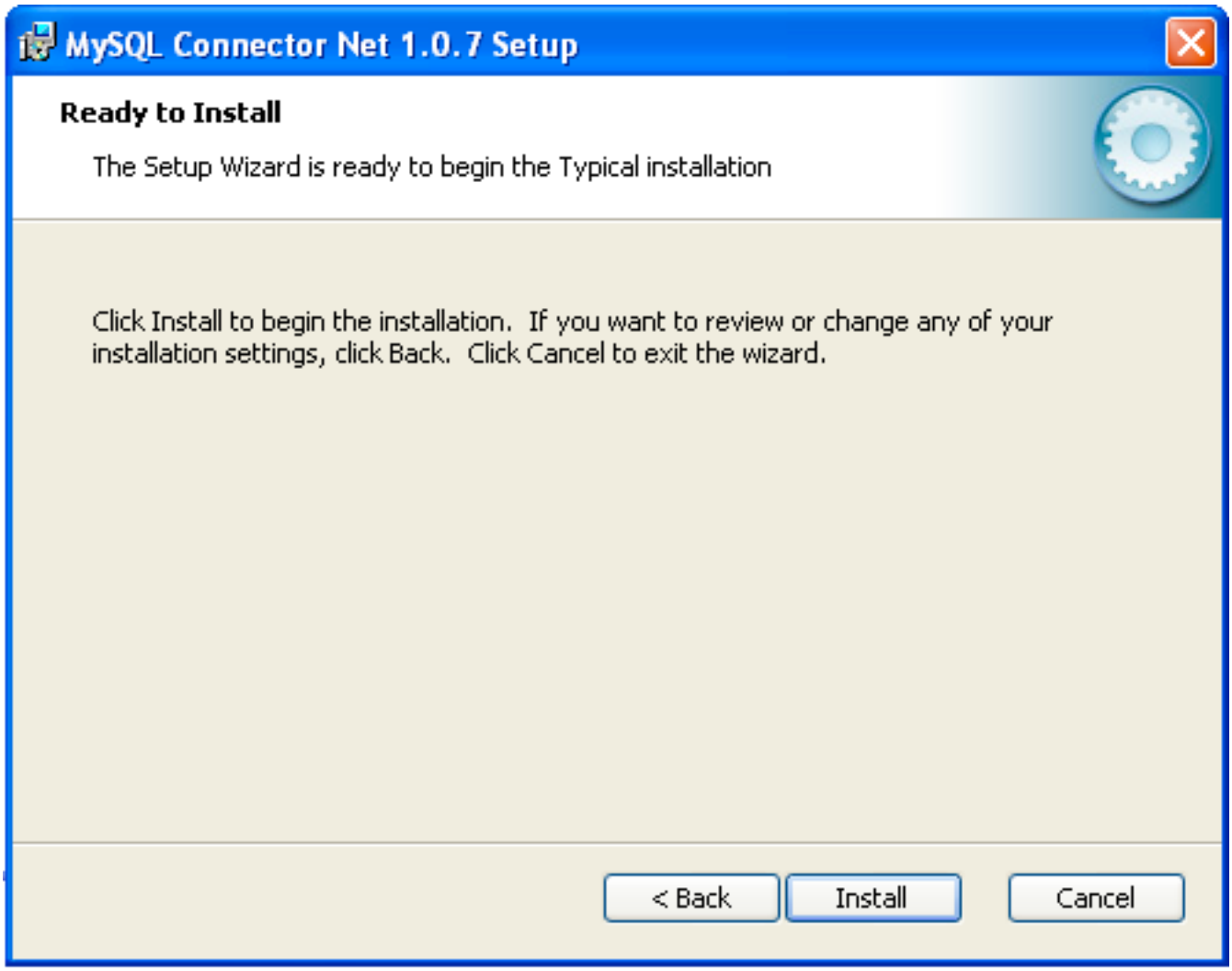
3. If you have chosen a custom installation, you can select the individual components that you want to install, including the core interface component, supporting documentation (a CHM file) samples and examples and the source code. Select the items, and their installation level, and then click Next to continue the installation.

> **Nota**
>
> For Connector/NET 1.0.8 or lower and Connector 5.0.4 and lower the installer will attempt to install binaries for both 1.x and 2.x of the .NET Framework. If you only have one version of the framework installed, the connector installation

may fail. If this happens, you can choose the framework version to be installed through the custom installation step.



4. You will be given a final opportunity to confirm the installation. Click Install to copy and install the files onto your machine.



5. Once the installation has been completed, click Finish to exit the installer.

Unless you choose otherwise, Connector/NET is installed in `C:\Program Files\MySQL\MySQL Connector Net X.X.X`, where `X.X.X` is replaced with the version of Connector/NET you are installing. New installations do not overwrite existing versions of Connector/NET.

Depending on your installation type, the installed components will include some or all of the following components:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `docs` - contains a CHM of the Connector/NET documentation.
- `samples` - sample code and applications that use the Connector/NET component.
- `src` - the source code for the Connector/NET component.

You may also use the `/quiet` or `/q` command line option with the `msiexec` tool to install the Connector/NET package automatically (using the default options) with no notification to the user. Using this option you cannot select options and no prompts, messages or dialog boxes will be displayed.

```
C:\> msiexec /package conector-net.msi /quiet
```

To provide a progress bar to the user during automatic installation, but still without presenting the user with a dialog box of the ability to select options, use the `/passive` option.

Installing Connector/NET using the Zip package

If you are having problems running the installer, you can download a .zip file without an installer as an alternative. That file is called `mysql-connector-net-version-noinstall.zip`. Once downloaded, you can extract the files to a location of your choice.

The .zip file contains the following directories:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `doc` - contains a CHM of the Connector/NET documentation.
- `Samples` - sample code and applications that use the Connector/NET component.
- `mysqlclient` - the source code for the Connector/NET component.
- `testsuite` - the test suite used to verify the operation of the Connector/NET component.

25.2.2.2. Installing Connector/NET on Unix with Mono

There is no installer available for installing the Connector/NET component on your Unix installation. However, the installation is very simple. Before installing, please ensure that you have a working Mono project installation.

Note that you should only install the Connector/NET component on Unix environments where you want to connect to a MySQL server through the Mono project. If you are deploying or developing on a different environment such as Java or Perl then you should use a more appropriate connectivity component. See the Capítulo 25, *Conectores*, or Capítulo 24, *APIs de MySQL*, for more information.

To install Connector/NET on Unix/Mono:

1. Download the `mysql-connector-net-version-noinstall.zip` and extract the contents.
2. Copy the `MySql.Data.dll` file to your Mono project installation folder.
3. You must register the Connector/NET component in the Global Assembly Cache using the `gacutil` command:

```
shell> gacutil /i MySql.Data.dll
```

Once installed, applications that are compiled with the Connector/NET component need no further changes. However, you must ensure that when you compile your applications you include the Connector/NET component using the `-r:MySqlData.dll` command line option.

25.2.2.3. Installing Connector/NET using the Source

Caution: You should read this section only if you are interested in helping us test our new code. If you just want to get Connector/NET up and running on your system, you should use a standard release distribution.

To be able to access the Connector/NET source tree, you must have Subversion installed. Subversion is freely available from http://subversion.tigris.org/.

The most recent development source tree is available from our public Subversion trees at http://dev.mysql.com/tech-resources/sources.html.

To checkout out the Connector/NET sources, change to the directory where you want the copy of the Connector/NET tree to be stored, then use the following command:

```
shell> svn co
http://svn.mysql.com/svnpublic/connector-net
```

A Visual Studio project is included in the source which you can use to build Connector/NET.

25.2.3. Connector/NET Examples

Connector/NET comprises several classes that are used to connect to the database, execute queries and statements, and manage query results.

The following are the major classes of Connector/NET:

- `MySqlCommand`: Represents an SQL statement to execute against a MySQL database.
- `MySqlCommandBuilder`: Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database.
- `MySqlConnection`: Represents an open connection to a MySQL Server database.
- `MySqlDataAdapter`: Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database.
- `MySqlDataReader`: Provides a means of reading a forward-only stream of rows from a MySQL database.
- `MySqlException`: The exception that is thrown when MySQL returns an error.
- `MySqlHelper`: Helper class that makes it easier to work with the provider.
- `MySqlTransaction`: Represents an SQL transaction to be made in a MySQL database.

This section contains basic information and examples for each of the above classes. For a more detailed reference guide please see Sección 25.2.4, “Connector/NET Reference”.

25.2.3.1. MySqlCommand

Represents a SQL statement to execute against a MySQL database. This class cannot be inherited.

`MySqlCommand` features the following methods for executing commands at a MySQL database:

Item	Description
ExecuteReader	Executes commands that return rows.
ExecuteNonQuery	Executes commands such as SQL INSERT, DELETE, and UPDATE statements.
ExecuteScalar	Retrieves a single value (for example, an aggregate value) from a database.

You can reset the `CommandText` property and reuse the `MySqlCommand` object. However, you must close the MySqlDataReader before you can execute a new or previous command.

If a MySqlException is generated by the method executing a `MySqlCommand`, the MySqlConnection remains open. It is the responsibility of the programmer to close the connection.

Note. Prior versions of the provider used the '@' symbol to mark parameters in SQL. This is incompatible with MySQL user variables, so the provider now uses the '?' symbol to locate parameters in SQL. To support older code, you can set 'old syntax=yes' on your connection string. If you do this, please be aware that an exception will not be throw if you fail to define a parameter that you intended to use in your SQL.

Examples

The following example creates a MySqlCommand and a `MySqlConnection`. The `MySqlConnection` is opened and set as the Connection for the `MySqlCommand`. The example then calls ExecuteNonQuery, and closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
  Public Sub InsertRow(myConnectionString As String)
  " If the connection string is null, use a default.
  If myConnectionString = "" Then
    myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
  End If
  Dim myConnection As New MySqlConnection(myConnectionString)
  Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
  Dim myCommand As New MySqlCommand(myInsertQuery)
  myCommand.Connection = myConnection
  myConnection.Open()
  myCommand.ExecuteNonQuery()
  myCommand.Connection.Close()
End Sub
```

C# example:

```
  public void InsertRow(string myConnectionString)
{
  // If the connection string is null, use a default.
  if(myConnectionString == "")
  {
    myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
  }
  MySqlConnection myConnection = new MySqlConnection(myConnectionString);
  string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
  MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
  myCommand.Connection = myConnection;
  myConnection.Open();
  myCommand.ExecuteNonQuery();
  myCommand.Connection.Close();
}
```

Class MySqlCommand Constructor Form 1

Overload methods for MySqlCommand

Initializes a new instance of the MySqlCommand class.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Note. This example shows how to use one of the overloaded versions of the MySqlCommand constructor. For other examples that might be available, see the individual overload topics.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
  Dim myConnection As New MySqlConnection _
      ("Persist Security Info=False;database=test;server=myServer")
  myConnection.Open()
  Dim myTrans As MySqlTransaction = myConnection.BeginTransaction()
  Dim mySelectQuery As String = "SELECT * FROM MyTable"
  Dim myCommand As New MySqlCommand(mySelectQuery, myConnection, myTrans)
  myCommand.CommandTimeout = 20
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
  MySqlConnection myConnection = new MySqlConnection("Persist Security Info=False;
    database=test;server=myServer");
  myConnection.Open();
  MySqlTransaction myTrans = myConnection.BeginTransaction();
  string mySelectQuery = "SELECT * FROM myTable";
  MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection,myTrans);
  myCommand.CommandTimeout = 20;
}
```

C++ example:

```
public:
void CreateMySqlCommand()
{
  MySqlConnection* myConnection = new MySqlConnection(S"Persist Security Info=False;
    database=test;server=myServer");
  myConnection-&gt;Open();
  MySqlTransaction* myTrans = myConnection-&gt;BeginTransaction();
  String* mySelectQuery = S"SELECT * FROM myTable";
  MySqlCommand* myCommand = new MySqlCommand(mySelectQuery, myConnection, myTrans);
  myCommand-&gt;CommandTimeout = 20;
};
```

Initializes a new instance of the MySqlCommand class.

The base constructor initializes all fields to their default values. The following table shows initial property values for an instance of `MySqlCommand`.

Properties	Initial Value
`CommandText`	**empty string ("")**
`CommandTimeout`	**0**
`CommandType`	**CommandType.Text**
`Connection`	**Null**

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandType = CommandType.Text
```

```
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
   MySqlCommand myCommand = new MySqlCommand();
   myCommand.CommandType = CommandType.Text;
}
```

Class MySqlCommand Constructor Form 2

Initializes a new instance of the `MySqlCommand` class with the text of the query.

Parameters: The text of the query.

When an instance of `MySqlCommand` is created, the following read/write properties are set to initial values.

Properties	Initial Value
`CommandText`	**`cmdText`**
`CommandTimeout`	**0**
`CommandType`	**CommandType.Text**
`Connection`	**Null**

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
  Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySqlCommand(sql)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
  string sql = "SELECT * FROM mytable";
  MySqlCommand myCommand = new MySqlCommand(sql);
  myCommand.CommandType = CommandType.Text;
}
```

Class MySqlCommand Constructor Form 3

Initializes a new instance of the `MySqlCommand` class with the text of the query and a `MySqlConnection`.

Parameters: The text of the query.

Parameters: A `MySqlConnection` that represents the connection to an instance of SQL Server.

When an instance of `MySqlCommand` is created, the following read/write properties are set to initial values.

Properties	Initial Value
`CommandText`	**`cmdText`**

CommandTimeout	**0**
CommandType	**CommandType.Text**
Connection	**connection**

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
  Dim conn as new MySqlConnection("server=myServer")
  Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySqlCommand(sql, conn)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
  MySqlConnection conn = new MySqlConnection("server=myserver")
  string sql = "SELECT * FROM mytable";
  MySqlCommand myCommand = new MySqlCommand(sql, conn);
  myCommand.CommandType = CommandType.Text;
}
```

Class MySqlCommand Constructor Form 4

Initializes a new instance of the MySqlCommand class with the text of the query, a MySqlConnection, and the MySqlTransaction.

Parameters: The text of the query.

Parameters: A MySqlConnection that represents the connection to an instance of SQL Server.

Parameters: The MySqlTransaction in which the MySqlCommand executes.

When an instance of MySqlCommand is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	**cmdText**
CommandTimeout	**0**
CommandType	**CommandType.Text**
Connection	**connection**

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
  Dim conn as new MySqlConnection("server=myServer")
```

```
  conn.Open();
  Dim txn as MySqlTransaction = conn.BeginTransaction()
  Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySqlCommand(sql, conn, txn)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
  MySqlConnection conn = new MySqlConnection("server=myserver")
  conn.Open();
  MySqlTransaction txn = conn.BeginTransaction();
  string sql = "SELECT * FROM mytable";
  MySqlCommand myCommand = new MySqlCommand(sql, conn, txn);
  myCommand.CommandType = CommandType.Text;
}
```

ExecuteNonQuery

Executes a SQL statement against the connection and returns the number of rows affected.

Returns: Number of rows affected

You can use ExecuteNonQuery to perform any type of database operation, however any resultsets returned will not be available. Any output parameters used in calling a stored procedure will be populated with data and can be retrieved after execution is complete. For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, the return value is -1.

Examples

The following example creates a MySqlCommand and then executes it using ExecuteNonQuery. The example is passed a string that is a SQL statement (such as UPDATE, INSERT, or DELETE) and a string to use to connect to the data source.

Visual Basic example:

```
   Public Sub CreateMySqlCommand(myExecuteQuery As String, myConnection As MySqlConnection)
     Dim myCommand As New MySqlCommand(myExecuteQuery, myConnection)
     myCommand.Connection.Open()
     myCommand.ExecuteNonQuery()
     myConnection.Close()
   End Sub
```

C# example:

```
   public void CreateMySqlCommand(string myExecuteQuery, MySqlConnection myConnection)
   {
     MySqlCommand myCommand = new MySqlCommand(myExecuteQuery, myConnection);
     myCommand.Connection.Open();
     myCommand.ExecuteNonQuery();
     myConnection.Close();
   }
```

ExecuteReader1

Sends the `CommandText` to the `MySqlConnection`Connection, and builds a `MySqlDataReader` using one of the `CommandBehavior` values.

Parameters: One of the `CommandBehavior` values.

When the `CommandType` property is set to `StoredProcedure`, the `CommandText` property should be set to the name of the stored procedure. The command executes this stored procedure when you call `ExecuteReader`.

The `MySqlDataReader` supports a special mode that enables large binary values to be read efficiently. For more information, see the `SequentialAccess` setting for `CommandBehavior`.

While the `MySqlDataReader` is in use, the associated `MySqlConnection` is busy serving the `MySqlDataReader`. While in this state, no other operations can be performed on the `MySqlConnection` other than closing it. This is the case until the `MySqlDataReader.Close` method of the `MySqlDataReader` is called. If the `MySqlDataReader` is created with `CommandBehavior` set to `CloseConnection`, closing the `MySqlDataReader` closes the connection automatically.

Note. When calling ExecuteReader with the SingleRow behavior, you should be aware that using a `limit` clause in your SQL will cause all rows (up to the limit given) to be retrieved by the client. The `MySqlDataReader.Read` method will still return false after the first row but pulling all rows of data into the client will have a performance impact. If the `limit` clause is not necessary, it should be avoided.

Returns: A `MySqlDataReader` object.

ExecuteReader

Sends the `CommandText` to the `MySqlConnection`Connection and builds a `MySqlDataReader`.

Returns: A `MySqlDataReader` object.

When the `CommandType` property is set to `StoredProcedure`, the `CommandText` property should be set to the name of the stored procedure. The command executes this stored procedure when you call `ExecuteReader`.

While the `MySqlDataReader` is in use, the associated `MySqlConnection` is busy serving the `MySqlDataReader`. While in this state, no other operations can be performed on the `MySqlConnection` other than closing it. This is the case until the `MySqlDataReader.Close` method of the `MySqlDataReader` is called.

Examples

The following example creates a `MySqlCommand`, then executes it by passing a string that is a SQL `SELECT` statement, and a string to use to connect to the data source.

Visual Basic example:

```
Public Sub CreateMySqlDataReader(mySelectQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
    myReader = myCommand.ExecuteReader()
    Try
    While myReader.Read()
        Console.WriteLine(myReader.GetString(0))
    End While
Finally
    myReader.Close
    myConnection.Close
    End Try
End Sub
```

C# example:

```
public void CreateMySqlDataReader(string mySelectQuery, MySqlConnection myConnection)
 {
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    try
    {
      while(myReader.Read())
      {
        Console.WriteLine(myReader.GetString(0));
      }
    }
    finally
    {
      myReader.Close();
      myConnection.Close();
    }
 }
```

Prepare

Creates a prepared version of the command on an instance of MySQL Server.

Prepared statements are only supported on MySQL version 4.1 and higher. Calling prepare while connected to earlier versions of MySQL will succeed but will execute the statement in the same way as unprepared.

Examples

The following example demonstrates the use of the `Prepare` method.

Visual Basic example:

```
 public sub PrepareExample()
   Dim cmd as New MySqlCommand("INSERT INTO mytable VALUES (?val)", myConnection)
   cmd.Parameters.Add( "?val", 10 )
   cmd.Prepare()
   cmd.ExecuteNonQuery()

   cmd.Parameters(0).Value = 20
   cmd.ExecuteNonQuery()
 end sub
```

C# example:

```
 private void PrepareExample()
 {
   MySqlCommand cmd = new MySqlCommand("INSERT INTO mytable VALUES (?val)", myConnection);
   cmd.Parameters.Add( "?val", 10 );
   cmd.Prepare();
   cmd.ExecuteNonQuery();

   cmd.Parameters[0].Value = 20;
   cmd.ExecuteNonQuery();
 }
```

ExecuteScalar

Executes the query, and returns the first column of the first row in the result set returned by the query. Extra columns or rows are ignored.

Returns: The first column of the first row in the result set, or a null reference if the result set is empty

Use the `ExecuteScalar` method to retrieve a single value (for example, an aggregate value) from a database. This requires less code than using the `ExecuteReader` method, and then performing the operations necessary to generate the single value using the data returned by a `MySqlDataReader`

A typical `ExecuteScalar` query can be formatted as in the following C# example:

C# example:

```
cmd.CommandText = "select count(*) from region";
Int32 count = (int32) cmd.ExecuteScalar();
```

Examples

The following example creates a `MySqlCommand` and then executes it using `ExecuteScalar`. The example is passed a string that is a SQL statement that returns an aggregate result, and a string to use to connect to the data source.

Visual Basic example:

```
Public Sub CreateMySqlCommand(myScalarQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(myScalarQuery, myConnection)
    myCommand.Connection.Open()
    myCommand.ExecuteScalar()
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlCommand(string myScalarQuery, MySqlConnection myConnection)
 {
    MySqlCommand myCommand = new MySqlCommand(myScalarQuery, myConnection);
    myCommand.Connection.Open();
    myCommand.ExecuteScalar();
    myConnection.Close();
 }
```

C++ example:

```
public:
    void CreateMySqlCommand(String* myScalarQuery, MySqlConnection* myConnection)
    {
        MySqlCommand* myCommand = new MySqlCommand(myScalarQuery, myConnection);
        myCommand-&gt;Connection-&gt;Open();
        myCommand-&gt;ExecuteScalar();
        myConnection-&gt;Close();
    }
```

CommandText

Gets or sets the SQL statement to execute at the data source.

Value: The SQL statement or stored procedure to execute. The default is an empty string.

When the `CommandType` property is set to `StoredProcedure`, the `CommandText` property should be set to the name of the stored procedure. The user may be required to use escape character syntax if the stored procedure name contains any special characters. The command executes this stored procedure when you call one of the Execute methods.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandText = "SELECT * FROM Mytable ORDER BY id"
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
 {
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandText = "SELECT * FROM mytable ORDER BY id";
    myCommand.CommandType = CommandType.Text;
 }
```

CommandTimeout

Gets or sets the wait time before terminating the attempt to execute a command and generating an error.

Value: The time (in seconds) to wait for the command to execute. The default is 0 seconds.

MySQL currently does not support any method of canceling a pending or executing operation. All commands issues against a MySQL server will execute until completion or exception occurs.

CommandType

Gets or sets a value indicating how the `CommandText` property is to be interpreted.

Value: One of the `System.Data.CommandType` values. The default is `Text`.

When you set the `CommandType` property to `StoredProcedure`, you should set the `CommandText` property to the name of the stored procedure. The command executes this stored procedure when you call one of the Execute methods.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
   MySqlCommand myCommand = new MySqlCommand();
   myCommand.CommandType = CommandType.Text;
}
```

Connection

Gets or sets the `MySqlConnection` used by this instance of the `MySqlCommand`.

Value: The connection to a data source. The default value is a null reference (`Nothing` in Visual Basic).

If you set `Connection` while a transaction is in progress and the `Transaction` property is not null, an `InvalidOperationException` is generated. If the `Transaction` property is not null and the transaction has already been committed or rolled back, `Transaction` is set to null.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim mySelectQuery As String = "SELECT * FROM mytable ORDER BY id"
    Dim myConnectString As String = "Persist Security Info=False;database=test;server=myServer"
    Dim myCommand As New MySqlCommand(mySelectQuery)
    myCommand.Connection = New MySqlConnection(myConnectString)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
 {
    string mySelectQuery = "SELECT * FROM mytable ORDER BY id";
    string myConnectString = "Persist Security Info=False;database=test;server=myServer";
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery);
    myCommand.Connection = new MySqlConnection(myConnectString);
    myCommand.CommandType = CommandType.Text;
 }
```

IsPrepared

Returns true if the statement is prepared.

Parameters

Get the `MySqlParameterCollection`

Value: The parameters of the SQL statement or stored procedure. The default is an empty collection.

Connector/Net does not support unnamed parameters. Every parameter added to the collection must have an associated name.

Examples

The following example creates a `MySqlCommand` and displays its parameters. To accomplish this, the method is passed a `MySqlConnection`, a query string that is a SQL `SELECT` statement, and an array of `MySqlParameter` objects.

Visual Basic example:

```
Public Sub CreateMySqlCommand(myConnection As MySqlConnection, _
mySelectQuery As String, myParamArray() As MySqlParameter)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?age"
    myCommand.UpdatedRowSource = UpdateRowSource.Both
    myCommand.Parameters.Add(myParamArray)
    Dim j As Integer
    For j = 0 To myCommand.Parameters.Count - 1
       myCommand.Parameters.Add(myParamArray(j))
    Next j
    Dim myMessage As String = ""
```

```
    Dim i As Integer
    For i = 0 To myCommand.Parameters.Count - 1
        myMessage += myCommand.Parameters(i).ToString() & ControlChars.Cr
    Next i
    Console.WriteLine(myMessage)
End Sub
```

C# example:

```
public void CreateMySqlCommand(MySqlConnection myConnection, string mySelectQuery,
  MySqlParameter[] myParamArray)
{
   MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
   myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?age";
   myCommand.Parameters.Add(myParamArray);
   for (int j=0; j&lt;myParamArray.Length; j++)
   {
      myCommand.Parameters.Add(myParamArray[j]) ;
   }
   string myMessage = "";
   for (int i = 0; i &lt; myCommand.Parameters.Count; i++)
   {
      myMessage += myCommand.Parameters[i].ToString() + "\n";
   }
   MessageBox.Show(myMessage);
}
```

Transaction

Gets or sets the `MySqlTransaction` within which the `MySqlCommand` executes.

Value: The `MySqlTransaction`. The default value is a null reference (`Nothing` in Visual Basic).

You cannot set the `Transaction` property if it is already set to a specific value, and the command is in the process of executing. If you set the transaction property to a `MySqlTransaction` object that is not connected to the same `MySqlConnection` as the `MySqlCommand` object, an exception will be thrown the next time you attempt to execute a statement.

UpdatedRowSource

Gets or sets how command results are applied to the `DataRow` when used by the `System.Data.Common.DbDataAdapter.Update` method of the `System.Data.Common.DbDataAdapter`.

Value: One of the `UpdateRowSource` values.

The default `System.Data.UpdateRowSource` value is `Both` unless the command is automatically generated (as in the case of the `MySqlCommandBuilder`), in which case the default is `None`.

25.2.3.2. MySqlCommandBuilder

Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database. This class cannot be inherited.

The `MySqlDataAdapter` does not automatically generate the SQL statements required to reconcile changes made to a `System.Data.DataSet`DataSet with the associated instance of MySQL. However, you can create a `MySqlCommandBuilder` object to automatically generate SQL statements for single-table updates if you set the `MySqlDataAdapter.SelectCommand`SelectCommand property of the `MySqlDataAdapter`. Then, any additional SQL statements that you do not set are generated by the `MySqlCommandBuilder`.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.OnRowUpdating`RowUpdating events whenever you set the `DataAdapter` property. You can only associate one `MySqlDataAdapter` or `MySqlCommandBuilder` object with each other at one time.

To generate INSERT, UPDATE, or DELETE statements, the `MySqlCommandBuilder` uses the `SelectCommand` property to retrieve a required set of metadata automatically. If you change the `SelectCommand` after the metadata has is retrieved (for example, after the first update), you should call the `RefreshSchema` method to update the metadata.

The `SelectCommand` must also return at least one primary key or unique column. If none are present, an `InvalidOperation` exception is generated, and the commands are not generated.

The `MySqlCommandBuilder` also uses the `MySqlCommand.Connection`Connection, `MySqlCommand.CommandTimeout`CommandTimeout, and `MySqlCommand.Transaction`Transaction properties referenced by the `SelectCommand`. The user should call `RefreshSchema` if any of these properties are modified, or if the `SelectCommand` itself is replaced. Otherwise the `MySqlDataAdapter.InsertCommand`InsertCommand, `MySqlDataAdapter.UpdateCommand`UpdateCommand, and `MySqlDataAdapter.DeleteCommand`DeleteCommand properties retain their previous values.

If you call `Dispose`, the `MySqlCommandBuilder` is disassociated from the `MySqlDataAdapter`, and the generated commands are no longer used.

Note. Caution must be used when using MySqlCOmmandBuilder on MySql 4.0 systems. With MySql 4.0, database/schema information is not provided to the connector for a query. This means that a query that pulls columns from two identically named tables in two or more different databases will not cause an exception to be thrown but will not work correctly. Even more dangerous is the situation where your select statement references database X but is executed in database Y and both databases have tables with similar layouts. This situation can cause unwanted changes or deletes. This note does not apply to MySQL versions 4.1 and later.

Examples

The following example uses the `MySqlCommand`, along `MySqlDataAdapter` and `MySqlConnection`, to select rows from a data source. The example is passed an initialized `System.Data.DataSet`, a connection string, a query string that is a SQL `SELECT` statement, and a string that is the name of the database table. The example then creates a `MySqlCommandBuilder`.

Visual Basic example:

```
Public Shared Function SelectRows(myConnection As String, mySelectQuery As String, myTableName As String) As
  Dim myConn As New MySqlConnection(myConnection)
  Dim myDataAdapter As New MySqlDataAdapter()
  myDataAdapter.SelectCommand = New MySqlCommand(mySelectQuery, myConn)
  Dim cb As SqlCommandBuilder = New MySqlCommandBuilder(myDataAdapter)
  myConn.Open()
  Dim ds As DataSet = New DataSet
  myDataAdapter.Fill(ds, myTableName)
  ' Code to modify data in DataSet here
  ' Without the MySqlCommandBuilder this line would fail.
  myDataAdapter.Update(ds, myTableName)
  myConn.Close()
End Function 'SelectRows
```

C# example:

```
public static DataSet SelectRows(string myConnection, string mySelectQuery, string myTableName)
{
```

```
MySqlConnection myConn = new MySqlConnection(myConnection);
MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
myDataAdapter.SelectCommand = new MySqlCommand(mySelectQuery, myConn);
MySqlCommandBuilder cb = new MySqlCommandBuilder(myDataAdapter);
myConn.Open();
DataSet ds = new DataSet();
myDataAdapter.Fill(ds, myTableName);
//code to modify data in DataSet here
//Without the MySqlCommandBuilder this line would fail
myDataAdapter.Update(ds, myTableName);
myConn.Close();
return ds;
}
```

Class MySqlCommandBuilder Constructor

Initializes a new instance of the `MySqlCommandBuilder` class.

Class MySqlCommandBuilder Constructor Form 1

Initializes a new instance of the `MySqlCommandBuilder` class and sets the last one wins property.

Parameters: False to generate change protection code. True otherwise.

The `lastOneWins` parameter indicates whether SQL code should be included with the generated DELETE and UPDATE commands that checks the underlying data for changes. If `lastOneWins` is true then this code is not included and data records could be overwritten in a multi-user or multi-threaded environments. Setting `lastOneWins` to false will include this check which will cause a concurrency exception to be thrown if the underlying data record has changed without our knowledge.

Class MySqlCommandBuilder Constructor Form 2

Initializes a new instance of the `MySqlCommandBuilder` class with the associated `MySqlDataAdapter` object.

Parameters: The `MySqlDataAdapter` to use.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.RowUpdating` events that are generated by the `MySqlDataAdapter` specified in this property.

When you create a new instance `MySqlCommandBuilder`, any existing `MySqlCommandBuilder` associated with this `MySqlDataAdapter` is released.

Class MySqlCommandBuilder Constructor Form 3

Initializes a new instance of the `MySqlCommandBuilder` class with the associated `MySqlDataAdapter` object.

Parameters: The `MySqlDataAdapter` to use.

Parameters: False to generate change protection code. True otherwise.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.RowUpdating` events that are generated by the `MySqlDataAdapter` specified in this property.

When you create a new instance `MySqlCommandBuilder`, any existing `MySqlCommandBuilder` associated with this `MySqlDataAdapter` is released.

The `lastOneWins` parameter indicates whether SQL code should be included with the generated DELETE and UPDATE commands that checks the underlying data for changes. If `lastOneWins` is true

then this code is not included and data records could be overwritten in a multi-user or multi-threaded environments. Setting `lastOneWins` to false will include this check which will cause a concurrency exception to be thrown if the underlying data record has changed without our knowledge.

DataAdapter

Gets or sets a `MySqlDataAdapter` object for which SQL statements are automatically generated.

Value: A `MySqlDataAdapter` object.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.RowUpdating` events that are generated by the `MySqlDataAdapter` specified in this property.

When you create a new instance `MySqlCommandBuilder`, any existing `MySqlCommandBuilder` associated with this `MySqlDataAdapter` is released.

QuotePrefix

Gets or sets the beginning character or characters to use when specifying MySQL database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

Value: The beginning character or characters to use. The default value is `.

Database objects in MySQL can contain special characters such as spaces that would make normal SQL strings impossible to correctly parse. Use of the `QuotePrefix` and the `QuoteSuffix` properties allows the `MySqlCommandBuilder` to build SQL commands that handle this situation.

QuoteSuffix

Gets or sets the beginning character or characters to use when specifying MySQL database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

Value: The beginning character or characters to use. The default value is `.

Database objects in MySQL can contain special characters such as spaces that would make normal SQL strings impossible to correctly parse. Use of the `QuotePrefix` and the `QuoteSuffix` properties allows the `MySqlCommandBuilder` to build SQL commands that handle this situation.

DeriveParameters

GetDeleteCommand

Gets the automatically generated `MySqlCommand` object required to perform deletions on the database.

Returns: The `MySqlCommand` object generated to handle delete operations.

An application can use the `GetDeleteCommand` method for informational or troubleshooting purposes because it returns the `MySqlCommand` object to be executed.

You can also use `GetDeleteCommand` as the basis of a modified command. For example, you might call `GetDeleteCommand` and modify the `MySqlCommand.CommandTimeout` value, and then explicitly set that on the `MySqlDataAdapter`.

After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetDeleteCommand` will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls `System.Data.Common.DataAdapter.Update` or `GetDeleteCommand`.

GetInsertCommand

Gets the automatically generated `MySqlCommand` object required to perform insertions on the database.

Returns: The `MySqlCommand` object generated to handle insert operations.

An application can use the `GetInsertCommand` method for informational or troubleshooting purposes because it returns the `MySqlCommand` object to be executed.

You can also use the `GetInsertCommand` as the basis of a modified command. For example, you might call `GetInsertCommand` and modify the `MySqlCommand.CommandTimeout` value, and then explicitly set that on the `MySqlDataAdapter`.

After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetInsertCommand` will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls `System.Data.Common.DataAdapter.Update` or `GetInsertCommand`.

GetUpdateCommand

Gets the automatically generated `MySqlCommand` object required to perform updates on the database.

Returns: The `MySqlCommand` object generated to handle update operations.

An application can use the `GetUpdateCommand` method for informational or troubleshooting purposes because it returns the `MySqlCommand` object to be executed.

You can also use `GetUpdateCommand` as the basis of a modified command. For example, you might call `GetUpdateCommand` and modify the `MySqlCommand.CommandTimeout` value, and then explicitly set that on the `MySqlDataAdapter`.

After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetUpdateCommand` will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls `System.Data.Common.DataAdapter.Update` or `GetUpdateCommand`.

RefreshSchema

Refreshes the database schema information used to generate INSERT, UPDATE, or DELETE statements.

An application should call `RefreshSchema` whenever the `SELECT` statement associated with the `MySqlCommandBuilder` changes.

An application should call `RefreshSchema` whenever the `MySqlDataAdapter.SelectCommand` value of the `MySqlDataAdapter` changes.

25.2.3.3. MySqlConnection

Represents an open connection to a MySQL Server database. This class cannot be inherited.

A `MySqlConnection` object represents a session to a MySQL Server data source. When you create an instance of `MySqlConnection`, all properties are set to their initial values. For a list of these values, see the `MySqlConnection` constructor.

If the `MySqlConnection` goes out of scope, it is not closed. Therefore, you must explicitly close the connection by calling `MySqlConnection.Close` or `MySqlConnection.Dispose`.

Examples

The following example creates a `MySqlCommand` and a `MySqlConnection`. The `MySqlConnection` is opened and set as the `MySqlCommand.Connection` for the `MySqlCommand`. The example then calls `MySqlCommand.ExecuteNonQuery`, and closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Sub InsertRow(myConnectionString As String)
  ' If the connection string is null, use a default.
  If myConnectionString = "" Then
    myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
  End If
  Dim myConnection As New MySqlConnection(myConnectionString)
  Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
  Dim myCommand As New MySqlCommand(myInsertQuery)
  myCommand.Connection = myConnection
  myConnection.Open()
  myCommand.ExecuteNonQuery()
  myCommand.Connection.Close()
End Sub
```

C# example:

```
public void InsertRow(string myConnectionString)
{
  // If the connection string is null, use a default.
  if(myConnectionString == "")
  {
    myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
  }
  MySqlConnection myConnection = new MySqlConnection(myConnectionString);
  string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
  MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
  myCommand.Connection = myConnection;
  myConnection.Open();
  myCommand.ExecuteNonQuery();
  myCommand.Connection.Close();
}
```

Class MySqlConnection Constructor (Default)

Initializes a new instance of the `MySqlConnection` class.

When a new instance of `MySqlConnection` is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the `ConnectionString` property.

Properties	Initial Value
`ConnectionString`	**empty string ("")**
`ConnectionTimeout`	**15**
`Database`	**empty string ("")**
`DataSource`	**empty string ("")**
`ServerVersion`	**empty string ("")**

You can change the value for these properties only by using the `ConnectionString` property.

Examples

Overload methods for MySqlConnection

Initializes a new instance of the `MySqlConnection` class.

Class MySqlConnection Constructor Form 1

Initializes a new instance of the `MySqlConnection` class when given a string containing the connection string.

When a new instance of `MySqlConnection` is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the `ConnectionString` property.

Properties	Initial Value
`ConnectionString`	**empty string ("")**
`ConnectionTimeout`	**15**
`Database`	**empty string ("")**
`DataSource`	**empty string ("")**
`ServerVersion`	**empty string ("")**

You can change the value for these properties only by using the `ConnectionString` property.

Examples

Parameters: The connection properties used to open the MySQL database.

Open

Opens a database connection with the property settings specified by the ConnectionString.

Exception: Cannot open a connection without specifying a data source or server.

Exception: A connection-level error occurred while opening the connection.

The `MySqlConnection` draws an open connection from the connection pool if one is available. Otherwise, it establishes a new connection to an instance of MySQL.

Examples

The following example creates a `MySqlConnection`, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlConnection(string myConnString)
{
  MySqlConnection myConnection = new MySqlConnection(myConnString);
```

```
  myConnection.Open();
  MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
          "\nState: " + myConnection.State.ToString());
  myConnection.Close();
}
```

Database

Gets the name of the current database or the database to be used after a connection is opened.

Returns: The name of the current database or the name of the database to be used after a connection is opened. The default value is an empty string.

The `Database` property does not update dynamically. If you change the current database using a SQL statement, then this property may reflect the wrong value. If you change the current database using the `ChangeDatabase` method, this property is updated to reflect the new database.

Examples

The following example creates a `MySqlConnection` and displays some of its read-only properties.

Visual Basic example:

```
Public Sub CreateMySqlConnection()
  Dim myConnString As String = _
    "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass"
  Dim myConnection As New MySqlConnection( myConnString )
  myConnection.Open()
  MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
    + ControlChars.NewLine + "Database: " + myConnection.Database )
  myConnection.ChangeDatabase( "test2" )
  MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.NewLine + "Database: " + myConnection.Database )
  myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlConnection()
{
  string myConnString =
    "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass";
  MySqlConnection myConnection = new MySqlConnection( myConnString );
  myConnection.Open();
  MessageBox.Show( "Server Version: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
  myConnection.ChangeDatabase( "test2" );
  MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
  myConnection.Close();
}
```

State

Gets the current state of the connection.

Returns: A bitwise combination of the `System.Data.ConnectionState` values. The default is `Closed`.

The allowed state changes are:

- From `Closed` to `Open`, using the `Open` method of the connection object.

- From `Open` to `Closed`, using either the `Close` method or the `Dispose` method of the connection object.

Examples

The following example creates a `MySqlConnection`, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlConnection(string myConnString)
{
  MySqlConnection myConnection = new MySqlConnection(myConnString);
  myConnection.Open();
  MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
          "\nState: " + myConnection.State.ToString());
  myConnection.Close();
}
```

ServerVersion

Gets a string containing the version of the MySQL server to which the client is connected.

Returns: The version of the instance of MySQL.

Exception: The connection is closed.

Examples

The following example creates a `MySqlConnection`, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlConnection(string myConnString)
{
  MySqlConnection myConnection = new MySqlConnection(myConnString);
  myConnection.Open();
  MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
          "\nState: " + myConnection.State.ToString());
  myConnection.Close();
}
```

Close

Closes the connection to the database. This is the preferred method of closing any open connection.

The `Close` method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled.

An application can call `Close` more than one time. No exception is generated.

Examples

The following example creates a `MySqlConnection`, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlConnection(string myConnString)
{
  MySqlConnection myConnection = new MySqlConnection(myConnString);
  myConnection.Open();
  MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
          "\nState: " + myConnection.State.ToString());
  myConnection.Close();
}
```

CreateCommand

Creates and returns a `MySqlCommand` object associated with the `MySqlConnection`.

Returns: A `MySqlCommand` object.

BeginTransaction

Begins a database transaction.

Returns: An object representing the new transaction.

Exception: Parallel transactions are not supported.

This command is equivalent to the MySQL BEGIN TRANSACTION command.

You must explicitly commit or roll back the transaction using the `MySqlTransaction.Commit` or `MySqlTransaction.Rollback` method.

Note. If you do not specify an isolation level, the default isolation level is used. To specify an isolation level with the `BeginTransaction` method, use the overload that takes the `iso` parameter.

Examples

The following example creates a `MySqlConnection` and a `MySqlTransaction`. It also demonstrates how to use the `BeginTransaction`, a `MySqlTransaction.Commit`, and `MySqlTransaction.Rollback` methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
      myCommand.CommandText = "Insert into Test (id, desc) VALUES (100, 'Description')"
      myCommand.ExecuteNonQuery()
      myCommand.CommandText = "Insert into Test (id, desc) VALUES (101, 'Description')"
      myCommand.ExecuteNonQuery()
      myTrans.Commit()
      Console.WriteLine("Both records are written to database.")
    Catch e As Exception
      Try
        myTrans.Rollback()
      Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
          Console.WriteLine("An exception of type " + ex.GetType().ToString() + _
                            " was encountered while attempting to roll back the transaction.")
        End If
      End Try

      Console.WriteLine("An exception of type " + e.GetType().ToString() + _
                        "was encountered while inserting the data.")
      Console.WriteLine("Neither record was written to database.")
    Finally
      myConnection.Close()
    End Try
End Sub
```

C# example:

```
public void RunTransaction(string myConnString)
{
  MySqlConnection myConnection = new MySqlConnection(myConnString);
  myConnection.Open();
  MySqlCommand myCommand = myConnection.CreateCommand();
  MySqlTransaction myTrans;
  // Start a local transaction
  myTrans = myConnection.BeginTransaction();
  // Must assign both transaction object and connection
  // to Command object for a pending local transaction
  myCommand.Connection = myConnection;
  myCommand.Transaction = myTrans;
    try
    {
      myCommand.CommandText = "insert into Test (id, desc) VALUES (100, 'Description')";
      myCommand.ExecuteNonQuery();
      myCommand.CommandText = "insert into Test (id, desc) VALUES (101, 'Description')";
      myCommand.ExecuteNonQuery();
```

```
      myTrans.Commit();
      Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
      try
      {
        myTrans.Rollback();
      }
      catch (SqlException ex)
      {
        if (myTrans.Connection != null)
        {
          Console.WriteLine("An exception of type " + ex.GetType() +
                            " was encountered while attempting to roll back the transaction.");
        }
      }

      Console.WriteLine("An exception of type " + e.GetType() +
                        " was encountered while inserting the data.");
      Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
      myConnection.Close();
    }
}
```

BeginTransaction1

Begins a database transaction with the specified isolation level.

Parameters: The isolation level under which the transaction should run.

Returns: An object representing the new transaction.

Exception: Parallel exceptions are not supported.

This command is equivalent to the MySQL BEGIN TRANSACTION command.

You must explicitly commit or roll back the transaction using the `MySqlTransaction.Commit` or `MySqlTransaction.Rollback` method.

Note. If you do not specify an isolation level, the default isolation level is used. To specify an isolation level with the `BeginTransaction` method, use the overload that takes the `iso` parameter.

Examples

The following example creates a `MySqlConnection` and a `MySqlTransaction`. It also demonstrates how to use the `BeginTransaction`, a `MySqlTransaction.Commit`, and `MySqlTransaction.Rollback` methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
```

```
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
      myCommand.CommandText = "Insert into Test (id, desc) VALUES (100, 'Description')"
      myCommand.ExecuteNonQuery()
      myCommand.CommandText = "Insert into Test (id, desc) VALUES (101, 'Description')"
      myCommand.ExecuteNonQuery()
      myTrans.Commit()
      Console.WriteLine("Both records are written to database.")
    Catch e As Exception
      Try
        myTrans.Rollback()
      Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
          Console.WriteLine("An exception of type " + ex.GetType().ToString() + _
                            " was encountered while attempting to roll back the transaction.")
        End If
      End Try

      Console.WriteLine("An exception of type " + e.GetType().ToString() + _
                      "was encountered while inserting the data.")
      Console.WriteLine("Neither record was written to database.")
    Finally
      myConnection.Close()
    End Try
End Sub
```

C# example:

```
public void RunTransaction(string myConnString)
{
  MySqlConnection myConnection = new MySqlConnection(myConnString);
  myConnection.Open();
  MySqlCommand myCommand = myConnection.CreateCommand();
  MySqlTransaction myTrans;
  // Start a local transaction
  myTrans = myConnection.BeginTransaction();
  // Must assign both transaction object and connection
  // to Command object for a pending local transaction
  myCommand.Connection = myConnection;
  myCommand.Transaction = myTrans;
    try
    {
      myCommand.CommandText = "insert into Test (id, desc) VALUES (100, 'Description')";
      myCommand.ExecuteNonQuery();
      myCommand.CommandText = "insert into Test (id, desc) VALUES (101, 'Description')";
      myCommand.ExecuteNonQuery();
      myTrans.Commit();
      Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
      try
      {
        myTrans.Rollback();
      }
      catch (SqlException ex)
      {
        if (myTrans.Connection != null)
        {
          Console.WriteLine("An exception of type " + ex.GetType() +
                            " was encountered while attempting to roll back the transaction.");
        }
```

```
      }

      Console.WriteLine("An exception of type " + e.GetType() +
                        " was encountered while inserting the data.");
      Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
      myConnection.Close();
    }
}
```

ChangeDatabase

Changes the current database for an open MySqlConnection.

Parameters: The name of the database to use.

The value supplied in the `database` parameter must be a valid database name. The `database` parameter cannot contain a null value, an empty string, or a string with only blank characters.

When you are using connection pooling against MySQL, and you close the connection, it is returned to the connection pool. The next time the connection is retrieved from the pool, the reset connection request executes before the user performs any operations.

Exception: The database name is not valid.

Exception: The connection is not open.

Exception: Cannot change the database.

Examples

The following example creates a `MySqlConnection` and displays some of its read-only properties.

Visual Basic example:

```
Public Sub CreateMySqlConnection()
  Dim myConnString As String = _
    "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass"
  Dim myConnection As New MySqlConnection( myConnString )
  myConnection.Open()
  MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
    + ControlChars.NewLine + "Database: " + myConnection.Database )
  myConnection.ChangeDatabase( "test2" )
  MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.NewLine + "Database: " + myConnection.Database )
  myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlConnection()
{
  string myConnString =
    "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass";
  MySqlConnection myConnection = new MySqlConnection( myConnString );
  myConnection.Open();
  MessageBox.Show( "Server Version: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
  myConnection.ChangeDatabase( "test2" );
  MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
```

```
  myConnection.Close();
}
```

StateChange

Occurs when the state of the connection changes.

The `StateChange` event fires whenever the `State` changes from closed to opened, or from opened to closed. `StateChange` fires immediately after the `MySqlConnection` transitions.

If an event handler throws an exception from within the `StateChange` event, the exception propagates to the caller of the `Open` or `Close` method.

The `StateChange` event is not raised unless you explicitly call `Close` or `Dispose`.

The event handler receives an argument of type `System.Data.StateChangeEventArgs` containing data related to this event. The following `StateChangeEventArgs` properties provide information specific to this event.

Property	Description
`System.Data.StateChangeEventArgs.CurrentState`	Gets the new state of the connection. The connection object will be in the new state already when the event is fired.
`System.Data.StateChangeEventArgs.OriginalState`	Gets the original state of the connection.

InfoMessage

Occurs when MySQL returns warnings as a result of executing a command or query.

ConnectionTimeout

Gets the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Exception: The value set is less than 0.

A value of 0 indicates no limit, and should be avoided in a `MySqlConnection.ConnectionString` because an attempt to connect will wait indefinitely.

Examples

The following example creates a MySqlConnection and sets some of its properties in the connection string.

Visual Basic example:

```
Public Sub CreateSqlConnection()
  Dim myConnection As New MySqlConnection()
  myConnection.ConnectionString = "Persist Security Info=False;Username=user;Password=pass;database=test1;
  myConnection.Open()
End Sub
```

C# example:

```
public void CreateSqlConnection()
{
  MySqlConnection myConnection = new MySqlConnection();
  myConnection.ConnectionString = "Persist Security Info=False;Username=user;Password=pass;database=test1;
  myConnection.Open();
}
```

ConnectionString

Gets or sets the string used to connect to a MySQL Server database.

The `ConnectionString` returned may not be exactly like what was originally set but will be indentical in terms of keyword/value pairs. Security information will not be included unless the Persist Security Info value is set to true.

You can use the `ConnectionString` property to connect to a database. The following example illustrates a typical connection string.

```
"Persist Security Info=False;database=MyDB;server=MySqlServer;user id=myUser;Password=myPass"
```

The `ConnectionString` property can be set only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, except when an error is detected. In this case, none of the properties are updated. `MySqlConnection` properties return only those settings contained in the `ConnectionString`.

To connect to a local machine, specify "localhost" for the server. If you do not specify a server, localhost is assumed.

Resetting the `ConnectionString` on a closed connection resets all connection string values (and related properties) including the password. For example, if you set a connection string that includes "Database= MyDb", and then reset the connection string to "Data Source=myserver;User Id=myUser;Password=myPass", the `MySqlConnection.Database` property is no longer set to MyDb.

The connection string is parsed immediately after being set. If errors in syntax are found when parsing, a runtime exception, such as `ArgumentException`, is generated. Other errors can be found only when an attempt is made to open the connection.

The basic format of a connection string consists of a series of keyword/value pairs separated by semicolons. The equal sign (=) connects each keyword and its value. To include values that contain a semicolon, single-quote character, or double-quote character, the value must be enclosed in double quotes. If the value contains both a semicolon and a double-quote character, the value can be enclosed in single quotes. The single quote is also useful if the value begins with a double-quote character. Conversely, the double quote can be used if the value begins with a single quote. If the value contains both single-quote and double-quote characters, the quote character used to enclose the value must be doubled each time it occurs within the value.

To include preceding or trailing spaces in the string value, the value must be enclosed in either single quotes or double quotes. Any leading or trailing spaces around integer, Boolean, or enumerated values are ignored, even if enclosed in quotes. However, spaces within a string literal keyword or value are preserved. Using .NET Framework version 1.1, single or double quotes may be used within a connection string without using delimiters (for example, Data Source= my'Server or Data Source= my"Server), unless a quote character is the first or last character in the value.

To include an equal sign (=) in a keyword or value, it must be preceded by another equal sign. For example, in the hypothetical connection string

```
"key==word=value"
```

the keyword is "key=word" and the value is "value".

If a specific keyword in a keyword= value pair occurs multiple times in a connection string, the last occurrence listed is used in the value set.

Keywords are not case sensitive.

The following table lists the valid names for keyword values within the `ConnectionString`.

Name	Default	Description
`Connect Timeout`, `Connection Timeout`	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
`Host`, `Server`, `Data Source`, `DataSource`, `Address`, `Addr`, `Network Address`	localhost	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by &. This can be useful where multiple MySQL servers are configured for replication and you are not concerned about the precise server you are connecting to. No attempt is made by the provider to synchronize writes to the database so care should be taken when using this option. In Unix environment with Mono, this can be a fully qualified path to MySQL socket filename. With this configuration, the Unix socket will be used instead of TCP/IP socket. Currently only a single socket name can be given so accessing MySQL in a replicated environment using Unix sockets is not currently supported.
`Ignore Prepare`	true	When true, instructs the provider to ignore any calls to MySqlCommand.Prepare(). This option is provided to prevent issues with corruption of the statements when use with server side prepared statements. If you want to use server-side prepare statements, set this option to false. This option was added in Connector/NET 5.0.3.
`Port`	3306	The port MySQL is using to listen for connections. Specify -1 for this value to use a named pipe connection (Windows only). This value is ignored if Unix socket is used.
`Protocol`	socket	Specifies the type of connection to make to the server.Values can be: socket or tcp for a socket connection pipe for a named pipe connection unix for a Unix socket

		connection memory to use MySQL shared memory
CharSet, Character Set		Specifies the character set that should be used to encode all queries sent to the server. Resultsets are still returned in the character set of the data returned.
Logging	false	When true, various pieces of information is output to any configured TraceListeners.
Allow Batch	true	When true, multiple SQL statements can be sent with one command execution. -Note-Starting with MySQL 4.1.1, batch statements should be separated by the server-defined seperator character. Commands sent to earlier versions of MySQL should be seperated with ';'.
Encrypt	false	For Connector/NET 5.0.3 and later, when true, SSL encryption is used for all data sent between the client and server if the server has a certificate installed. Recognized values are true, false, yes, and no. In versions before 5.0.3, this option had no effect.
Initial Catalog, Database	mysql	The name of the database to use intially
Password, pwd		The password for the MySQL account being used.
Persist Security Info	false	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.
User Id, Username, Uid, User name		The MySQL login account being used.
Shared Memory Name	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to memory.

`Allow Zero Datetime`	false	True to have MySqlDataReader.GetValue() return a MySqlDateTime for date or datetime columns that have illegal values. False will cause a `System.DateTime` object to be returned for legal values and an exception will be thrown for illegal values.
`Convert Zero Datetime`	false	True to have `MySqlDataReader.GetValue()` and `MySqlDataReader.GetDateTime()` return DateTime.MinValue for date or datetime columns that have illegal values.
`Old Syntax`, `OldSyntax`	false	Allows use of '@' symbol as a parameter marker. See `MySqlCommand` for more info. This is for compatibility only. All future code should be written to use the new '?' parameter marker.
`Pipe Name`, `Pipe`	mysql	When set to the name of a named pipe, the `MySqlConnection` will attempt to connect to MySQL on that named pipe.This settings only applies to the Windows platform.

The following table lists the valid names for connection pooling values within the `ConnectionString`. For more information about connection pooling, see Connection Pooling for the MySql Data Provider.

Name	Default	Description
`Connection Lifetime`	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by `Connection Lifetime`. This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
`Max Pool Size`	100	The maximum number of connections allowed in the pool.
`Min Pool Size`	0	The minimum number of connections allowed in the pool.

`Pooling`	true	When `true`, the `MySqlConnection` object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are `true`, `false`, `yes`, and `no`.
`Reset Pooled Connections`, `ResetConnections`, `ResetPooledConnections`	true	Specifies whether a ping and a reset should be sent to the server before a pooled connection is returned. Not resetting will yeild faster connection opens but also will not clear out session items such as temp tables.
`Cache Server Configuration`, `CacheServerConfiguration`, `CacheServerConfig`	false	Specifies whether server variables should be updated when a pooled connection is returned. Turning this one will yeild faster opens but will also not catch any server changes made by other connections.

When setting keyword or connection pooling values that require a Boolean value, you can use 'yes' instead of 'true', and 'no' instead of 'false'.

`Note` The MySql Data Provider uses the native socket protocol to communicate with MySQL. Therefore, it does not support the use of an ODBC data source name (DSN) when connecting to MySQL because it does not add an ODBC layer.

`CAUTION` In this release, the application should use caution when constructing a connection string based on user input (for example when retrieving user ID and password information from a dialog box, and appending it to the connection string). The application should ensure that a user cannot embed extra connection string parameters in these values (for example, entering a password as "validpassword;database=somedb" in an attempt to attach to a different database).

Examples

The following example creates a `MySqlConnection` and sets some of its properties

Visual Basic example:

```
Public Sub CreateConnection()
  Dim myConnection As New MySqlConnection()
  myConnection.ConnectionString = "Persist Security Info=False;database=myDB;server=myHost;Connect Timeout=3
  myConnection.Open()
End Sub 'CreateConnection
```

C# example:

```
public void CreateConnection()
{
  MySqlConnection myConnection = new MySqlConnection();
  myConnection.ConnectionString = "Persist Security Info=False;database=myDB;server=myHost;Connect Timeout=3
  myConnection.Open();
}
```

Examples

The following example creates a `MySqlConnection` in Unix environment with Mono installed. MySQL socket filename used in this example is "/var/lib/mysql/mysql.sock". The actual filename depends on your MySQL configuration.

Visual Basic example:

```
Public Sub CreateConnection()
  Dim myConnection As New MySqlConnection()
  myConnection.ConnectionString = "database=myDB;server=/var/lib/mysql/mysql.sock;user id=myUser; pwd=myP
  myConnection.Open()
End Sub 'CreateConnection
```

C# example:

```
public void CreateConnection()
{
  MySqlConnection myConnection = new MySqlConnection();
  myConnection.ConnectionString = "database=myDB;server=/var/lib/mysql/mysql.sock;user id=myUser; pwd=myP
  myConnection.Open();
}
```

25.2.3.4. MySqlDataAdapter

Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database. This class cannot be inherited.

The `MySQLDataAdapter`, serves as a bridge between a `System.Data.DataSet` and MySQL for retrieving and saving data. The `MySQLDataAdapter` provides this bridge by mapping `DbDataAdapter.Fill`, which changes the data in the `DataSet` to match the data in the data source, and `DbDataAdapter.Update`, which changes the data in the data source to match the data in the `DataSet`, using the appropriate SQL statements against the data source.

When the `MySQLDataAdapter` fills a `DataSet`, it will create the necessary tables and columns for the returned data if they do not already exist. However, primary key information will not be included in the implicitly created schema unless the `System.Data.MissingSchemaAction` property is set to `System.Data.MissingSchemaAction.AddWithKey`. You may also have the `MySQLDataAdapter` create the schema of the `DataSet`, including primary key information, before filling it with data using `System.Data.Common.DbDataAdapter.FillSchema`.

`MySQLDataAdapter` is used in conjunction with `MySqlConnection` and `MySqlCommand` to increase performance when connecting to a MySQL database.

The `MySQLDataAdapter` also includes the `MySqlDataAdapter.SelectCommand`, `MySqlDataAdapter.InsertCommand`, `MySqlDataAdapter.DeleteCommand`, `MySqlDataAdapter.UpdateCommand`, and `DataAdapter.TableMappings` properties to facilitate the loading and updating of data.

When an instance of `MySQLDataAdapter` is created, the read/write properties are set to initial values. For a list of these values, see the `MySQLDataAdapter` constructor.

Note. Please be aware that the `DataColumn` class in .NET 1.0 and 1.1 does not allow columns with type of UInt16, UInt32, or UInt64 to be autoincrement columns. If you plan to use autoincremement columns with MySQL, you should consider using signed integer columns.

Examples

The following example creates a `MySqlCommand` and a `MySqlConnection`. The `MySqlConnection` is opened and set as the `MySqlCommand.Connection` for the `MySqlCommand`. The example then calls `MySqlCommand.ExecuteNonQuery`, and closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Function SelectRows(dataSet As DataSet, connection As String, query As String) As DataSet
    Dim conn As New MySqlConnection(connection)
    Dim adapter As New MySqlDataAdapter()
    adapter.SelectCommand = new MySqlCommand(query, conn)
    adapter.Fill(dataset)
    Return dataset
End Function
```

C# example:

```
public DataSet SelectRows(DataSet dataset,string connection,string query)
{
    MySqlConnection conn = new MySqlConnection(connection);
    MySqlDataAdapter adapter = new MySqlDataAdapter();
    adapter.SelectCommand = new MySqlCommand(query, conn);
    adapter.Fill(dataset);
    return dataset;
}
```

Class MySqlDataAdapter Constructor

Overload methods for MySqlDataAdapter

Initializes a new instance of the MySqlDataAdapter class.

When an instance of `MySqlDataAdapter` is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
`MissingMappingAction`	**`MissingMappingAction.Passthrough`**
`MissingSchemaAction`	**`MissingSchemaAction.Add`**

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a `MySqlDataAdapter` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
    "database=test")
    Dim da As MySqlDataAdapter = New MySqlDataAdapter
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.SelectCommand = New MySqlCommand("SELECT id, name FROM mytable", conn)
    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
                                        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
                                        "WHERE id=?oldId", conn)
```

```
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Or
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlDataAdapter da = new MySqlDataAdapter();
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.SelectCommand = new MySqlCommand("SELECT id, name FROM mytable", conn);
    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
                                        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
                                        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Or
}
```

Class MySqlDataAdapter Constructor Form 1

Initializes a new instance of the `MySqlDataAdapter` class with the specified `MySqlCommand` as the `SelectCommand` property.

Parameters: `MySqlCommand` that is a SQL `SELECT` statement or stored procedure and is set as the `SelectCommand` property of the `MySqlDataAdapter`.

When an instance of `MySqlDataAdapter` is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
`MissingMappingAction`	**`MissingMappingAction.Passthrough`**
`MissingSchemaAction`	**`MissingSchemaAction.Add`**

You can change the value of any of these properties through a separate call to the property.

When `SelectCommand` (or any of the other command properties) is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `SelectCommand` maintains a reference to the previously created `MySqlCommand` object.

Examples

The following example creates a `MySqlDataAdapter` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
    "database=test")
  Dim cmd as new MySqlCommand("SELECT id, name FROM mytable", conn)
    Dim da As MySqlDataAdapter = New MySqlDataAdapter(cmd)
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
                                        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
                                        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Ori
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Origin
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlCommand cmd = new MySqlCommand("SELECT id, name FROM mytable", conn);
    MySqlDataAdapter da = new MySqlDataAdapter(cmd);
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
                                        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
                                        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Ori
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Origin
}
```

Class MySqlDataAdapter Constructor Form 2

Initializes a new instance of the `MySqlDataAdapter` class with a `SelectCommand` and a `MySqlConnection` object.

Parameters: A `String` that is a SQL `SELECT` statement or stored procedure to be used by the `SelectCommand` property of the `MySqlDataAdapter`.

Parameters: A `MySqlConnection` that represents the connection.

This implementation of the `MySqlDataAdapter` opens and closes a `MySqlConnection` if it is not already open. This can be useful in a an application that must call the `DbDataAdapter.Fill` method for two or more `MySqlDataAdapter` objects. If the `MySqlConnection` is already open, you must explicitly call `MySqlConnection.Close` or `MySqlConnection.Dispose` to close it.

When an instance of `MySqlDataAdapter` is created, the following read/write properties are set to the following initial values.

Properties	Initial Value

MissingMappingAction	**MissingMappingAction.Passthrough**
MissingSchemaAction	**MissingSchemaAction.Add**

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a `MySqlDataAdapter` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
    "database=test")
    Dim da As MySqlDataAdapter = New MySqlDataAdapter("SELECT id, name FROM mytable", conn)
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
                                        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
                                        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Ori
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT id, name FROM mytable", conn);
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
                                        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
                                        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Ori
}
```

Class MySqlDataAdapter Constructor Form 3

Initializes a new instance of the `MySqlDataAdapter` class with a `SelectCommand` and a connection string.

Parameters: A `string` that is a SQL `SELECT` statement or stored procedure to be used by the `SelectCommand` property of the `MySqlDataAdapter`.

Parameters: The connection string

When an instance of `MySqlDataAdapter` is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
`MissingMappingAction`	**`MissingMappingAction.Passthrough`**
`MissingSchemaAction`	**`MissingSchemaAction.Add`**

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a `MySqlDataAdapter` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim da As MySqlDataAdapter = New MySqlDataAdapter("SELECT id, name FROM mytable", "Data Source=localhost;d
    Dim conn As MySqlConnection = da.SelectCommand.Connection
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
                                        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
                                        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Ori
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Origin
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT id, name FROM mytable", "Data Source=localhost;database
    MySqlConnection conn = da.SelectCommand.Connection;
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
                                        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
                                        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Ori
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Origin
}
```

DeleteCommand

Gets or sets a SQL statement or stored procedure used to delete records from the data set.

Value: A `MySqlCommand` used during `System.Data.Common.DataAdapter.Update` to delete records in the database that correspond to deleted rows in the `DataSet`.

During `System.Data.Common.DataAdapter.Update`, if this property is not set and primary key information is present in the `DataSet`, the `DeleteCommand` can be generated automatically if you set the `SelectCommand` property and use the `MySqlCommandBuilder`. Then, any additional commands that you do not set are generated by the `MySqlCommandBuilder`. This generation logic requires key column information to be present in the `DataSet`.

When `DeleteCommand` is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `DeleteCommand` maintains a reference to the previously created `MySqlCommand` object.

Examples

The following example creates a `MySqlDataAdapter` and sets the `SelectCommand` and `DeleteCommand` properties. It assumes you have already created a `MySqlConnection` object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter

  Dim da As MySqlDataAdapter = New MySqlDataAdapter()
  Dim cmd As MySqlCommand
  Dim parm As MySqlParameter
  ' Create the SelectCommand.
  cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
  da.SelectCommand = cmd
  ' Create the DeleteCommand.
  cmd = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
  parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
  parm.SourceVersion = DataRowVersion.Original
  da.DeleteCommand = cmd
  Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
  MySqlDataAdapter da = new MySqlDataAdapter();
  MySqlCommand cmd;
  MySqlParameter parm;
  // Create the SelectCommand.
  cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
  da.SelectCommand = cmd;
  // Create the DeleteCommand.
  cmd = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
  parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
  parm.SourceVersion = DataRowVersion.Original;
  da.DeleteCommand = cmd;
  return da;
}
```

InsertCommand

Gets or sets a SQL statement or stored procedure used to insert records into the data set.

Value: A `MySqlCommand` used during `System.Data.Common.DataAdapter.Update` to insert records into the database that correspond to new rows in the `DataSet`.

During `System.Data.Common.DataAdapter.Update`, if this property is not set and primary key information is present in the `DataSet`, the `InsertCommand` can be generated automatically if you set the

`SelectCommand` property and use the `MySqlCommandBuilder`. Then, any additional commands that you do not set are generated by the `MySqlCommandBuilder`. This generation logic requires key column information to be present in the `DataSet`.

When `InsertCommand` is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `InsertCommand` maintains a reference to the previously created `MySqlCommand` object.

Note. If execution of this command returns rows, these rows may be added to the `DataSet` depending on how you set the `MySqlCommand.UpdatedRowSource` property of the `MySqlCommand` object.

Examples

The following example creates a `MySqlDataAdapter` and sets the `SelectCommand` and `InsertCommand` properties. It assumes you have already created a `MySqlConnection` object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter

  Dim da As MySqlDataAdapter = New MySqlDataAdapter()
  Dim cmd As MySqlCommand
  Dim parm As MySqlParameter
  ' Create the SelectCommand.
  cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
  da.SelectCommand = cmd
  ' Create the InsertCommand.
  cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id, ?name)", conn)
  cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
  cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )
  da.InsertCommand = cmd

  Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
  MySqlDataAdapter da = new MySqlDataAdapter();
  MySqlCommand cmd;
  MySqlParameter parm;
  // Create the SelectCommand.
  cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
  da.SelectCommand = cmd;
  // Create the InsertCommand.
  cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,?name)", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );

  da.InsertCommand = cmd;
  return da;
}
```

UpdateCommand

Gets or sets a SQL statement or stored procedure used to updated records in the data source.

Value: A `MySqlCommand` used during `System.Data.Common.DataAdapter.Update` to update records in the database with data from the `DataSet`.

During `System.Data.Common.DataAdapter.Update`, if this property is not set and primary key information is present in the `DataSet`, the `UpdateCommand` can be generated automatically if you set the `SelectCommand` property and use the `MySqlCommandBuilder`. Then, any additional commands that you do not set are generated by the `MySqlCommandBuilder`. This generation logic requires key column information to be present in the `DataSet`.

When `UpdateCommand` is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `UpdateCommand` maintains a reference to the previously created `MySqlCommand` object.

Note. If execution of this command returns rows, these rows may be merged with the DataSet depending on how you set the `MySqlCommand.UpdatedRowSource` property of the `MySqlCommand` object.

Examples

The following example creates a `MySqlDataAdapter` and sets the `SelectCommand` and `UpdateCommand` properties. It assumes you have already created a `MySqlConnection` object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter

  Dim da As MySqlDataAdapter = New MySqlDataAdapter()
  Dim cmd As MySqlCommand
  Dim parm As MySqlParameter
  ' Create the SelectCommand.
  cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
  da.SelectCommand = cmd
  ' Create the UpdateCommand.
  cmd = New MySqlCommand("UPDATE mytable SET id=?id, name=?name WHERE id=?oldId", conn)
  cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
  cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )

  parm = cmd.Parameters.Add("?oldId", MySqlDbType.VarChar, 15, "id")
  parm.SourceVersion = DataRowVersion.Original

  da.UpdateCommand = cmd

  Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
  MySqlDataAdapter da = new MySqlDataAdapter();
  MySqlCommand cmd;
  MySqlParameter parm;
  // Create the SelectCommand.
  cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
  da.SelectCommand = cmd;
  // Create the UpdateCommand.
  cmd = new MySqlCommand("UPDATE mytable SET id=?id, name=?name WHERE id=?oldId", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );

  parm = cmd.Parameters.Add( "?oldId", MySqlDbType.VarChar, 15, "id" );
  parm.SourceVersion = DataRowVersion.Original;
```

```
  da.UpdateCommand = cmd;
  return da;
}
```

SelectCommand

Gets or sets a SQL statement or stored procedure used to select records in the data source.

Value: A `MySqlCommand` used during `System.Data.Common.DbDataAdapter.Fill` to select records from the database for placement in the `DataSet`.

When `SelectCommand` is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `SelectCommand` maintains a reference to the previously created `MySqlCommand` object.

If the `SelectCommand` does not return any rows, no tables are added to the `DataSet`, and no exception is raised.

Examples

The following example creates a `MySqlDataAdapter` and sets the `SelectCommand` and `InsertCommand` properties. It assumes you have already created a `MySqlConnection` object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter

  Dim da As MySqlDataAdapter = New MySqlDataAdapter()
  Dim cmd As MySqlCommand
  Dim parm As MySqlParameter
  ' Create the SelectCommand.
  cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
  da.SelectCommand = cmd
  ' Create the InsertCommand.
  cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id, ?name)", conn)
  cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
  cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )
  da.InsertCommand = cmd

  Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
  MySqlDataAdapter da = new MySqlDataAdapter();
  MySqlCommand cmd;
  MySqlParameter parm;
  // Create the SelectCommand.
  cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
  da.SelectCommand = cmd;
  // Create the InsertCommand.
  cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,?name)", conn);
  cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );
  cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );

  da.InsertCommand = cmd;
  return da;
}
```

25.2.3.5. MySqlDataReader

To create a `MySQLDataReader`, you must call the `MySqlCommand.ExecuteReader` method of the `MySqlCommand` object, rather than directly using a constructor.

While the `MySqlDataReader` is in use, the associated `MySqlConnection` is busy serving the `MySqlDataReader`, and no other operations can be performed on the `MySqlConnection` other than closing it. This is the case until the `MySqlDataReader.Close` method of the `MySqlDataReader` is called.

`MySqlDataReader.IsClosed` and `MySqlDataReader.RecordsAffected` are the only properties that you can call after the `MySqlDataReader` is closed. Though the `RecordsAffected` property may be accessed at any time while the `MySqlDataReader` exists, always call `Close` before returning the value of `RecordsAffected` to ensure an accurate return value.

For optimal performance, `MySqlDataReader` avoids creating unnecessary objects or making unnecessary copies of data. As a result, multiple calls to methods such as `MySqlDataReader.GetValue` return a reference to the same object. Use caution if you are modifying the underlying value of the objects returned by methods such as `GetValue`.

Examples

The following example creates a `MySqlConnection`, a `MySqlCommand`, and a `MySqlDataReader`. The example reads through the data, writing it out to the console. Finally, the example closes the `MySqlDataReader`, then the `MySqlConnection`.

Visual Basic example:

```
Public Sub ReadMyData(myConnString As String)
    Dim mySelectQuery As String = "SELECT OrderID, CustomerID FROM Orders"
    Dim myConnection As New MySqlConnection(myConnString)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
    myReader = myCommand.ExecuteReader()
    ' Always call Read before accessing data.
    While myReader.Read()
        Console.WriteLine((myReader.GetInt32(0) & ", " & myReader.GetString(1)))
    End While
    ' always call Close when done reading.
    myReader.Close()
    ' Close the connection when done with it.
    myConnection.Close()
End Sub 'ReadMyData
```

C# example:

```
public void ReadMyData(string myConnString) {
    string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnection);
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    // Always call Read before accessing data.
    while (myReader.Read()) {
       Console.WriteLine(myReader.GetInt32(0) + ", " + myReader.GetString(1));
    }
    // always call Close when done reading.
    myReader.Close();
    // Close the connection when done with it.
```

```
    myConnection.Close();
}
```

GetBytes

`GetBytes` returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if `GetBytes` has already been used to obtain bytes from the field. This may be the case, for example, if the `MySqlDataReader` is reading a large data structure into a buffer. For more information, see the `SequentialAccess` setting for `MySqlCommand.CommandBehavior`.

If you pass a buffer that is a null reference (`Nothing` in Visual Basic), `GetBytes` returns the length of the field in bytes.

No conversions are performed; therefore the data retrieved must already be a byte array.

GetTimeSpan

Gets the value of the specified column as a `TimeSpan` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetDateTime

Gets the value of the specified column as a `System.DateTime` object.

Note. MySql allows date columns to contain the value '0000-00-00' and datetime columns to contain the value '0000-00-00 00:00:00'. The DateTime structure cannot contain or represent these values. To read a datetime value from a column that might contain zero values, use `GetMySqlDateTime`. The behavior of reading a zero datetime column using this method is defined by the `ZeroDateTimeBehavior` connection string option. For more information on this option, please refer to `MySqlConnection.ConnectionString`.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetMySqlDateTime

Gets the value of the specified column as a `MySql.Data.Types.MySqlDateTime` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetString

Gets the value of the specified column as a `String` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetDecimal

Gets the value of the specified column as a `Decimal` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetDouble

Gets the value of the specified column as a double-precision floating point number.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetFloat

Gets the value of the specified column as a single-precision floating point number.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetGiud

Gets the value of the specified column as a globally-unique identifier (GUID).

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetInt16

Gets the value of the specified column as a 16-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetInt32

Gets the value of the specified column as a 32-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetInt64

Gets the value of the specified column as a 64-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetUInt16

Gets the value of the specified column as a 16-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetUInt32

Gets the value of the specified column as a 32-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

GetUInt64

Gets the value of the specified column as a 64-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

25.2.3.6. MySqlException

This class is created whenever the MySql Data Provider encounters an error generated from the server.

Any open connections are not automatically closed when an exception is thrown. If the client application determines that the exception is fatal, it should close any open `MySqlDataReader` objects or `MySqlConnection` objects.

Examples

The following example generates a `MySqlException` due to a missing server, and then displays the exception.

Visual Basic example:

```
Public Sub ShowException()
     Dim mySelectQuery As String = "SELECT column1 FROM table1"
     Dim myConnection As New MySqlConnection ("Data Source=localhost;Database=Sample;")
     Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
     Try
         myCommand.Connection.Open()
     Catch e As MySqlException
    MessageBox.Show( e.Message )
     End Try
 End Sub
```

C# example:

```
public void ShowException()
{
   string mySelectQuery = "SELECT column1 FROM table1";
   MySqlConnection myConnection =
      new MySqlConnection("Data Source=localhost;Database=Sample;");
   MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnection);
   try
   {
      myCommand.Connection.Open();
   }
   catch (MySqlException e)
   {
```

```
    MessageBox.Show( e.Message );
   }
}
```

25.2.3.7. MySqlParameter

Parameter names are not case sensitive.

Examples

The following example creates multiple instances of `MySqlParameter` through the `MySqlParameterCollection` collection within the `MySqlDataAdapter`. These parameters are used to select data from the data source and place the data in the `DataSet`. This example assumes that a `DataSet` and a `MySqlDataAdapter` have already been created with the appropriate schema, commands, and connection.

Visual Basic example:

```
Public Sub AddSqlParameters()
    ' ...
    ' create myDataSet and myDataAdapter
    ' ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters"
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239

    myDataAdapter.Fill(myDataSet)
End Sub 'AddSqlParameters
```

C# example:

```
public void AddSqlParameters()
{
// ...
// create myDataSet and myDataAdapter
// ...
  myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters";
  myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239;
  myDataAdapter.Fill(myDataSet);
}
```

25.2.3.8. MySqlParameterCollection

The number of the parameters in the collection must be equal to the number of parameter placeholders within the command text, or an exception will be generated.

Examples

The following example creates multiple instances of `MySqlParameter` through the `MySqlParameterCollection` collection within the `MySqlDataAdapter`. These parameters are used to select data within the data source and place the data in the `DataSet`. This code assumes that a `DataSet` and a `MySqlDataAdapter` have already been created with the appropriate schema, commands, and connection.

Visual Basic example:

```
Public Sub AddParameters()
    ' ...
    ' create myDataSet and myDataAdapter
```

```
    ' ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters"
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239

    myDataAdapter.Fill(myDataSet)
End Sub 'AddSqlParameters
```

C# example:

```
public void AddSqlParameters()
{
// ...
// create myDataSet and myDataAdapter
// ...
  myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters";
  myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239;
  myDataAdapter.Fill(myDataSet);
}
```

25.2.3.9. MySqlTransaction

Represents a SQL transaction to be made in a MySQL database. This class cannot be inherited.

The application creates a `MySqlTransaction` object by calling `MySqlConnection.BeginTransaction` on the `MySqlConnection` object. All subsequent operations associated with the transaction (for example, committing or aborting the transaction), are performed on the `MySqlTransaction` object.

Examples

The following example creates a `MySqlConnection` and a `MySqlTransaction`. It also demonstrates how to use the `MySqlConnection.BeginTransaction`, `MySqlTransaction.Commit`, and `MySqlTransaction.Rollback` methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')"
      myCommand.ExecuteNonQuery()
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')"
      myCommand.ExecuteNonQuery()
      myTrans.Commit()
      Console.WriteLine("Both records are written to database.")
    Catch e As Exception
      Try
        myTrans.Rollback()
      Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
```

```
          Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                            " was encountered while attempting to roll back the transaction.")
        End If
      End Try

      Console.WriteLine("An exception of type " & e.GetType().ToString() & _
                        "was encountered while inserting the data.")
      Console.WriteLine("Neither record was written to database.")
    Finally
      myConnection.Close()
    End Try
End Sub 'RunTransaction
```

C# example:

```
public void RunTransaction(string myConnString)
 {
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description'
      myCommand.ExecuteNonQuery();
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description'
      myCommand.ExecuteNonQuery();
      myTrans.Commit();
      Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
      try
      {
        myTrans.Rollback();
      }
      catch (MySqlException ex)
      {
        if (myTrans.Connection != null)
        {
          Console.WriteLine("An exception of type " + ex.GetType() +
                            " was encountered while attempting to roll back the transaction.");
        }
      }

      Console.WriteLine("An exception of type " + e.GetType() +
                        " was encountered while inserting the data.");
      Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
      myConnection.Close();
    }
}
```

Rollback

Rolls back a transaction from a pending state.

The Rollback method is equivalent to the MySQL statement ROLLBACK. The transaction can only be rolled back from a pending state (after BeginTransaction has been called, but before Commit is called).

Examples

The following example creates `MySqlConnection` and a `MySqlTransaction`. It also demonstrates how to use the `MySqlConnection.BeginTransaction`, `Commit`, and `Rollback` methods.

Visual Basic example:

```
Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')"
      myCommand.ExecuteNonQuery()
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')"
      myCommand.ExecuteNonQuery()
      myTrans.Commit()
      Console.WriteLine("Success.")
    Catch e As Exception
      Try
        myTrans.Rollback()
      Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
          Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                            " was encountered while attempting to roll back the transaction.")
        End If
      End Try

      Console.WriteLine("An exception of type " & e.GetType().ToString() & _
                        "was encountered while inserting the data.")
      Console.WriteLine("Neither record was written to database.")
    Finally
      myConnection.Close()
    End Try
End Sub
```

C# example:

```
public void RunSqlTransaction(string myConnString)
 {
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
```

```
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')";
      myCommand.ExecuteNonQuery();
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')";
      myCommand.ExecuteNonQuery();
      myTrans.Commit();
      Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
      try
      {
        myTrans.Rollback();
      }
      catch (MySqlException ex)
      {
        if (myTrans.Connection != null)
        {
          Console.WriteLine("An exception of type " + ex.GetType() +
                            " was encountered while attempting to roll back the transaction.");
        }
      }

      Console.WriteLine("An exception of type " + e.GetType() +
                        " was encountered while inserting the data.");
      Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
      myConnection.Close();
    }
}
```

Commit

Commits the database transaction.

The `Commit` method is equivalent to the MySQL SQL statement COMMIT.

Examples

The following example creates `MySqlConnection` and a `MySqlTransaction`. It also demonstrates how to use the `MySqlConnection.BeginTransaction`, `Commit`, and `Rollback` methods.

Visual Basic example:

```
Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')"
      myCommand.ExecuteNonQuery()
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')"
      myCommand.ExecuteNonQuery()
```

```
      myTrans.Commit()
      Console.WriteLine("Success.")
    Catch e As Exception
      Try
        myTrans.Rollback()
      Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
          Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                            " was encountered while attempting to roll back the transaction.")
        End If
      End Try

      Console.WriteLine("An exception of type " & e.GetType().ToString() & _
                        "was encountered while inserting the data.")
      Console.WriteLine("Neither record was written to database.")
    Finally
      myConnection.Close()
    End Try
End Sub
```

C# example:

```
public void RunSqlTransaction(string myConnString)
 {
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')";
      myCommand.ExecuteNonQuery();
      myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')";
      myCommand.ExecuteNonQuery();
      myTrans.Commit();
      Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
      try
      {
        myTrans.Rollback();
      }
      catch (MySqlException ex)
      {
        if (myTrans.Connection != null)
        {
          Console.WriteLine("An exception of type " + ex.GetType() +
                            " was encountered while attempting to roll back the transaction.");
        }
      }

      Console.WriteLine("An exception of type " + e.GetType() +
                        " was encountered while inserting the data.");
      Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
      myConnection.Close();
    }
}
```

25.2.4. Connector/NET Reference

This section of the manual contains a complete reference to the Connector/NET ADO.NET component, automatically generated from the embedded documentation.

25.2.4.1. MySql.Data.MySqlClient

Namespace hierarchy

Classes

Class	Description
MySqlCommand	
MySqlCommandBuilder	
MySqlConnection	
MySqlDataAdapter	
MySqlDataReader	Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.
MySqlError	Collection of error codes that can be returned by the server
MySqlException	The exception that is thrown when MySQL returns an error. This class cannot be inherited.
MySqlHelper	Helper class that makes it easier to work with the provider.
MySqlInfoMessageEventArgs	Provides data for the InfoMessage event. This class cannot be inherited.
MySqlParameter	Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSetcolumns. This class cannot be inherited.
MySqlParameterCollection	Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.
MySqlRowUpdatedEventArgs	Provides data for the RowUpdated event. This class cannot be inherited.
MySqlRowUpdatingEventArgs	Provides data for the RowUpdating event. This class cannot be inherited.
MySqlTransaction	

Delegates

Delegate	Description
MySqlInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a MySqlConnection.
MySqlRowUpdatedEventHandler	Represents the method that will handle the RowUpdatedevent of a MySqlDataAdapter .

MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter .

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL specific data type of a field, property, for use in a MySqlParameter .
MySqlErrorCode	

MySql.Data.MySqlClientHierarchy

See Also

MySql.Data.MySqlClient Namespace

MySqlCommand Class

For a list of all members of this type, see MySqlCommand Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommand_
  Inherits Component_
  Implements IDbCommand, ICloneable
```

Syntax: C#

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlCommand Members , MySql.Data.MySqlClient Namespace

MySqlCommand Members

MySqlCommand overview

Public Instance Constructors

MySqlCommand	Overloaded. Initializes a new instance of the MySqlCommand class.

Public Instance Properties

CommandText	

CommandTimeout	
CommandType	
Connection	
Container(inherited from Component)	Gets the IContainerthat contains the Component.
IsPrepared	
Parameters	
Site(inherited from Component)	Gets or sets the ISiteof the Component.
Transaction	
UpdatedRowSource	

Public Instance Methods

Cancel	Attempts to cancel the execution of a MySqlCommand. This operation is not supported.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
CreateParameter	Creates a new instance of a MySqlParameter object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
ExecuteNonQuery	
ExecuteReader	Overloaded.
ExecuteScalar	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Prepare	
ToString(inherited from Component)	Returns a Stringcontaining the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposedevent on the component.

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand Constructor

Initializes a new instance of the MySqlCommand class.

Overload List

Initializes a new instance of the MySqlCommand class.

- public MySqlCommand();
- public MySqlCommand(string);
- public MySqlCommand(string,MySqlConnection);
- public MySqlCommand(string,MySqlConnection,MySqlTransaction);

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand Constructor ()

Initializes a new instance of the MySqlCommand class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommand();
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace , MySqlCommand Constructor Overload List

MySqlCommand Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal cmdText As String _
)
```

Syntax: C#

```
public MySqlCommand(
stringcmdText
);
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace , MySqlCommand Constructor Overload List

MySqlCommand Constructor (String, MySqlConnection)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal cmdText As String, _
   ByVal connection As MySqlConnection _
)
```

Syntax: C#

```
public MySqlCommand(
stringcmdText,
MySqlConnectionconnection
);
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace , MySqlCommand Constructor Overload List

MySqlConnection Class

For a list of all members of this type, see MySqlConnection Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_
  Inherits Component_
  Implements IDbConnection, ICloneable
```

Syntax: C#

```
public sealed class MySqlConnection : Component, IDbConnection, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlConnection Members , MySql.Data.MySqlClient Namespace

MySqlConnection Members

MySqlConnection overview

Public Instance Constructors

MySqlConnection	Overloaded. Initializes a new instance of the MySqlConnection class.

Public Instance Properties

ConnectionString	
ConnectionTimeout	
Container(inherited from Component)	Gets the IContainerthat contains the Component.
Database	
DataSource	Gets the name of the MySQL server to which to connect.

ServerThread	Returns the id of the server thread this connection is executing on
ServerVersion	
Site(inherited from Component)	Gets or sets the ISiteof the Component.
State	
UseCompression	Indicates if this connection should use compression when communicating with the server.

Public Instance Methods

BeginTransaction	Overloaded.
ChangeDatabase	
Close	
CreateCommand	
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Open	
Ping	Ping
ToString(inherited from Component)	Returns a Stringcontaining the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposedevent on the component.
InfoMessage	
StateChange	

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection Constructor

Initializes a new instance of the MySqlConnection class.

Overload List

Initializes a new instance of the MySqlConnection class.

- public MySqlConnection();
- public MySqlConnection(string);

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection Constructor ()

Initializes a new instance of the MySqlConnection class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlConnection();
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace , MySqlConnection Constructor Overload List

MySqlConnection Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal connectionString As String _
)
```

Syntax: C#

```
public MySqlConnection(
stringconnectionString
);
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace , MySqlConnection Constructor Overload List

ConnectionString Property

Syntax: Visual Basic

```
NotOverridable Public Property ConnectionString As String _
_
  Implements IDbConnection.ConnectionString
```

Syntax: C#

```
public string ConnectionString {get; set;}
```

Implements

IDbConnection.ConnectionString

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

ConnectionTimeout Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer _
_
  Implements IDbConnection.ConnectionTimeout
```

Syntax: C#

```
public int ConnectionTimeout {get;}
```

Implements

IDbConnection.ConnectionTimeout

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

Database Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Database As String _
_
  Implements IDbConnection.Database
```

Syntax: C#

```
public string Database {get;}
```

Implements

IDbConnection.Database

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

DataSource Property

Gets the name of the MySQL server to which to connect.

Syntax: Visual Basic

```
Public ReadOnly Property DataSource As String
```

Syntax: C#

```
public string DataSource {get;}
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

ServerThread Property

Returns the id of the server thread this connection is executing on

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

ServerVersion Property

Syntax: Visual Basic

```
Public ReadOnly Property ServerVersion As String
```

Syntax: C#

```
public string ServerVersion {get;}
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

State Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property State As ConnectionState _
_
  Implements IDbConnection.State
```

Syntax: C#

```
public System.Data.ConnectionState State {get;}
```

Implements

IDbConnection.State

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

UseCompression Property

Indicates if this connection should use compression when communicating with the server.

Syntax: Visual Basic

```
Public ReadOnly Property UseCompression As Boolean
```

Syntax: C#

```
public bool UseCompression {get;}
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

BeginTransaction Method

Overload List

- public MySqlTransaction BeginTransaction();
- public MySqlTransaction BeginTransaction(IsolationLevel);

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection.BeginTransaction Method ()

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction() As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction BeginTransaction();
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace , MySqlConnection.BeginTransaction Overload List

MySqlTransaction Class

For a list of all members of this type, see MySqlTransaction Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlTransaction_
  Implements IDbTransaction, IDisposable
```

Syntax: C#

```
public sealed class MySqlTransaction : IDbTransaction, IDisposable
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlTransaction Members , MySql.Data.MySqlClient Namespace

MySqlTransaction Members

MySqlTransaction overview

Public Instance Properties

Connection	Gets the MySqlConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevelfor this transaction.

Public Instance Methods

Commit	
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
Rollback	
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

MySqlTransaction Class , MySql.Data.MySqlClient Namespace

Connection Property

Gets the MySqlConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax: Visual Basic

```
Public ReadOnly Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get;}
```

Property Value

The MySqlConnection object associated with this transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by BeginTransaction .

See Also

MySqlTransaction Class , MySql.Data.MySqlClient Namespace

IsolationLevel Property

Specifies the IsolationLevelfor this transaction.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsolationLevel As IsolationLevel _
_
  Implements IDbTransaction.IsolationLevel
```

Syntax: C#

```
public System.Data.IsolationLevel IsolationLevel {get;}
```

Property Value

The IsolationLevel for this transaction. The default is ReadCommitted.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

See Also

MySqlTransaction Class , MySql.Data.MySqlClient Namespace

MySqlTransaction.Commit Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _
_
  Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

MySqlTransaction Class , MySql.Data.MySqlClient Namespace

MySqlTransaction.Rollback Method

Syntax: Visual Basic

```
NotOverridable Public Sub Rollback() _
_
  Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

MySqlTransaction Class , MySql.Data.MySqlClient Namespace

MySqlConnection.BeginTransaction Method (IsolationLevel)

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction( _
   ByVal iso As IsolationLevel _
) As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction BeginTransaction(
IsolationLeveliso
);
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace , MySqlConnection.BeginTransaction Overload List

MySqlConnection.ChangeDatabase Method

Syntax: Visual Basic

```
NotOverridable Public Sub ChangeDatabase( _
   ByVal databaseName As String _
) _
_
  Implements IDbConnection.ChangeDatabase
```

Syntax: C#

```
public void ChangeDatabase(
stringdatabaseName
);
```

Implements

IDbConnection.ChangeDatabase

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection.Close Method

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _
_
  Implements IDbConnection.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDbConnection.Close

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection.CreateCommand Method

Syntax: Visual Basic

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection.Open Method

Syntax: Visual Basic

```
NotOverridable Public Sub Open() _
_
  Implements IDbConnection.Open
```

Syntax: C#

```
public void Open();
```

Implements

IDbConnection.Open

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection.Ping Method

Ping

Syntax: Visual Basic

```
Public Function Ping() As Boolean
```

Syntax: C#

```
public bool Ping();
```

Return Value

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlConnection.InfoMessage Event

Syntax: Visual Basic

```
Public Event InfoMessage As MySqlInfoMessageEventHandler
```

Syntax: C#

```
public event MySqlInfoMessageEventHandler InfoMessage;
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventHandler Delegate

Represents the method that will handle the InfoMessage event of a MySqlConnection .

Syntax: Visual Basic

```
Public Delegate Sub MySqlInfoMessageEventHandler( _
   ByVal sender As Object, _
   ByVal args As MySqlInfoMessageEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlInfoMessageEventHandler(
objectsender,
MySqlInfoMessageEventArgsargs
);
```

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventArgs Class

Provides data for the InfoMessage event. This class cannot be inherited.

For a list of all members of this type, see MySqlInfoMessageEventArgs Members .

Syntax: Visual Basic

```
Public Class MySqlInfoMessageEventArgs_
  Inherits EventArgs
```

Syntax: C#

```
public class MySqlInfoMessageEventArgs : EventArgs
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlInfoMessageEventArgs Members , MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventArgs Members

MySqlInfoMessageEventArgs overview

Public Instance Constructors

MySqlInfoMessageEventArgs Constructor	Initializes a new instance of the MySqlInfoMessageEventArgs class.

Public Instance Fields

errors	

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Objectto attempt to free resources and perform other cleanup operations before the Objectis reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlInfoMessageEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventArgs Constructor

Initializes a new instance of the MySqlInfoMessageEventArgs class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlInfoMessageEventArgs();
```

See Also

MySqlInfoMessageEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventArgs.errors Field

Syntax: Visual Basic

```
Public errors As MySqlError()
```

Syntax: C#

```
public MySqlError[] errors;
```

See Also

MySqlInfoMessageEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlError Class

Collection of error codes that can be returned by the server

For a list of all members of this type, see MySqlError Members .

Syntax: Visual Basic

```
Public Class MySqlError
```

Syntax: C#

```
public class MySqlError
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlError Members , MySql.Data.MySqlClient Namespace

MySqlError Members

MySqlError overview

Public Instance Constructors

MySqlError Constructor	

Public Instance Properties

Code	Error code
Level	Error level
Message	Error message

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.

GetType(inherited from Object)	Gets the Typeof the current instance.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Objectto attempt to free resources and perform other cleanup operations before the Objectis reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlError Class , MySql.Data.MySqlClient Namespace

MySqlError Constructor

Syntax: Visual Basic

```
Public Sub New( _
   ByVal level As String, _
   ByVal code As Integer, _
   ByVal message As String _
)
```

Syntax: C#

```
public MySqlError(
stringlevel,
intcode,
stringmessage
);
```

Parameters

- `level`:
- `code`:
- `message`:

See Also

MySqlError Class , MySql.Data.MySqlClient Namespace

Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```

See Also

MySqlError Class , MySql.Data.MySqlClient Namespace

Level Property

Error level

Syntax: Visual Basic

```
Public ReadOnly Property Level As String
```

Syntax: C#

```
public string Level {get;}
```

See Also

MySqlError Class , MySql.Data.MySqlClient Namespace

Message Property

Error message

Syntax: Visual Basic

```
Public ReadOnly Property Message As String
```

Syntax: C#

```
public string Message {get;}
```

See Also

MySqlError Class , MySql.Data.MySqlClient Namespace

MySqlConnection.StateChange Event

Syntax: Visual Basic

```
Public Event StateChange As StateChangeEventHandler
```

Syntax: C#

```
public event StateChangeEventHandler StateChange;
```

See Also

MySqlConnection Class , MySql.Data.MySqlClient Namespace

MySqlCommand Constructor (String, MySqlConnection, MySqlTransaction)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal cmdText As String, _
   ByVal connection As MySqlConnection, _
   ByVal transaction As MySqlTransaction _
)
```

Syntax: C#

```
public MySqlCommand(
stringcmdText,
MySqlConnectionconnection,
MySqlTransactiontransaction
);
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace , MySqlCommand Constructor Overload List

CommandText Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandText As String _
_
  Implements IDbCommand.CommandText
```

Syntax: C#

```
public string CommandText {get; set;}
```

Implements

IDbCommand.CommandText

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

CommandTimeout Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandTimeout As Integer _
_
  Implements IDbCommand.CommandTimeout
```

Syntax: C#

```
public int CommandTimeout {get; set;}
```

Implements

IDbCommand.CommandTimeout

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

CommandType Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandType As CommandType _
_
  Implements IDbCommand.CommandType
```

Syntax: C#

```
public System.Data.CommandType CommandType {get; set;}
```

Implements

IDbCommand.CommandType

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

Connection Property

Syntax: Visual Basic

```
Public Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get; set;}
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

IsPrepared Property

Syntax: Visual Basic

```
Public ReadOnly Property IsPrepared As Boolean
```

Syntax: C#

```
public bool IsPrepared {get;}
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

Parameters Property

Syntax: Visual Basic

```
Public ReadOnly Property Parameters As MySqlParameterCollection
```

Syntax: C#

```
public MySqlParameterCollection Parameters {get;}
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection Class

Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

For a list of all members of this type, see MySqlParameterCollection Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameterCollection_
  Inherits MarshalByRefObject_
  Implements IDataParameterCollection, IList, ICollection, IEnumerable
```

Syntax: C#

```
public sealed class MySqlParameterCollection : MarshalByRefObject, IDataParameterCollection, IList, IColle
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlParameterCollection Members , MySql.Data.MySqlClient Namespace

MySqlParameterCollection Members

MySqlParameterCollection overview

Public Instance Constructors

MySqlParameterCollection Constructor	Initializes a new instance of the MySqlParameterCollection class.

Public Instance Properties

Count	Gets the number of MySqlParameter objects in the collection.
Item	Overloaded. Gets the MySqlParameter with a specified attribute. In C#, this property is the indexer for the MySqlParameterCollection class.

Public Instance Methods

Add	Overloaded. Adds the specified MySqlParameter object to the MySqlParameterCollection .
Clear	Removes all items from the collection.
Contains	Overloaded. Gets a value indicating whether a MySqlParameter exists in the collection.
CopyTo	Copies MySqlParameter objects from the MySqlParameterCollection to the specified array.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
IndexOf	Overloaded. Gets the location of a MySqlParameter in the collection.

InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Insert	Inserts a MySqlParameter into the collection at the specified index.
Remove	Removes the specified MySqlParameter from the collection.
RemoveAt	Overloaded. Removes the specified MySqlParameter from the collection.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection Constructor

Initializes a new instance of the MySqlParameterCollection class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlParameterCollection();
```

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

Count Property

Gets the number of MySqlParameter objects in the collection.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Count As Integer _
_
  Implements ICollection.Count
```

Syntax: C#

```
public int Count {get;}
```

Implements

ICollection.Count

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

Item Property

Gets the MySqlParameter with a specified attribute. In C#, this property is the indexer for the MySqlParameterCollection class.

Overload List

Gets the MySqlParameter at the specified index.

- public MySqlParameter this[int] {get; set;}

Gets the MySqlParameter with the specified name.

- public MySqlParameter this[string] {get; set;}

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameter Class

Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSetcolumns. This class cannot be inherited.

For a list of all members of this type, see MySqlParameter Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameter_
  Inherits MarshalByRefObject_
  Implements IDataParameter, IDbDataParameter, ICloneable
```

Syntax: C#

```
public sealed class MySqlParameter : MarshalByRefObject, IDataParameter, IDbDataParameter, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlParameter Members , MySql.Data.MySqlClient Namespace

MySqlParameter Members

MySqlParameter overview

Public Instance Constructors

MySqlParameter	Overloaded. Initializes a new instance of the MySqlParameter class.

Public Instance Properties

DbType	Gets or sets the DbTypeof the parameter.
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional,

	or a stored procedure return value parameter. As of MySql version 4.1 and earlier, input-only is the only valid choice.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
IsUnsigned	
MySqlDbType	Gets or sets the MySqlDbType of the parameter.
ParameterName	Gets or sets the name of the MySqlParameter.
Precision	Gets or sets the maximum number of digits used to represent the Value property.
Scale	Gets or sets the number of decimal places to which Value is resolved.
Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSetand used for loading or returning the Value .
SourceVersion	Gets or sets the DataRowVersionto use when loading Value .
Value	Gets or sets the value of the parameter.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ToString	Overridden. Gets a string containing the ParameterName .

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

MySqlParameter Constructor

Initializes a new instance of the MySqlParameter class.

Overload List

Initializes a new instance of the MySqlParameter class.

- public MySqlParameter();

Initializes a new instance of the MySqlParameter class with the parameter name and the data type.

- public MySqlParameter(string,MySqlDbType);

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType , and the size.

- public MySqlParameter(string,MySqlDbType,int);

Initializes a new instance of the MySqlParameter class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersionto use, and the value of the parameter.

- public MySqlParameter(string,MySqlDbType,int,ParameterDirection,bool,byte,byte,string,DataRowVersion,object);

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType , the size, and the source column name.

- public MySqlParameter(string,MySqlDbType,int,string);

Initializes a new instance of the MySqlParameter class with the parameter name and a value of the new MySqlParameter.

- public MySqlParameter(string,object);

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

MySqlParameter Constructor ()

Initializes a new instance of the MySqlParameter class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlParameter();
```

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace , MySqlParameter Constructor Overload List

MySqlParameter Constructor (String, MySqlDbType)

Initializes a new instance of the MySqlParameter class with the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType _
)
```

Syntax: C#

```
public MySqlParameter(
stringparameterName,
MySqlDbTypedbType
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the MySqlDbType values.

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace , MySqlParameter Constructor Overload List

MySqlDbType Enumeration

Specifies MySQL specific data type of a field, property, for use in a MySqlParameter .

Syntax: Visual Basic

```
Public Enum MySqlDbType
```

Syntax: C#

```
public enum MySqlDbType
```

Members

Member Name	Description
VarString	A variable-length string containing 0 to 65535 characters
Timestamp	A timestamp. The range is '1970-01-01 00:00:01' to sometime in the year 2037
LongBlob	A BLOB or TEXT column with a maximum length of 4294967295 or 4G (2^32 - 1) characters
Time	The range is '-838:59:59' to '838:59:59'.
TinyBlob	A BLOB or TEXT column with a maximum length of 255 (2^8 - 1) characters
Datetime	The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
Decimal	A fixed precision and scale numeric value between -1038 -1 and 10 38 -1.
UByte	
Blob	A BLOB or TEXT column with a maximum length of 65535 (2^16 - 1) characters
Double	A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E +308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.

Newdate	Obsolete Use Datetime or Date type
Byte	The signed range is -128 to 127. The unsigned range is 0 to 255.
Date	Date The supported range is '1000-01-01' to '9999-12-31'.
VarChar	A variable-length string containing 0 to 255 characters
UInt16	
UInt24	
Int16	A 16-bit signed integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535
NewDecimal	New Decimal
Set	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', ... A SET can have a maximum of 64 members.
String	Obsolete Use VarChar type
Enum	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. An ENUM can have a maximum of 65535 distinct values
Geometry	
UInt64	
Int64	A 64-bit signed integer.
UInt32	
Int24	Specifies a 24 (3 byte) signed or unsigned value.
Bit	Bit-field data type
Float	A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
Year	A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970-2069 if you use the 2-digit format (70-69)
Int32	A 32-bit signed integer
MediumBlob	A BLOB or TEXT column with a maximum length of 16777215 (2^24 - 1) characters

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

MySqlParameter Constructor (String, MySqlDbType, Int32)

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType , and the size.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer _
)
```

Syntax: C#

```
public MySqlParameter(
stringparameterName,
MySqlDbTypedbType,
intsize
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the MySqlDbType values.
- `size`: The length of the parameter.

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace , MySqlParameter Constructor Overload List

MySqlParameter Constructor (String, MySqlDbType, Int32, ParameterDirection, Boolean, Byte, Byte, String, DataRowVersion, Object)

Initializes a new instance of the MySqlParameter class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersionto use, and the value of the parameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer, _
   ByVal direction As ParameterDirection, _
   ByVal isNullable As Boolean, _
   ByVal precision As Byte, _
   ByVal scale As Byte, _
   ByVal sourceColumn As String, _
   ByVal sourceVersion As DataRowVersion, _
   ByVal value As Object _
)
```

Syntax: C#

```
public MySqlParameter(
stringparameterName,
MySqlDbTypedbType,
```

```
intsize,
ParameterDirectiondirection,
boolisNullable,
byteprecision,
bytescale,
stringsourceColumn,
DataRowVersionsourceVersion,
objectvalue
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the MySqlDbType values.
- `size`: The length of the parameter.
- `direction`: One of the ParameterDirectionvalues.
- `isNullable`: true if the value of the field can be null, otherwise false.
- `precision`: The total number of digits to the left and right of the decimal point to which Value is resolved.
- `scale`: The total number of decimal places to which Value is resolved.
- `sourceColumn`: The name of the source column.
- `sourceVersion`: One of the DataRowVersionvalues.
- `value`: An Objectthat is the value of the MySqlParameter .

Exceptions

Exception Type	Condition
ArgumentException	

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace , MySqlParameter Constructor Overload List

Value Property

Gets or sets the value of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property Value As Object _
_
  Implements IDataParameter.Value
```

Syntax: C#

```
public object Value {get; set;}
```

Implements

IDataParameter.Value

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

MySqlParameter Constructor (String, MySqlDbType, Int32, String)

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType , the size, and the source column name.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer, _
   ByVal sourceColumn As String _
)
```

Syntax: C#

```
public MySqlParameter(
stringparameterName,
MySqlDbTypedbType,
intsize,
stringsourceColumn
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the MySqlDbType values.
- `size`: The length of the parameter.
- `sourceColumn`: The name of the source column.

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace , MySqlParameter Constructor Overload List

MySqlParameter Constructor (String, Object)

Initializes a new instance of the MySqlParameter class with the parameter name and a value of the new MySqlParameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal value As Object _
)
```

Syntax: C#

```
public MySqlParameter(
stringparameterName,
objectvalue
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `value`: An Objectthat is the value of the MySqlParameter .

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace , MySqlParameter Constructor Overload List

DbType Property

Gets or sets the DbTypeof the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property DbType As DbType _
_
  Implements IDataParameter.DbType
```

Syntax: C#

```
public System.Data.DbType DbType {get; set;}
```

Implements

IDataParameter.DbType

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySql version 4.1 and earlier, input-only is the only valid choice.

Syntax: Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _
_
  Implements IDataParameter.Direction
```

Syntax: C#

```
public System.Data.ParameterDirection Direction {get; set;}
```

Implements

IDataParameter.Direction

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

Syntax: Visual Basic

```
NotOverridable Public Property IsNullable As Boolean _
```

```
_
  Implements IDataParameter.IsNullable
```

Syntax: C#

```
public bool IsNullable {get; set;}
```

Implements

IDataParameter.IsNullable

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

IsUnsigned Property

Syntax: Visual Basic

```
Public Property IsUnsigned As Boolean
```

Syntax: C#

```
public bool IsUnsigned {get; set;}
```

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

MySqlDbType Property

Gets or sets the MySqlDbType of the parameter.

Syntax: Visual Basic

```
Public Property MySqlDbType As MySqlDbType
```

Syntax: C#

```
public MySqlDbType MySqlDbType {get; set;}
```

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

ParameterName Property

Gets or sets the name of the MySqlParameter.

Syntax: Visual Basic

```
NotOverridable Public Property ParameterName As String _
_
  Implements IDataParameter.ParameterName
```

Syntax: C#

```
public string ParameterName {get; set;}
```

Implements

IDataParameter.ParameterName

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

Precision Property

Gets or sets the maximum number of digits used to represent the Value property.

Syntax: Visual Basic

```
NotOverridable Public Property Precision As Byte _
_
  Implements IDbDataParameter.Precision
```

Syntax: C#

```
public byte Precision {get; set;}
```

Implements

IDbDataParameter.Precision

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

Scale Property

Gets or sets the number of decimal places to which Value is resolved.

Syntax: Visual Basic

```
NotOverridable Public Property Scale As Byte _
_
  Implements IDbDataParameter.Scale
```

Syntax: C#

```
public byte Scale {get; set;}
```

Implements

IDbDataParameter.Scale

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

Syntax: Visual Basic

```
NotOverridable Public Property Size As Integer _
_
  Implements IDbDataParameter.Size
```

Syntax: C#

```
public int Size {get; set;}
```

Implements

IDbDataParameter.Size

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSetand used for loading or returning the Value .

Syntax: Visual Basic

```
NotOverridable Public Property SourceColumn As String _
_
  Implements IDataParameter.SourceColumn
```

Syntax: C#

```
public string SourceColumn {get; set;}
```

Implements

IDataParameter.SourceColumn

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

SourceVersion Property

Gets or sets the DataRowVersionto use when loading Value .

Syntax: Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _
_
  Implements IDataParameter.SourceVersion
```

Syntax: C#

```
public System.Data.DataRowVersion SourceVersion {get; set;}
```

Implements

IDataParameter.SourceVersion

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

MySqlParameter.ToString Method

Overridden. Gets a string containing the ParameterName .

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

Return Value

See Also

MySqlParameter Class , MySql.Data.MySqlClient Namespace

Item Property (Int32)

Gets the MySqlParameter at the specified index.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _
   ByVal index As Integer _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter this[
intindex
] {get; set;}
```

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Item Overload List

Item Property (String)

Gets the MySqlParameter with the specified name.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _
   ByVal name As String _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter this[
stringname
] {get; set;}
```

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Item Overload List

Add Method

Adds the specified MySqlParameter object to the MySqlParameterCollection .

Overload List

Adds the specified MySqlParameter object to the MySqlParameterCollection .

- public MySqlParameter Add(MySqlParameter);

Adds the specified MySqlParameter object to the MySqlParameterCollection .

- public int Add(object);

Adds a MySqlParameter to the MySqlParameterCollection given the parameter name and the data type.

- public MySqlParameter Add(string,MySqlDbType);

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, and the column length.

- public MySqlParameter Add(string,MySqlDbType,int);

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, the column length, and the source column name.

- public MySqlParameter Add(string,MySqlDbType,int,string);

Adds a MySqlParameter to the MySqlParameterCollection given the specified parameter name and value.

- public MySqlParameter Add(string,object);

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection.Add Method (MySqlParameter)

Adds the specified MySqlParameter object to the MySqlParameterCollection .

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal value As MySqlParameter _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
MySqlParametervalue
);
```

Parameters

- `value`: The MySqlParameter to add to the collection.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method (Object)

Adds the specified MySqlParameter object to the MySqlParameterCollection .

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Add( _
   ByVal value As Object _
) As Integer _
_
  Implements IList.Add
```

Syntax: C#

```
public int Add(
objectvalue
);
```

Parameters

- `value`: The MySqlParameter to add to the collection.

Return Value

The index of the new MySqlParameter object.

Implements

IList.Add

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method (String, MySqlDbType)

Adds a MySqlParameter to the MySqlParameterCollection given the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
stringparameterName,
MySqlDbTypedbType
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the MySqlDbType values.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method (String, MySqlDbType, Int32)

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, and the column length.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
stringparameterName,
MySqlDbTypedbType,
intsize
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the MySqlDbType values.
- `size`: The length of the column.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method (String, MySqlDbType, Int32, String)

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, the column length, and the source column name.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer, _
   ByVal sourceColumn As String _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
stringparameterName,
MySqlDbTypedbType,
intsize,
```

```
stringsourceColumn
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the MySqlDbType values.
- `size`: The length of the column.
- `sourceColumn`: The name of the source column.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method (String, Object)

Adds a MySqlParameter to the MySqlParameterCollection given the specified parameter name and value.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal value As Object _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
stringparameterName,
objectvalue
);
```

Parameters

- `parameterName`: The name of the parameter.
- `value`: The Value of the MySqlParameter to add to the collection.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Clear Method

Removes all items from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Clear() _
_
  Implements IList.Clear
```

Syntax: C#

```
public void Clear();
```

Implements

IList.Clear

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

Contains Method

Gets a value indicating whether a MySqlParameter exists in the collection.

Overload List

Gets a value indicating whether a MySqlParameter exists in the collection.

- public bool Contains(object);

Gets a value indicating whether a MySqlParameter with the specified parameter name exists in the collection.

- public bool Contains(string);

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection.Contains Method (Object)

Gets a value indicating whether a MySqlParameter exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _
   ByVal value As Object _
) As Boolean _
_
  Implements IList.Contains
```

Syntax: C#

```
public bool Contains(
objectvalue
);
```

Parameters

- `value`: The value of the MySqlParameter object to find.

Return Value

true if the collection contains the MySqlParameter object; otherwise, false.

Implements

IList.Contains

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Contains Overload List

MySqlParameterCollection.Contains Method (String)

Gets a value indicating whether a MySqlParameter with the specified parameter name exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _
   ByVal name As String _
) As Boolean _
_
  Implements IDataParameterCollection.Contains
```

Syntax: C#

```
public bool Contains(
stringname
);
```

Parameters

- `name`: The name of the MySqlParameter object to find.

Return Value

true if the collection contains the parameter; otherwise, false.

Implements

IDataParameterCollection.Contains

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.Contains Overload List

MySqlParameterCollection.CopyTo Method

Copies MySqlParameter objects from the MySqlParameterCollection to the specified array.

Syntax: Visual Basic

```
NotOverridable Public Sub CopyTo( _
   ByVal array As Array, _
   ByVal index As Integer _
) _
_
  Implements ICollection.CopyTo
```

Syntax: C#

```
public void CopyTo(
```

```
Arrayarray,
intindex
);
```

Parameters

- `array`:
- `index`:

Implements

ICollection.CopyTo

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

IndexOf Method

Gets the location of a MySqlParameter in the collection.

Overload List

Gets the location of a MySqlParameter in the collection.

- public int IndexOf(object);

Gets the location of the MySqlParameter in the collection with a specific parameter name.

- public int IndexOf(string);

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection.IndexOf Method (Object)

Gets the location of a MySqlParameter in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _
   ByVal value As Object _
) As Integer _
_
  Implements IList.IndexOf
```

Syntax: C#

```
public int IndexOf(
objectvalue
);
```

Parameters

- `value`: The MySqlParameter object to locate.

Return Value

The zero-based location of the MySqlParameter in the collection.

Implements

IList.IndexOf

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace ,
MySqlParameterCollection.IndexOf Overload List

MySqlParameterCollection.IndexOf Method (String)

Gets the location of the MySqlParameter in the collection with a specific parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _
   ByVal parameterName As String _
) As Integer _
_
  Implements IDataParameterCollection.IndexOf
```

Syntax: C#

```
public int IndexOf(
stringparameterName
);
```

Parameters

- `parameterName`: The name of the MySqlParameter object to retrieve.

Return Value

The zero-based location of the MySqlParameter in the collection.

Implements

IDataParameterCollection.IndexOf

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace ,
MySqlParameterCollection.IndexOf Overload List

MySqlParameterCollection.Insert Method

Inserts a MySqlParameter into the collection at the specified index.

Syntax: Visual Basic

```
NotOverridable Public Sub Insert( _
   ByVal index As Integer, _
   ByVal value As Object _
) _
_
  Implements IList.Insert
```

Syntax: C#

```
public void Insert(
intindex,
```

```
objectvalue
);
```

Parameters

- `index`:
- `value`:

Implements

IList.Insert

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection.Remove Method

Removes the specified MySqlParameter from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Remove( _
   ByVal value As Object _
) _
_
  Implements IList.Remove
```

Syntax: C#

```
public void Remove(
objectvalue
);
```

Parameters

- `value`:

Implements

IList.Remove

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

RemoveAt Method

Removes the specified MySqlParameter from the collection.

Overload List

Removes the specified MySqlParameter from the collection using a specific index.

- public void RemoveAt(int);

Removes the specified MySqlParameter from the collection using the parameter name.

- public void RemoveAt(string);

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace

MySqlParameterCollection.RemoveAt Method (Int32)

Removes the specified MySqlParameter from the collection using a specific index.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _
   ByVal index As Integer _
) _
_
  Implements IList.RemoveAt
```

Syntax: C#

```
public void RemoveAt(
intindex
);
```

Parameters

- `index`: The zero-based index of the parameter.

Implements

IList.RemoveAt

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.RemoveAt Overload List

MySqlParameterCollection.RemoveAt Method (String)

Removes the specified MySqlParameter from the collection using the parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _
   ByVal name As String _
) _
_
  Implements IDataParameterCollection.RemoveAt
```

Syntax: C#

```
public void RemoveAt(
stringname
);
```

Parameters

- `name`: The name of the MySqlParameter object to retrieve.

Implements

IDataParameterCollection.RemoveAt

See Also

MySqlParameterCollection Class , MySql.Data.MySqlClient Namespace , MySqlParameterCollection.RemoveAt Overload List

Transaction Property

Syntax: Visual Basic

```
Public Property Transaction As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction Transaction {get; set;}
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

UpdatedRowSource Property

Syntax: Visual Basic

```
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _
_
  Implements IDbCommand.UpdatedRowSource
```

Syntax: C#

```
public System.Data.UpdateRowSource UpdatedRowSource {get; set;}
```

Implements

IDbCommand.UpdatedRowSource

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand.Cancel Method

Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

Syntax: Visual Basic

```
NotOverridable Public Sub Cancel() _
_
  Implements IDbCommand.Cancel
```

Syntax: C#

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

Cancelling an executing command is currently not supported on any version of MySQL.

Exceptions

Exception Type	Condition
NotSupportedException	This operation is not supported.

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand.CreateParameter Method

Creates a new instance of a MySqlParameter object.

Syntax: Visual Basic

```
Public Function CreateParameter() As MySqlParameter
```

Syntax: C#

```
public MySqlParameter CreateParameter();
```

Return Value

A MySqlParameter object.

Remarks

This method is a strongly-typed version of CreateParameter.

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand.ExecuteNonQuery Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteNonQuery() As Integer _
_
  Implements IDbCommand.ExecuteNonQuery
```

Syntax: C#

```
public int ExecuteNonQuery();
```

Implements

IDbCommand.ExecuteNonQuery

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

ExecuteReader Method

Overload List

- public MySqlDataReader ExecuteReader();
- public MySqlDataReader ExecuteReader(CommandBehavior);

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand.ExecuteReader Method ()

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader() As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader();
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace , MySqlCommand.ExecuteReader Overload List

MySqlDataReader Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

For a list of all members of this type, see MySqlDataReader Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataReader_
  Inherits MarshalByRefObject_
  Implements IEnumerable, IDataReader, IDisposable, IDataRecord
```

Syntax: C#

```
public sealed class MySqlDataReader : MarshalByRefObject, IEnumerable, IDataReader, IDisposable, IDataReco
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlDataReader Members , MySql.Data.MySqlClient Namespace

MySqlDataReader Members

MySqlDataReader overview

Public Instance Properties

Depth	Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

FieldCount	Gets the number of columns in the current row.
HasRows	Gets a value indicating whether the MySqlDataReader contains one or more rows.
IsClosed	Gets a value indicating whether the data reader is closed.
Item	Overloaded. Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Public Instance Methods

Close	Closes the MySqlDataReader object.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetBoolean	Gets the value of the specified column as a Boolean.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.
GetChar	Gets the value of the specified column as a single character.
GetChars	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDateTime	
GetDecimal	
GetDouble	
GetFieldType	Gets the Type that is the data type of the object.
GetFloat	
GetGuid	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetInt16	
GetInt32	
GetInt64	
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.

GetMySqlDateTime	
GetName	Gets the name of the specified column.
GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a DataTable that describes the column metadata of the MySqlDataReader.
GetString	
GetTimeSpan	
GetType(inherited from Object)	Gets the Typeof the current instance.
GetUInt16	
GetUInt32	
GetUInt64	
GetValue	Gets the value of the specified column in its native format.
GetValues	Gets all attribute columns in the collection for the current row.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read	Advances the MySqlDataReader to the next record.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

Depth Property

Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Depth As Integer _
_
  Implements IDataReader.Depth
```

Syntax: C#

```
public int Depth {get;}
```

Implements

IDataReader.Depth

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

FieldCount Property

Gets the number of columns in the current row.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property FieldCount As Integer _
_
  Implements IDataRecord.FieldCount
```

Syntax: C#

```
public int FieldCount {get;}
```

Implements

IDataRecord.FieldCount

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

HasRows Property

Gets a value indicating whether the MySqlDataReader contains one or more rows.

Syntax: Visual Basic

```
Public ReadOnly Property HasRows As Boolean
```

Syntax: C#

```
public bool HasRows {get;}
```

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

IsClosed Property

Gets a value indicating whether the data reader is closed.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsClosed As Boolean _
_
  Implements IDataReader.IsClosed
```

Syntax: C#

```
public bool IsClosed {get;}
```

Implements

IDataReader.IsClosed

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

Item Property

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Overload List

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

- public object this[int] {get;}

Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

- public object this[string] {get;}

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

Item Property (Int32)

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _
   ByVal i As Integer _
) _
_
  Implements IDataRecord.Item As Object _
_
  Implements IDataRecord.Item
```

Syntax: C#

```
public object this[
inti
] {get;}
```

Implements

IDataRecord.Item

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace , MySqlDataReader.Item Overload List

Item Property (String)

Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _
   ByVal name As String _
) _
_
```

```
  Implements IDataRecord.Item As Object _
_
  Implements IDataRecord.Item
```

Syntax: C#

```
public object this[
stringname
] {get;}
```

Implements

IDataRecord.Item

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace , MySqlDataReader.Item Overload List

RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property RecordsAffected As Integer _
_
  Implements IDataReader.RecordsAffected
```

Syntax: C#

```
public int RecordsAffected {get;}
```

Implements

IDataReader.RecordsAffected

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.Close Method

Closes the MySqlDataReader object.

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _
_
  Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetBoolean Method

Gets the value of the specified column as a Boolean.

Syntax: Visual Basic

```
NotOverridable Public Function GetBoolean( _
   ByVal i As Integer _
) As Boolean _
_
  Implements IDataRecord.GetBoolean
```

Syntax: C#

```
public bool GetBoolean(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetBoolean

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetByte Method

Gets the value of the specified column as a byte.

Syntax: Visual Basic

```
NotOverridable Public Function GetByte( _
   ByVal i As Integer _
) As Byte _
_
  Implements IDataRecord.GetByte
```

Syntax: C#

```
public byte GetByte(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetByte

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetBytes Method

Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetBytes( _
   ByVal i As Integer, _
   ByVal dataIndex As Long, _
   ByVal buffer As Byte(), _
   ByVal bufferIndex As Integer, _
   ByVal length As Integer _
) As Long _
_
  Implements IDataRecord.GetBytes
```

Syntax: C#

```
public long GetBytes(
inti,
longdataIndex,
byte[]buffer,
intbufferIndex,
intlength
);
```

Parameters

- `i`: The zero-based column ordinal.
- `dataIndex`: The index within the field from which to begin the read operation.
- `buffer`: The buffer into which to read the stream of bytes.
- `bufferIndex`: The index for buffer to begin the read operation.
- `length`: The maximum length to copy into the buffer.

Return Value

The actual number of bytes read.

Implements

IDataRecord.GetBytes

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetChar Method

Gets the value of the specified column as a single character.

Syntax: Visual Basic

```
NotOverridable Public Function GetChar( _
   ByVal i As Integer _
```

```
) As Char _
_
  Implements IDataRecord.GetChar
```

Syntax: C#

```
public char GetChar(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetChar

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetChars Method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetChars( _
   ByVal i As Integer, _
   ByVal fieldOffset As Long, _
   ByVal buffer As Char(), _
   ByVal bufferoffset As Integer, _
   ByVal length As Integer _
) As Long _
_
  Implements IDataRecord.GetChars
```

Syntax: C#

```
public long GetChars(
inti,
longfieldOffset,
char[]buffer,
intbufferoffset,
intlength
);
```

Parameters

- i:
- fieldOffset:
- buffer:
- bufferoffset:
- length:

Return Value

Implements

IDataRecord.GetChars

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDataTypeName Method

Gets the name of the source data type.

Syntax: Visual Basic

```
NotOverridable Public Function GetDataTypeName( _
   ByVal i As Integer _
) As String _
_
  Implements IDataRecord.GetDataTypeName
```

Syntax: C#

```
public string GetDataTypeName(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetDataTypeName

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDateTime Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDateTime( _
   ByVal index As Integer _
) As Date _
_
  Implements IDataRecord.GetDateTime
```

Syntax: C#

```
public DateTime GetDateTime(
intindex
);
```

Implements

IDataRecord.GetDateTime

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDecimal Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDecimal( _
   ByVal index As Integer _
) As Decimal _
_
  Implements IDataRecord.GetDecimal
```

Syntax: C#

```
public decimal GetDecimal(
intindex
);
```

Implements

IDataRecord.GetDecimal

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDouble Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDouble( _
   ByVal index As Integer _
) As Double _
_
  Implements IDataRecord.GetDouble
```

Syntax: C#

```
public double GetDouble(
intindex
);
```

Implements

IDataRecord.GetDouble

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetFieldType Method

Gets the Type that is the data type of the object.

Syntax: Visual Basic

```
NotOverridable Public Function GetFieldType( _
   ByVal i As Integer _
) As Type _
```

```
_
  Implements IDataRecord.GetFieldType
```

Syntax: C#

```
public Type GetFieldType(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetFieldType

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetFloat Method

Syntax: Visual Basic

```
NotOverridable Public Function GetFloat( _
   ByVal index As Integer _
) As Single _
_
  Implements IDataRecord.GetFloat
```

Syntax: C#

```
public float GetFloat(
intindex
);
```

Implements

IDataRecord.GetFloat

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetGuid Method

Syntax: Visual Basic

```
NotOverridable Public Function GetGuid( _
   ByVal index As Integer _
) As Guid _
_
  Implements IDataRecord.GetGuid
```

Syntax: C#

```
public Guid GetGuid(
intindex
```

```
);
```

Implements

IDataRecord.GetGuid

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetInt16 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt16( _
   ByVal index As Integer _
) As Short _
_
  Implements IDataRecord.GetInt16
```

Syntax: C#

```
public short GetInt16(
intindex
);
```

Implements

IDataRecord.GetInt16

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetInt32 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt32( _
   ByVal index As Integer _
) As Integer _
_
  Implements IDataRecord.GetInt32
```

Syntax: C#

```
public int GetInt32(
intindex
);
```

Implements

IDataRecord.GetInt32

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetInt64 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt64( _
   ByVal index As Integer _
) As Long _
_
  Implements IDataRecord.GetInt64
```

Syntax: C#

```
public long GetInt64(
intindex
);
```

Implements

IDataRecord.GetInt64

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetMySqlDateTime Method

Syntax: Visual Basic

```
Public Function GetMySqlDateTime( _
   ByVal index As Integer _
) As MySqlDateTime
```

Syntax: C#

```
public MySqlDateTime GetMySqlDateTime(
intindex
);
```

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetName Method

Gets the name of the specified column.

Syntax: Visual Basic

```
NotOverridable Public Function GetName( _
   ByVal i As Integer _
) As String _
_
  Implements IDataRecord.GetName
```

Syntax: C#

```
public string GetName(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetName

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetOrdinal Method

Gets the column ordinal, given the name of the column.

Syntax: Visual Basic

```
NotOverridable Public Function GetOrdinal( _
   ByVal name As String _
) As Integer _
_
  Implements IDataRecord.GetOrdinal
```

Syntax: C#

```
public int GetOrdinal(
stringname
);
```

Parameters

- `name`:

Return Value

Implements

IDataRecord.GetOrdinal

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetSchemaTable Method

Returns a DataTable that describes the column metadata of the MySqlDataReader.

Syntax: Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _
_
  Implements IDataReader.GetSchemaTable
```

Syntax: C#

```
public DataTable GetSchemaTable();
```

Return Value

Implements

IDataReader.GetSchemaTable

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetString Method

Syntax: Visual Basic

```
NotOverridable Public Function GetString( _
   ByVal index As Integer _
) As String _
_
  Implements IDataRecord.GetString
```

Syntax: C#

```
public string GetString(
intindex
);
```

Implements

IDataRecord.GetString

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetTimeSpan Method

Syntax: Visual Basic

```
Public Function GetTimeSpan( _
   ByVal index As Integer _
) As TimeSpan
```

Syntax: C#

```
public TimeSpan GetTimeSpan(
intindex
);
```

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetUInt16 Method

Syntax: Visual Basic

```
Public Function GetUInt16( _
   ByVal index As Integer _
) As UInt16
```

Syntax: C#

```
public ushort GetUInt16(
intindex
);
```

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetUInt32 Method

Syntax: Visual Basic

```
Public Function GetUInt32( _
   ByVal index As Integer _
) As UInt32
```

Syntax: C#

```
public uint GetUInt32(
intindex
);
```

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetUInt64 Method

Syntax: Visual Basic

```
Public Function GetUInt64( _
   ByVal index As Integer _
) As UInt64
```

Syntax: C#

```
public ulong GetUInt64(
intindex
);
```

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetValue Method

Gets the value of the specified column in its native format.

Syntax: Visual Basic

```
NotOverridable Public Function GetValue( _
   ByVal i As Integer _
) As Object _
_
  Implements IDataRecord.GetValue
```

Syntax: C#

```
public object GetValue(
inti
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetValue

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.GetValues Method

Gets all attribute columns in the collection for the current row.

Syntax: Visual Basic

```
NotOverridable Public Function GetValues( _
   ByVal values As Object() _
) As Integer _
_
  Implements IDataRecord.GetValues
```

Syntax: C#

```
public int GetValues(
object[]values
);
```

Parameters

- `values`:

Return Value

Implements

IDataRecord.GetValues

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.IsDBNull Method

Gets a value indicating whether the column contains non-existent or missing values.

Syntax: Visual Basic

```
NotOverridable Public Function IsDBNull( _
   ByVal i As Integer _
) As Boolean _
_
  Implements IDataRecord.IsDBNull
```

Syntax: C#

```
public bool IsDBNull(
inti
);
```

Parameters

- `i`:

Return Value

Implements

IDataRecord.IsDBNull

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.NextResult Method

Advances the data reader to the next result, when reading the results of batch SQL statements.

Syntax: Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _
_
  Implements IDataReader.NextResult
```

Syntax: C#

```
public bool NextResult();
```

Return Value

Implements

IDataReader.NextResult

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlDataReader.Read Method

Advances the MySqlDataReader to the next record.

Syntax: Visual Basic

```
NotOverridable Public Function Read() As Boolean _
_
  Implements IDataReader.Read
```

Syntax: C#

```
public bool Read();
```

Return Value

Implements

IDataReader.Read

See Also

MySqlDataReader Class , MySql.Data.MySqlClient Namespace

MySqlCommand.ExecuteReader Method (CommandBehavior)

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader( _
```

```
   ByVal behavior As CommandBehavior _
) As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader(
CommandBehaviorbehavior
);
```

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace , MySqlCommand.ExecuteReader Overload List

MySqlCommand.ExecuteScalar Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteScalar() As Object _
_
  Implements IDbCommand.ExecuteScalar
```

Syntax: C#

```
public object ExecuteScalar();
```

Implements

IDbCommand.ExecuteScalar

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommand.Prepare Method

Syntax: Visual Basic

```
NotOverridable Public Sub Prepare() _
_
  Implements IDbCommand.Prepare
```

Syntax: C#

```
public void Prepare();
```

Implements

IDbCommand.Prepare

See Also

MySqlCommand Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder Class

For a list of all members of this type, see MySqlCommandBuilder Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandBuilder_
```

```
  Inherits Component
```

Syntax: C#

```
public sealed class MySqlCommandBuilder : Component
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlCommandBuilder Members , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder Members

MySqlCommandBuilder overview

Public Static (Shared) Methods

DeriveParameters	Overloaded. Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Public Instance Constructors

MySqlCommandBuilder	Overloaded. Initializes a new instance of the MySqlCommandBuilder class.

Public Instance Properties

Container(inherited from Component)	Gets the IContainerthat contains the Component.
DataAdapter	
QuotePrefix	
QuoteSuffix	
Site(inherited from Component)	Gets or sets the ISiteof the Component.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetDeleteCommand	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetInsertCommand	
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
GetUpdateCommand	
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
RefreshSchema	
ToString(inherited from Component)	Returns a Stringcontaining the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposedevent on the component.

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Overload List

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

- public static void DeriveParameters(MySqlCommand);
- public static void DeriveParameters(MySqlCommand,bool);

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.DeriveParameters Method (MySqlCommand)

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
   ByVal command As MySqlCommand _
)
```

Syntax: C#

```
public static void DeriveParameters(
MySqlCommandcommand
);
```

Parameters

- `command`: The MySqlCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the MySqlCommand.

Exceptions

Exception Type	Condition
InvalidOperationException	The command text is not a valid stored procedure name.

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace , MySqlCommandBuilder.DeriveParameters Overload List

MySqlCommandBuilder.DeriveParameters Method (MySqlCommand, Boolean)

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
   ByVal command As MySqlCommand, _
   ByVal useProc As Boolean _
)
```

Syntax: C#

```
public static void DeriveParameters(
MySqlCommandcommand,
booluseProc
);
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace , MySqlCommandBuilder.DeriveParameters Overload List

MySqlCommandBuilder Constructor

Initializes a new instance of the MySqlCommandBuilder class.

Overload List

Initializes a new instance of the MySqlCommandBuilder class.

- public MySqlCommandBuilder();
- public MySqlCommandBuilder(MySqlDataAdapter);

- public MySqlCommandBuilder(MySqlDataAdapter,bool);
- public MySqlCommandBuilder(bool);

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder Constructor ()

Initializes a new instance of the MySqlCommandBuilder class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommandBuilder();
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace , MySqlCommandBuilder Constructor Overload List

MySqlCommandBuilder Constructor (MySqlDataAdapter)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal adapter As MySqlDataAdapter _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
MySqlDataAdapteradapter
);
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace , MySqlCommandBuilder Constructor Overload List

MySqlDataAdapter Class

For a list of all members of this type, see MySqlDataAdapter Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataAdapter_
  Inherits DbDataAdapter
```

Syntax: C#

```
public sealed class MySqlDataAdapter : DbDataAdapter
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlDataAdapter Members , MySql.Data.MySqlClient Namespace

MySqlDataAdapter Members

MySqlDataAdapter overview

Public Instance Constructors

MySqlDataAdapter	Overloaded. Initializes a new instance of the MySqlDataAdapter class.

Public Instance Properties

AcceptChangesDuringFill(inherited from DataAdapter)	Gets or sets a value indicating whether AcceptChangesis called on a DataRowafter it is added to the DataTableduring any of the Fill operations.
AcceptChangesDuringUpdate(inherited from DataAdapter)	Gets or sets whether AcceptChangesis called during a Update.
Container(inherited from Component)	Gets the IContainerthat contains the Component.
ContinueUpdateOnError(inherited from DataAdapter)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand	Overloaded.
FillLoadOption(inherited from DataAdapter)	Gets or sets the LoadOptionthat determines how the adapter fills the DataTablefrom the DbDataReader.
InsertCommand	Overloaded.
MissingMappingAction(inherited from DataAdapter)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction(inherited from DataAdapter)	Determines the action to take when existing DataSetschema does not match incoming data.
ReturnProviderSpecificTypes(inherited from DataAdapter)	Gets or sets whether the Fillmethod should return provider-specific values or common CLS-compliant values.
SelectCommand	Overloaded.
Site(inherited from Component)	Gets or sets the ISiteof the Component.
TableMappings(inherited from DataAdapter)	Gets a collection that provides the master mapping between a source table and a DataTable.
UpdateBatchSize(inherited from DbDataAdapter)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Overloaded.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
Fill(inherited from DbDataAdapter)	Overloaded. Adds or refreshes rows in the DataSetto match those in the data source using the DataSetname, and creates a DataTablenamed "Table."
FillSchema(inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTablebased on the specified SchemaType.
GetFillParameters(inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL `SELECT` statement.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ResetFillLoadOption(inherited from DataAdapter)	Resets FillLoadOptionto its default state and causes Fillto honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill(inherited from DataAdapter)	Determines whether the AcceptChangesDuringFillproperty should be persisted.
ShouldSerializeFillLoadOption(inherited from DataAdapter)	Determines whether the FillLoadOptionproperty should be persisted.
ToString(inherited from Component)	Returns a Stringcontaining the name of the Component, if any. This method should not be overridden.
Update(inherited from DbDataAdapter)	Overloaded. Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposedevent on the component.
FillError(inherited from DataAdapter)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

RowUpdating	Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Protected Internal Instance Properties

FillCommandBehavior(inherited from DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

MySqlDataAdapter Constructor

Initializes a new instance of the MySqlDataAdapter class.

Overload List

Initializes a new instance of the MySqlDataAdapter class.

- public MySqlDataAdapter();
- public MySqlDataAdapter(MySqlCommand);
- public MySqlDataAdapter(string,MySqlConnection);
- public MySqlDataAdapter(string,string);

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

MySqlDataAdapter Constructor ()

Initializes a new instance of the MySqlDataAdapter class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlDataAdapter();
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace , MySqlDataAdapter Constructor Overload List

MySqlDataAdapter Constructor (MySqlCommand)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal selectCommand As MySqlCommand _
)
```

Syntax: C#

```
public MySqlDataAdapter(
MySqlCommandselectCommand
);
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace , MySqlDataAdapter Constructor Overload List

MySqlDataAdapter Constructor (String, MySqlConnection)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal selectCommandText As String, _
   ByVal connection As MySqlConnection _
)
```

Syntax: C#

```
public MySqlDataAdapter(
stringselectCommandText,
MySqlConnectionconnection
);
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace , MySqlDataAdapter Constructor Overload List

MySqlDataAdapter Constructor (String, String)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal selectCommandText As String, _
   ByVal selectConnString As String _
)
```

Syntax: C#

```
public MySqlDataAdapter(
stringselectCommandText,
stringselectConnString
);
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace , MySqlDataAdapter Constructor Overload List

DeleteCommand Property

Syntax: Visual Basic

```
Overloads Public Property DeleteCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand DeleteCommand {get; set;}
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

InsertCommand Property

Syntax: Visual Basic

```
Overloads Public Property InsertCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand InsertCommand {get; set;}
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

SelectCommand Property

Syntax: Visual Basic

```
Overloads Public Property SelectCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand SelectCommand {get; set;}
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

UpdateCommand Property

Syntax: Visual Basic

```
Overloads Public Property UpdateCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

MySqlDataAdapter.RowUpdated Event

Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdated As MySqlRowUpdatedEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatedEventHandler RowUpdated;
```

Event Data

The event handler receives an argument of type MySqlRowUpdatedEventArgs containing data related to this event. The following MySqlRowUpdatedEventArgsproperties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand executed when Update is called.
Errors	Gets any errors generated by the .NET Framework data provider when the Commandwas executed.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row	Gets the DataRowsent through an Update.
RowCount	Gets the number of rows processed in a batch of updated records.
StatementType	Gets the type of SQL statement executed.
Status	Gets the UpdateStatusof the Commandproperty.
TableMapping	Gets the DataTableMappingsent through an Update.

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

MySqlRowUpdatedEventHandler Delegate

Represents the method that will handle the RowUpdatedevent of a MySqlDataAdapter .

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatedEventHandler( _
   ByVal sender As Object, _
   ByVal e As MySqlRowUpdatedEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatedEventHandler(
objectsender,
MySqlRowUpdatedEventArgse
);
```

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

MySqlRowUpdatedEventArgs Class

Provides data for the RowUpdated event. This class cannot be inherited.

For a list of all members of this type, see MySqlRowUpdatedEventArgs Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatedEventArgs_
  Inherits RowUpdatedEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatedEventArgs : RowUpdatedEventArgs
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlRowUpdatedEventArgs Members , MySql.Data.MySqlClient Namespace

MySqlRowUpdatedEventArgs Members

MySqlRowUpdatedEventArgs overview

Public Instance Constructors

MySqlRowUpdatedEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand executed when Update is called.
Errors(inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the Commandwas executed.
RecordsAffected(inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row(inherited from RowUpdatedEventArgs)	Gets the DataRowsent through an Update.
RowCount(inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
StatementType(inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status(inherited from RowUpdatedEventArgs)	Gets the UpdateStatusof the Commandproperty.
TableMapping(inherited from RowUpdatedEventArgs)	Gets the DataTableMappingsent through an Update.

Public Instance Methods

CopyToRows(inherited from RowUpdatedEventArgs)	Overloaded. Copies references to the modified rows into the provided array.

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

MySqlRowUpdatedEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlRowUpdatedEventArgs Constructor

Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
   ByVal row As DataRow, _
   ByVal command As IDbCommand, _
   ByVal statementType As StatementType, _
   ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySqlRowUpdatedEventArgs(
DataRowrow,
IDbCommandcommand,
StatementTypestatementType,
DataTableMappingtableMapping
);
```

Parameters

- `row`: The DataRowsent through an Update.
- `command`: The IDbCommandexecuted when Updateis called.
- `statementType`: One of the StatementTypevalues that specifies the type of query executed.
- `tableMapping`: The DataTableMappingsent through an Update.

See Also

MySqlRowUpdatedEventArgs Class , MySql.Data.MySqlClient Namespace

Command Property

Gets or sets the MySqlCommand executed when Update is called.

Syntax: Visual Basic

```
Overloads Public ReadOnly Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get;}
```

See Also

MySqlRowUpdatedEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlDataAdapter.RowUpdating Event

Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdating As MySqlRowUpdatingEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatingEventHandler RowUpdating;
```

Event Data

The event handler receives an argument of type MySqlRowUpdatingEventArgs containing data related to this event. The following MySqlRowUpdatingEventArgsproperties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand to execute when performing the Update.
Errors	Gets any errors generated by the .NET Framework data provider when the Commandexecutes.
Row	Gets the DataRowthat will be sent to the server as part of an insert, update, or delete operation.
StatementType	Gets the type of SQL statement to execute.
Status	Gets or sets the UpdateStatusof the Commandproperty.
TableMapping	Gets the DataTableMappingto send through the Update.

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

MySqlRowUpdatingEventHandler Delegate

Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter .

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatingEventHandler( _
   ByVal sender As Object, _
   ByVal e As MySqlRowUpdatingEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatingEventHandler(
objectsender,
MySqlRowUpdatingEventArgse
```

```
);
```

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

MySqlRowUpdatingEventArgs Class

Provides data for the RowUpdating event. This class cannot be inherited.

For a list of all members of this type, see MySqlRowUpdatingEventArgs Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatingEventArgs_
  Inherits RowUpdatingEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatingEventArgs : RowUpdatingEventArgs
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlRowUpdatingEventArgs Members , MySql.Data.MySqlClient Namespace

MySqlRowUpdatingEventArgs Members

MySqlRowUpdatingEventArgs overview

Public Instance Constructors

MySqlRowUpdatingEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatingEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand to execute when performing the Update.
Errors(inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the Commandexecutes.

Row(inherited from RowUpdatingEventArgs)	Gets the DataRowthat will be sent to the server as part of an insert, update, or delete operation.
StatementType(inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status(inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatusof the Commandproperty.
TableMapping(inherited from RowUpdatingEventArgs)	Gets the DataTableMappingto send through the Update.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

MySqlRowUpdatingEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlRowUpdatingEventArgs Constructor

Initializes a new instance of the MySqlRowUpdatingEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
   ByVal row As DataRow, _
   ByVal command As IDbCommand, _
   ByVal statementType As StatementType, _
   ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySqlRowUpdatingEventArgs(
DataRowrow,
IDbCommandcommand,
StatementTypestatementType,
DataTableMappingtableMapping
);
```

Parameters

- `row`: The DataRowto Update.
- `command`: The IDbCommandto execute during Update.
- `statementType`: One of the StatementTypevalues that specifies the type of query executed.
- `tableMapping`: The DataTableMappingsent through an Update.

See Also

MySqlRowUpdatingEventArgs Class , MySql.Data.MySqlClient Namespace

Command Property

Gets or sets the MySqlCommand to execute when performing the Update.

Syntax: Visual Basic

```
Overloads Public Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get; set;}
```

See Also

MySqlRowUpdatingEventArgs Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder Constructor (MySqlDataAdapter, Boolean)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal adapter As MySqlDataAdapter, _
   ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
MySqlDataAdapteradapter,
boollastOneWins
);
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace , MySqlCommandBuilder Constructor Overload List

MySqlCommandBuilder Constructor (Boolean)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
boollastOneWins
);
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace , MySqlCommandBuilder Constructor Overload List

DataAdapter Property

Syntax: Visual Basic

```
Public Property DataAdapter As MySqlDataAdapter
```

Syntax: C#

```
public MySqlDataAdapter DataAdapter {get; set;}
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

QuotePrefix Property

Syntax: Visual Basic

```
Public Property QuotePrefix As String
```

Syntax: C#

```
public string QuotePrefix {get; set;}
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

```
public string QuoteSuffix {get; set;}
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.GetDeleteCommand Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.GetInsertCommand Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.GetUpdateCommand Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.RefreshSchema Method

Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

MySqlCommandBuilder Class , MySql.Data.MySqlClient Namespace

MySqlException Class

The exception that is thrown when MySQL returns an error. This class cannot be inherited.

For a list of all members of this type, see MySqlException Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlException_
  Inherits SystemException
```

Syntax: C#

```
public sealed class MySqlException : SystemException
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlException Members , MySql.Data.MySqlClient Namespace

MySqlException Members

MySqlException overview

Public Instance Properties

Data(inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink(inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException(inherited from Exception)	Gets the Exceptioninstance that caused the current exception.
Message(inherited from Exception)	Gets a message that describes the current exception.
Number	Gets a number that identifies the type of error.
Source(inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace(inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite(inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetBaseException(inherited from Exception)	When overridden in a derived class, returns the Exceptionthat is the root cause of one or more subsequent exceptions.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData(inherited from Exception)	When overridden in a derived class, sets the SerializationInfowith information about the exception.
GetType(inherited from Exception)	Gets the runtime type of the current instance.
ToString(inherited from Exception)	Creates and returns a string representation of the current exception.

See Also

MySqlException Class , MySql.Data.MySqlClient Namespace

Number Property

Gets a number that identifies the type of error.

Syntax: Visual Basic

```
Public ReadOnly Property Number As Integer
```

Syntax: C#

```
public int Number {get;}
```

See Also

MySqlException Class , MySql.Data.MySqlClient Namespace

MySqlHelper Class

Helper class that makes it easier to work with the provider.

For a list of all members of this type, see MySqlHelper Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlHelper
```

Syntax: C#

```
public sealed class MySqlHelper
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlHelper Members , MySql.Data.MySqlClient Namespace

MySqlHelper Members

MySqlHelper overview

Public Static (Shared) Methods

ExecuteDataRow	Executes a single SQL command and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteDataset	Overloaded. Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteNonQuery	Overloaded. Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

ExecuteReader	Overloaded. Executes a single command against a MySQL database.
ExecuteScalar	Overloaded. Execute a single command against a MySQL database.
UpdateDataSet	Updates the given table with data from the given DataSet

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteDataRow Method

Executes a single SQL command and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Public Shared Function ExecuteDataRow( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray parms As MySqlParameter() _
) As DataRow
```

Syntax: C#

```
public static DataRow ExecuteDataRow(
stringconnectionString,
stringcommandText,
   params MySqlParameter[]parms
);
```

Parameters

- `connectionString`: Settings to be used for the connection
- `commandText`: Command to execute
- `parms`: Parameters to use for the command

Return Value

DataRow containing the first row of the resultset

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

ExecuteDataset Method

Executes a single SQL command and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

Overload List

Executes a single SQL command and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

- public static DataSet ExecuteDataset(MySqlConnection,string);

Executes a single SQL command and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

- public static DataSet ExecuteDataset(MySqlConnection,string,params MySqlParameter[]);

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- public static DataSet ExecuteDataset(string,string);

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- public static DataSet ExecuteDataset(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteDataset Method (MySqlConnection, String)

Executes a single SQL command and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
MySqlConnectionconnection,
stringcommandText
);
```

Parameters

- `connection`: MySqlConnection object to use
- `commandText`: Command to execute

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteDataset Overload List

MySqlHelper.ExecuteDataset Method (MySqlConnection, String, MySqlParameter[])

Executes a single SQL command and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
MySqlConnectionconnection,
stringcommandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connection`: MySqlConnection object to use
- `commandText`: Command to execute
- `commandParameters`: Parameters to use for the command

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteDataset Overload List

MySqlHelper.ExecuteDataset Method (String, String)

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connectionString As String, _
   ByVal commandText As String _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
stringconnectionString,
stringcommandText
);
```

Parameters

- `connectionString`: Settings to be used for the connection
- `commandText`: Command to execute

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteDataset Overload List

MySqlHelper.ExecuteDataset Method (String, String, MySqlParameter[])

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
stringconnectionString,
stringcommandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connectionString`: Settings to be used for the connection
- `commandText`: Command to execute
- `commandParameters`: Parameters to use for the command

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteDataset Overload List

ExecuteNonQuery Method

Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

Overload List

Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

- public static int ExecuteNonQuery(MySqlConnection,string,params MySqlParameter[]);

Executes a single command against a MySQL database. A new MySqlConnection is created using the ConnectionString given.

- public static int ExecuteNonQuery(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteNonQuery Method (MySqlConnection, String, MySqlParameter[])

Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
MySqlConnectionconnection,
stringcommandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connection`: MySqlConnection object to use
- `commandText`: SQL command to be executed
- `commandParameters`: Array of MySqlParameter objects to use with the command.

Return Value

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteNonQuery Overload List

MySqlHelper.ExecuteNonQuery Method (String, String, MySqlParameter[])

Executes a single command against a MySQL database. A new MySqlConnection is created using the ConnectionString given.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray parms As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
stringconnectionString,
```

```
stringcommandText,
   params MySqlParameter[]parms
);
```

Parameters

- `connectionString`: ConnectionString to use
- `commandText`: SQL command to be executed
- `parms`: Array of MySqlParameter objects to use with the command.

Return Value

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteNonQuery Overload List

ExecuteReader Method

Executes a single command against a MySQL database.

Overload List

Executes a single command against a MySQL database.

- public static MySqlDataReader ExecuteReader(string,string);

Executes a single command against a MySQL database.

- public static MySqlDataReader ExecuteReader(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteReader Method (String, String)

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
   ByVal connectionString As String, _
   ByVal commandText As String _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
stringconnectionString,
stringcommandText
);
```

Parameters

- `connectionString`: Settings to use for this command
- `commandText`: Command text to use

Return Value

MySqlDataReader object ready to read the results of the command

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteReader Overload List

MySqlHelper.ExecuteReader Method (String, String, MySqlParameter[])

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
stringconnectionString,
stringcommandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connectionString`: Settings to use for this command
- `commandText`: Command text to use
- `commandParameters`: Array of MySqlParameter objects to use with the command

Return Value

MySqlDataReader object ready to read the results of the command

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteReader Overload List

ExecuteScalar Method

Execute a single command against a MySQL database.

Overload List

Execute a single command against a MySQL database.

- public static object ExecuteScalar(MySqlConnection,string);

Execute a single command against a MySQL database.

- public static object ExecuteScalar(MySqlConnection,string,params MySqlParameter[]);

Execute a single command against a MySQL database.

- public static object ExecuteScalar(string,string);

Execute a single command against a MySQL database.

- public static object ExecuteScalar(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteScalar Method (MySqlConnection, String)

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
MySqlConnectionconnection,
stringcommandText
);
```

Parameters

- `connection`: MySqlConnection object to use
- `commandText`: Command text to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteScalar Overload List

MySqlHelper.ExecuteScalar Method (MySqlConnection, String, MySqlParameter[])

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
MySqlConnectionconnection,
stringcommandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connection`: MySqlConnection object to use

- `commandText`: Command text to use for the command
- `commandParameters`: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteScalar Overload List

MySqlHelper.ExecuteScalar Method (String, String)

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connectionString As String, _
   ByVal commandText As String _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
stringconnectionString,
stringcommandText
);
```

Parameters

- `connectionString`: Settings to use for the update
- `commandText`: Command text to use for the update

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteScalar Overload List

MySqlHelper.ExecuteScalar Method (String, String, MySqlParameter[])

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
stringconnectionString,
stringcommandText,
   params MySqlParameter[]commandParameters
```

```
);
```

Parameters

- `connectionString`: Settings to use for the command
- `commandText`: Command text to use for the command
- `commandParameters`: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace , MySqlHelper.ExecuteScalar Overload List

MySqlHelper.UpdateDataSet Method

Updates the given table with data from the given DataSet

Syntax: Visual Basic

```
Public Shared Sub UpdateDataSet( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ByVal ds As DataSet, _
   ByVal tablename As String _
)
```

Syntax: C#

```
public static void UpdateDataSet(
stringconnectionString,
stringcommandText,
DataSetds,
stringtablename
);
```

Parameters

- `connectionString`: Settings to use for the update
- `commandText`: Command text to use for the update
- `ds`: DataSetcontaining the new data to use in the update
- `tablename`: Tablename in the dataset to update

See Also

MySqlHelper Class , MySql.Data.MySqlClient Namespace

MySqlErrorCode Enumeration

Syntax: Visual Basic

```
Public Enum MySqlErrorCode
```

Syntax: C#

```
public enum MySqlErrorCode
```

Members

Member Name	Description
PacketTooLarge	
PasswordNotAllowed	
DuplicateKeyEntry	
HostNotPrivileged	
PasswordNoMatch	
AnonymousUser	
DuplicateKey	
KeyNotFound	
DuplicateKeyName	

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

25.2.4.2. MySql.Data.Types

Namespace hierarchy

Classes

Class	Description
MySqlConversionException	Summary description for MySqlConversionException.
MySqlDateTime	Summary description for MySqlDateTime.
MySqlValue	

MySql.Data.TypesHierarchy

See Also

MySql.Data.Types Namespace

MySqlConversionException Class

Summary description for MySqlConversionException.

For a list of all members of this type, see MySqlConversionException Members .

Syntax: Visual Basic

```
Public Class MySqlConversionException_
```

```
  Inherits ApplicationException
```

Syntax: C#

```
public class MySqlConversionException : ApplicationException
```

Thread Safety

Public static (Sharedin Visual Basic) members of this type are safe for multithreaded operations. Instance members are notguaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.Types

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlConversionException Members , MySql.Data.Types Namespace

MySqlConversionException Members

MySqlConversionException overview

Public Instance Constructors

MySqlConversionException Constructor	Ctor

Public Instance Properties

Data(inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink(inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException(inherited from Exception)	Gets the Exceptioninstance that caused the current exception.
Message(inherited from Exception)	Gets a message that describes the current exception.
Source(inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace(inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite(inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetBaseException(inherited from Exception)	When overridden in a derived class, returns the Exceptionthat is the root cause of one or more subsequent exceptions.

GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData(inherited from Exception)	When overridden in a derived class, sets the SerializationInfowith information about the exception.
GetType(inherited from Exception)	Gets the runtime type of the current instance.
ToString(inherited from Exception)	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult(inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Objectto attempt to free resources and perform other cleanup operations before the Objectis reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlConversionException Class , MySql.Data.Types Namespace

MySqlConversionException Constructor

Syntax: Visual Basic

```
Public Sub New( _
   ByVal msg As String _
)
```

Syntax: C#

```
public MySqlConversionException(
stringmsg
);
```

See Also

MySqlConversionException Class , MySql.Data.Types Namespace

MySqlDateTime Class

Summary description for MySqlDateTime.

For a list of all members of this type, see MySqlDateTime Members .

Syntax: Visual Basic

```
Public Class MySqlDateTime_
  Inherits MySqlValue_
  Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySqlDateTime : MySqlValue, IConvertible, IComparable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.Types

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlDateTime Members , MySql.Data.Types Namespace

MySqlDateTime Members

MySqlDateTime overview

Public Static (Shared) Type Conversions

Explicit MySqlDateTime to DateTime Conversion	

Public Instance Properties

Day	Returns the day portion of this datetime
Hour	Returns the hour portion of this datetime
IsNull (inherited from MySqlValue)	
IsValidDateTime	Indicates if this object contains a value that can be represented as a DateTime
Minute	Returns the minute portion of this datetime
Month	Returns the month portion of this datetime
Second	Returns the second portion of this datetime
ValueAsObject (inherited from MySqlValue)	Returns the value of this field as an object
Year	Returns the year portion of this datetime

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetDateTime	Returns this value as a DateTime
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString	Returns a MySQL specific string representation of this value

Protected Instance Fields

classType (inherited from MySqlValue)	The system type represented by this value
dbType (inherited from MySqlValue)	The generic dbtype of this value
isNull (inherited from MySqlValue)	Is this value null
mySqlDbType (inherited from MySqlValue)	The specific MySQL db type
mySqlTypeName (inherited from MySqlValue)	The MySQL specific typename of this value
objectValue (inherited from MySqlValue)	

Protected Instance Methods

Finalize(inherited from Object)	Allows an Objectto attempt to free resources and perform other cleanup operations before the Objectis reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

MySqlDateTime Explicit MySqlDateTime to DateTime Conversion

Syntax: Visual Basic

```
MySqlDateTime.op_Explicit(val)
```

Syntax: C#

```
public static explicit operator DateTime(
MySqlDateTimeval
);
```

Parameters

- `val`:

Return Value

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

Day Property

Returns the day portion of this datetime

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#

```
public int Day {get; set;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

Hour Property

Returns the hour portion of this datetime

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue Class

For a list of all members of this type, see MySqlValue Members .

Syntax: Visual Basic

```
MustInherit Public Class MySqlValue
```

Syntax: C#

```
public abstract class MySqlValue
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.Types

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlValue Members , MySql.Data.Types Namespace

MySqlValue Members

MySqlValue overview

Protected Static (Shared) Fields

numberFormat	

Public Instance Constructors

MySqlValue Constructor	Initializes a new instance of the MySqlValue class.

Public Instance Properties

IsNull	
ValueAsObject	Returns the value of this field as an object

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString	Returns a string representation of this value

Protected Instance Fields

classType	The system type represented by this value
dbType	The generic dbtype of this value
isNull	Is this value null
mySqlDbType	The specific MySQL db type
mySqlTypeName	The MySQL specific typename of this value
objectValue	

Protected Instance Methods

Finalize(inherited from Object)	Allows an Objectto attempt to free resources and perform other cleanup operations before the Objectis reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.numberFormat Field

Syntax: Visual Basic

```
Protected Shared numberFormat As NumberFormatInfo
```

Syntax: C#

```
protected static NumberFormatInfo numberFormat;
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue Constructor

Initializes a new instance of the MySqlValue class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlValue();
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

ValueAsObject Property

Returns the value of this field as an object

Syntax: Visual Basic

```
Public ReadOnly Property ValueAsObject As Object
```

Syntax: C#

```
public object ValueAsObject {get;}
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.ToString Method

Returns a string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.classType Field

The system type represented by this value

Syntax: Visual Basic

```
Protected classType As Type
```

Syntax: C#

```
protected Type classType;
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.dbType Field

The generic dbtype of this value

Syntax: Visual Basic

```
Protected dbType As DbType
```

Syntax: C#

```
protected DbType dbType;
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.mySqlDbType Field

The specific MySQL db type

Syntax: Visual Basic

```
Protected mySqlDbType As MySqlDbType
```

Syntax: C#

```
protected MySqlDbType mySqlDbType;
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.mySqlTypeName Field

The MySQL specific typename of this value

Syntax: Visual Basic

```
Protected mySqlTypeName As String
```

Syntax: C#

```
protected string mySqlTypeName;
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

MySqlValue.objectValue Field

Syntax: Visual Basic

```
Protected objectValue As Object
```

Syntax: C#

```
protected object objectValue;
```

See Also

MySqlValue Class , MySql.Data.Types Namespace

IsValidDateTime Property

Indicates if this object contains a value that can be represented as a DateTime

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

Minute Property

Returns the minute portion of this datetime

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

Month Property

Returns the month portion of this datetime

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

Second Property

Returns the second portion of this datetime

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

Year Property

Returns the year portion of this datetime

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

MySqlDateTime.GetDateTime Method

Returns this value as a DateTime

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

MySqlDateTime.ToString Method

Returns a MySQL specific string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

MySqlDateTime Class , MySql.Data.Types Namespace

25.2.5. Connector/NET Notes and Tips

In this section we will cover some of the more common use cases for Connector/NET, including BLOB handling, date handling, and using Connector/NET with common tools such as Crystal Reports.

25.2.5.1. Connecting to MySQL Using Connector/NET

Introduction

All interaction between a .NET application and the MySQL server is routed through a `MySqlConnection` object. Before your application can interact with the server, a `MySqlConnection` object must be instanced, configured, and opened.

Even when using the `MySqlHelper` class, a `MySqlConnection` object is created by the helper class.

In this section, we will describe how to connect to MySQL using the `MySqlConnection` object.

Creating a Connection String

The `MySqlConnection` object is configured using a connection string. A connection string contains sever key/value pairs, separated by semicolons. Each key/value pair is joined with an equals sign.

The following is a sample connection string:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

In this example, the `MySqlConnection` object is configured to connect to a MySQL server at `127.0.0.1`, with a username of `root` and a password of `12345`. The default database for all statements will be the `test` database.

The following options are typically used (a full list of options is available in the API documentation for "ConnectionString"):

- `Server`: The name or network address of the instance of MySQL to which to connect. The default is `localhost`. Aliases include `host`, `Data Source`, `DataSource`, `Address`, `Addr` and `Network Address`.
- `Uid`: The MySQL user account to use when connecting. Aliases include `User Id`, `Username` and `User name`.
- `Pwd`: The password for the MySQL account being used. Alias `Password` can also be used.
- `Database`: The default database that all statements are applied to. Default is `mysql`. Alias `Initial Catalog` can also be used.
- `Port`: The port MySQL is using to listen for connections. Default is `3306`. Specify `-1` for this value to use a named-pipe connection.

Opening a Connection

Once you have created a connection string it can be used to open a connection to the MySQL server.

The following code is used to create a `MySqlConnection` object, assign the connection string, and open the connection.

Visual Basic Example

```
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
```

```
            & "uid=root;" _
            & "pwd=12345;" _
            & "database=test;"

Try
  conn.ConnectionString = myConnectionString
  conn.Open()

Catch ex As MySql.Data.MySqlClient.MySqlException
  MessageBox.Show(ex.Message)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection();
    conn.ConnectionString = myConnectionString;
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

You can also pass the connection string to the constructor of the `MySqlConnection` class:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
              & "uid=root;" _
              & "pwd=12345;" _
              & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
   MessageBox.Show(ex.Message)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
```

```
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

Once the connection is open it can be used by the other Connector/NET classes to communicate with the MySQL server.

Handling Connection Errors

Because connecting to an external server is unpredictable, it is important to add error handling to your .NET application. When there is an error connecting, the `MySqlConnection` class will return a `MySqlException` object. This object has two properties that are of interest when handling errors:

- `Message`: A message that describes the current exception.
- `Number`: The MySQL error number.

When handling errors, you can your application's response based on the error number. The two most common error numbers when connecting are as follows:

- `0`: Cannot connect to server.
- `1045`: Invalid username and/or password.

The following code shows how to adapt the application's response based on the actual error:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
          & "uid=root;" _
          & "pwd=12345;" _
          & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact administrator")
        Case 1045
            MessageBox.Show("Invalid username/password, please try again")
    End Select
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
```

```
}
    catch (MySql.Data.MySqlClient.MySqlException ex)
{
    switch (ex.Number)
    {
        case 0:
            MessageBox.Show("Cannot connect to server.  Contact administrator");
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
    }
}
```

Important: Note that if you are using multilanguage databases you must specify the character set in the connection string. If you do not specify the character set, the connection defaults to the `latin1` charset. You can specify the character set as part of the connection string, for example:

```
MySqlConnection myConnection = new MySqlConnection("server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;Charset=latin1;");
```

25.2.5.2. Using the Connector/NET with Prepared Statements

Introduction

As of MySQL 4.1, it is possible to use prepared statements with Connector/NET. Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

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.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

Preparing Statements in Connector/NET

To prepare a statement, create a command object and set the `.CommandText` property to your query.

After entering your statement, call the `.Prepare` method of the `MySqlCommand` object. After the statement is prepared, add parameters for each of the dynamic elements in the query.

After you enter your query and enter parameters, execute the statement using the `.ExecuteNonQuery()`, `.ExecuteScalar()`, or `.ExecuteReader` methods.

For subsequent executions, you need only modify the values of the parameters and call the execute method again, there is no need to set the `.CommandText` property or redefine the parameters.

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = strConnection

Try
   conn.Open()
   cmd.Connection = conn
```

```
   cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)"
   cmd.Prepare()

   cmd.Parameters.Add("?number", 1)
   cmd.Parameters.Add("?text", "One")

   For i = 1 To 1000
       cmd.Parameters["?number"].Value = i
       cmd.Parameters["?text"].Value = "A string value"

       cmd.ExecuteNonQuery()
     Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, Mess
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = strConnection;

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)";
    cmd.Prepare();

    cmd.Parameters.Add("?number", 1);
    cmd.Parameters.Add("?text", "One");

    for (int i=1; i <= 1000; i++)
    {
        cmd.Parameters["?number"].Value = i;
        cmd.Parameters["?text"].Value = "A string value";

        cmd.ExecuteNonQuery();
    }
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

25.2.5.3. Accessing Stored Procedures with Connector/NET

Introduction

With the release of MySQL version 5 the MySQL server now supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure 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 procedure instead.

Stored procedures can be particularly useful in situations such as the following:

- 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 for all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Connector/NET supports the calling of stored procedures through the `MySqlCommand` object. Data can be passed in and our of a MySQL stored procedure through use of the `MySqlCommand.Parameters` collection.

Note: When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to http://dev.mysql.com/doc/mysql/en/stored-procedures.html.

A sample application demonstrating how to use stored procedures with Connector/NET can be found in the `Samples` directory of your Connector/NET installation.

Creating Stored Procedures from Connector/NET

Stored procedures in MySQL can be created using a variety of tools. First, stored procedures can be created using the `mysql` command-line client. Second, stored procedures can be created using the `MySQL Query Browser` GUI client. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySqlCommand` object:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "CREATE PROCEDURE add_emp(" _
        & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT) " _
        & "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " _
        & "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END"

    cmd.ExecuteNonQuery()
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK,
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
```

```
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "CREATE PROCEDURE add_emp(" +
        "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT) " +
        "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
        "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END";

    cmd.ExecuteNonQuery();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
    "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

It should be noted that, unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/NET.

Calling a Stored Procedure from Connector/NET

To call a stored procedure using Connector/NET, create a `MySqlCommand` object and pass the stored procedure name as the `.CommandText` property. Set the `.CommandType` property to `CommandType.StoredProcedure`.

After the stored procedure is named, create one `MySqlCommand` parameter for every parameter in the stored procedure. `IN` parameters are defined with the parameter name and the object containing the value, `OUT` parameters are defined with the parameter name and the datatype that is expected to be returned. All parameters need the parameter direction defined.

After defining parameters, call the stored procedure by using the `MySqlCommand.ExecuteNonQuery()` method:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "add_emp"
    cmd.CommandType = CommandType.StoredProcedure

    cmd.Parameters.Add("?lname", 'Jones')
    cmd.Parameters["?lname"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("?fname", 'Tom')
    cmd.Parameters["?fname"].Direction = ParameterDirection.Input
```

```
    cmd.Parameters.Add("?bday", #12/13/1977 2:17:36 PM#)
    cmd.Parameters["?bday"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("?empno", MySqlDbType.Int32)
    cmd.Parameters["?empno"].Direction = ParameterDirection.Output

    cmd.ExecuteNonQuery()

    MessageBox.Show(cmd.Parameters["?empno"].Value)
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, I
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "add_emp";
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("?lname", "Jones");
    cmd.Parameters["?lname"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?fname", "Tom");
    cmd.Parameters["?fname"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?bday", DateTime.Parse("12/13/1977 2:17:36 PM"));
    cmd.Parameters["?bday"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?empno", MySqlDbType.Int32);
    cmd.Parameters["?empno"].Direction = ParameterDirection.Output;

    cmd.ExecuteNonQuery();

    MessageBox.Show(cmd.Parameters["?empno"].Value);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
      "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Once the stored procedure is called, the values of output parameters can be retrieved by using the `.Value` property of the `MySqlConnector.Parameters` collection.

25.2.5.4. Handling BLOB Data With Connector/NET

Introduction

One common use for MySQL is the storage of binary data in `BLOB` columns. MySQL supports four different BLOB datatypes: `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, and `LONGBLOB`.

Data stored in a BLOB column can be accessed using Connector/NET and manipulated using client-side code. There are no special requirements for using Connector/NET with BLOB data.

Simple code examples will be presented within this section, and a full sample application can be found in the `Samples` directory of the Connector/NET installation.

Preparing the MySQL Server

The first step is using MySQL with BLOB data is to configure the server. Let's start by creating a table to be accessed. In my file tables, I usually have four columns: an AUTO_INCREMENT column of appropriate size (UNSIGNED SMALLINT) to serve as a primary key to identify the file, a VARCHAR column that stores the filename, an UNSIGNED MEDIUMINT column that stores the size of the file, and a MEDIUMBLOB column that stores the file itself. For this example, I will use the following table definition:

```
CREATE TABLE file(
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
file_name VARCHAR(64) NOT NULL,
file_size MEDIUMINT UNSIGNED NOT NULL,
file MEDIUMBLOB NOT NULL);
```

After creating a table, you may need to modify the max_allowed_packet system variable. This variable determines how large of a packet (i.e. a single row) can be sent to the MySQL server. By default, the server will only accept a maximum size of 1 meg from our client application. If you do not intend to exceed 1 meg, this should be fine. If you do intend to exceed 1 meg in your file transfers, this number has to be increased.

The max_allowed_packet option can be modified using MySQL Administrator's Startup Variables screen. Adjust the Maximum allowed option in the Memory section of the Networking tab to an appropriate setting. After adjusting the value, click the Apply Changes button and restart the server using the `Service Control` screen of MySQL Administrator. You can also adjust this value directly in the my.cnf file (add a line that reads max_allowed_packet=xxM), or use the SET max_allowed_packet=xxM; syntax from within MySQL.

Try to be conservative when setting max_allowed_packet, as transfers of BLOB data can take some time to complete. Try to set a value that will be adequate for your intended use and increase the value if necessary.

Writing a File to the Database

To write a file to a database we need to convert the file to a byte array, then use the byte array as a parameter to an `INSERT` query.

The following code opens a file using a FileStream object, reads it into a byte array, and inserts it into the `file` table:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

Dim SQL As String

Dim FileSize As UInt32
Dim rawData() As Byte
Dim fs As FileStream
```

```
conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)
    FileSize = fs.Length

    rawData = New Byte(FileSize) {}
    fs.Read(rawData, 0, FileSize)
    fs.Close()

    conn.Open()

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File)"

    cmd.Connection = conn
    cmd.CommandText = SQL
    cmd.Parameters.Add("?FileName", strFileName)
    cmd.Parameters.Add("?FileSize", FileSize)
    cmd.Parameters.Add("?File", rawData)

    cmd.ExecuteNonQuery()

    MessageBox.Show("File Inserted into database successfully!", _
    "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.Read);
    FileSize = fs.Length;

    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();

    conn.Open();

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File)";

    cmd.Connection = conn;
    cmd.CommandText = SQL;
```

```
    cmd.Parameters.Add("?FileName", strFileName);
    cmd.Parameters.Add("?FileSize", FileSize);
    cmd.Parameters.Add("?File", rawData);

    cmd.ExecuteNonQuery();

    MessageBox.Show("File Inserted into database successfully!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

The `Read` method of the `FileStream` object is used to load the file into a byte array which is sized according to the `Length` property of the FileStream object.

After assigning the byte array as a parameter of the `MySqlCommand` object, the `ExecuteNonQuery` method is called and the BLOB is inserted into the `file` table.

Reading a BLOB from the Database to a File on Disk

Once a file is loaded into the `file` table, we can use the `MySqlDataReader` class to retrieve it.

The following code retrieves a row from the `file` table, then loads the data into a `FileStream` object to be written to disk:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

SQL = "SELECT file_name, file_size, file FROM file"

Try
    conn.Open()

    cmd.Connection = conn
    cmd.CommandText = SQL

    myData = cmd.ExecuteReader

    If Not myData.HasRows Then Throw New Exception("There are no BLOBs to save")

    myData.Read()

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
    rawData = New Byte(FileSize) {}

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize)
```

```
    fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write)
    fs.Write(rawData, 0, FileSize)
    fs.Close()

    MessageBox.Show("File successfully written to disk!", "Success!", MessageBoxButtons.OK, MessageBoxIcon

    myData.Close()
    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Err
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataReader myData;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

SQL = "SELECT file_name, file_size, file FROM file";

try
{
    conn.Open();

    cmd.Connection = conn;
    cmd.CommandText = SQL;

    myData = cmd.ExecuteReader();

    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");

    myData.Read();

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize);

    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write);
    fs.Write(rawData, 0, FileSize);
    fs.Close();

    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    myData.Close();
    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
}
```

After connecting, the contents of the `file` table are loaded into a `MySqlDataReader` object. The `GetBytes` method of the MySqlDataReader is used to load the BLOB into a byte array, which is then written to disk using a FileStream object.

The `GetOrdinal` method of the MySqlDataReader can be used to determine the integer index of a named column. Use of the GetOrdinal method prevents errors if the column order of the `SELECT` query is changed.

25.2.5.5. Using Connector/NET with Crystal Reports

Introduction

Crystal Reports is a common tool used by Windows application developers to perform reporting and document generation. In this section we will show how to use Crystal Reports XI with MySQL and Connector/NET.

Creating a Data Source

When creating a report in Crystal Reports there are two options for accessing the MySQL data while designing your report.

The first option is to use Connector/ODBC as an ADO data source when designing your report. You will be able to browse your database and choose tables and fields using drag and drop to build your report. The disadvantage of this approach is that additional work must be performed within your application to produce a dataset that matches the one expected by your report.

The second option is to create a dataset in VB.NET and save it as XML. This XML file can then be used to design a report. This works quite well when displaying the report in your application, but is less versatile at design time because you must choose all relevant columns when creating the dataset. If you forget a column you must re-create the dataset before the column can be added to the report.

The following code can be used to create a dataset from a query and write it to disk:

Visual Basic Example

```
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
```

```
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
  "pwd=12345;database=test;";

try
{
  cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
  "country.name, country.population, country.continent " +
  "FROM country, city ORDER BY country.continent, country.name";
  cmd.Connection = conn;

  myAdapter.SelectCommand = cmd;
  myAdapter.Fill(myData);

  myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
  MessageBox.Show(ex.Message, "Report could not be created",
  MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

The resulting XML file can be used as an ADO.NET XML datasource when designing your report.

If you choose to design your reports using Connector/ODBC, it can be downloaded from dev.mysql.com.

Creating the Report

For most purposes the Standard Report wizard should help with the initial creation of a report. To start the wizard, open Crystal Reports and choose the New > Standard Report option from the File menu.

The wizard will first prompt you for a data source. If you are using Connector/ODBC as your data source, use the OLEDB provider for ODBC option from the OLE DB (ADO) tree instead of the ODBC (RDO) tree when choosing a data source. If using a saved dataset, choose the ADO.NET (XML) option and browse to your saved dataset.

The remainder of the report creation process is done automatically by the wizard.

After the report is created, choose the Report Options... entry of the File menu. Un-check the Save Data With Report option. This prevents saved data from interfering with the loading of data within our application.

Displaying the Report

To display a report we first populate a dataset with the data needed for the report, then load the report and bind it to the dataset. Finally we pass the report to the crViewer control for display to the user.

The following references are needed in a project that displays a report:

- CrytalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a dataset saved using the code shown in “Creating a Data Source”, and have a crViewer control on your form named `myViewer`.

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()

    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.SetDataSource(myData)
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
```

```
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.SetDataSource(myData);
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

A new dataset it generated using the same query used to generate the previously saved dataset. Once the dataset is filled, a ReportDocument is used to load the report file and bind it to the dataset. The ReportDocument is the passed as the ReportSource of the crViewer.

This same approach is taken when a report is created from a single table using Connector/ODBC. The dataset replaces the table used in the report and the report is displayed properly.

When a report is created from multiple tables using Connector/ODBC, a dataset with multiple tables must be created in our application. This allows each table in the report data source to be replaced with a report in the dataset.

We populate a dataset with multiple tables by providing multiple `SELECT` statements in our MySqlCommand object. These `SELECT` statements are based on the SQL query shown in Crystal Reports in the Database menu's Show SQL Query option. Assume the following query:

```
SELECT `country`.`Name`, `country`.`Continent`, `country`.`Population`, `city`.`Name`, `city`.`Population`
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code`=`city`.`CountryC
ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`
```

This query is converted to two `SELECT` queries and displayed with the following code:

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
```

```
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER BY countrycode, name; " _
        & "SELECT name, population, code, continent FROM country ORDER BY continent, name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER " +
        "BY countrycode, name; SELECT name, population, code, continent FROM " +
        "country ORDER BY continent, name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

It is important to order the `SELECT` queries in alphabetical order, as this is the order the report will expect its source tables to be in. One SetDataSource statement is needed for each table in the report.

This approach can cause performance problems because Crystal Reports must bind the tables together on the client-side, which will be slower than using a pre-saved dataset.

25.2.5.6. Handling Date and Time Information in Connector/NET

Introduction

MySQL and the .NET languages handle date and time information differently, with MySQL allowing dates that cannot be represented by a .NET data type, such as '`0000-00-00 00:00:00`'. These differences can cause problems if not properly handled.

In this section we will demonstrate how to properly handle date and time information when using Connector/NET.

Problems when Using Invalid Dates

The differences in date handling can cause problems for developers who use invalid dates. Invalid MySQL dates cannot be loaded into native .NET `DateTime` objects, including `NULL` dates.

Because of this issue, .NET `DataSet` objects cannot be populated by the `Fill` method of the `MySqlDataAdapter` class as invalid dates will cause a `System.ArgumentOutOfRangeException` exception to occur.

Restricting Invalid Dates

The best solution to the date problem is to restrict users from entering invalid dates. This can be done on either the client or the server side.

Restricting invalid dates on the client side is as simple as always using the .NET `DateTime` class to handle dates. The `DateTime` class will only allow valid dates, ensuring that the values in your database are also valid. The disadvantage of this is that it is not useful in a mixed environment where .NET and non .NET code are used to manipulate the database, as each application must perform its own date validation.

Users of MySQL 5.0.2 and higher can use the new `traditional` SQL mode to restrict invalid date values. For information on using the `traditional` SQL mode, see Sección 5.3.2, “El modo SQL del servidor”.

Handling Invalid Dates

Although it is strongly recommended that you avoid the use of invalid dates within your .NET application, it is possible to use invalid dates by means of the `MySqlDateTime` datatype.

The `MySqlDateTime` datatype supports the same date values that are supported by the MySQL server. The default behavior of Connector/NET is to return a .NET DateTime object for valid date values, and return an error for invalid dates. This default can be modified to cause Connector/NET to return `MySqlDateTime` objects for invalid dates.

To instruct Connector/NET to return a `MySqlDateTime` object for invalid dates, add the following line to your connection string:

```
Allow Zero Datetime=True
```

Please note that the use of the `MySqlDateTime` class can still be problematic. The following are some known issues:

1. Data binding for invalid dates can still cause errors (zero dates like 0000-00-00 do not seem to have this problem).

2. The `ToString` method return a date formatted in the standard MySQL format (for example, `2005-02-23 08:50:25`). This differs from the `ToString` behavior of the .NET DateTime class.

3. The `MySqlDateTime` class supports NULL dates, while the .NET DateTime class does not. This can cause errors when trying to convert a MySQLDateTime to a DateTime if you do not check for NULL first.

Because of the known issues, the best recommendation is still to use only valid dates in your application.

Handling NULL Dates

The .NET `DateTime` datatype cannot handle `NULL` values. As such, when assigning values from a query to a `DateTime` variable, you must first check whether the value is in fact `NULL`.

When using a `MySqlDataReader`, use the `.IsDBNull` method to check whether a value is `NULL` before making the assignment:

Visual Basic Example

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

C# Example

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

`NULL` values will work in a dataset and can be bound to form controls without special handling.

25.2.6. Connector/NET Support

The developers of Connector/NET greatly value the input of our users in the software development process. If you find Connector/NET lacking some feature important to you, or if you discover a bug and need to file a bug report, please use the instructions in Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

25.2.6.1. Connector/NET Community Support

- Community support for Connector/NET can be found through the forums at http://forums.mysql.com.
- Community support for Connector/NET can also be found through the mailing lists at http://lists.mysql.com.
- Paid support is available from MySQL AB. Additional information is available at http://dev.mysql.com/support/.

25.2.6.2. How to report Connector/NET Problems or Bugs

If you encounter difficulties or problems with Connector/NET, contact the Connector/NET community Sección 25.2.6.1, “Connector/NET Community Support”.

You should first try to execute the same SQL statements and commands from the `mysql` client program or from `admndemo`. This helps you determine whether the error is in Connector/NET or MySQL.

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version
- Connector/NET version
- MySQL server version
- Copies of error messages or other unexpected output
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through http://bugs.mysql.com/.

25.2.6.3. Connector/NET Change History

The Connector/NET Change History (Changelog) is located with the main Changelog for MySQL. See Sección C.3, “Connector/NET Change History”.

25.3. MySQL Visual Studio Plugin

The MySQL Visual Studio Plugin is a DDEX provider; a plug-in for Visual Studio 2005 that allows developers to maintain database structures, and supports built-in data-driven application development tools.

The current version of the MySQL Visual Studio Pluginincludes only database maintenance tools. Data-driven application development tools are not supported.

The MySQL DDEX Provider operates as a standard extension to the Visual Studio Data Designer functionality available through the Server Explorer menu of Visual Studio 2005, and enables developers to create database objects and data within a MySQL database.

The MySQL Visual Studio Plugin is designed to work with MySQL version 5.0, but is also compatible with MySQL 4.1.1 and provides limited compatibility with MySQL 5.1.

25.3.1. Installing the MySQL Visual Studio Plugin

The MySQL Visual Studio Plugin requires Visual Studio 2005 Professional Edition to be installed, and therefore it has the same hardware and system requirements.

Here is the list of components that should already be installed before starting the installation of the MySQL Visual Studio Plugin:

- Visual Studio 2005 Standard, Professional or Team Developer Edition.
- MySQL Server 4.1.1 or later (either installed on the same machine, or a separate server).
- MySQL Connector.NET 5.0.

The user used to connect to the MySQL server must have the following privileges to use the functionality provided by the MySQL Visual Studio Plugin:

- The `SELECT` privilege for the `INFORMATION_SCHEMA` database.

- The `EXECUTE` privilege for the `SHOW CREATE TABLE` statement.
- The `SELECT` privilege for the `mysql.proc` table (required for operations with stored procedures and functions).
- The `SELECT` privilege for the `mysql.func` table (required for operations with User Defined Functions (UDF)).
- The `EXECUTE` privilege for the `SHOW ENGINE STATUS` statement (required for retrieving extended error information).
- Appropriate privileges for performed operations (e.g. the `SELECT` privilege is required to browse data from a table etc.).

The MySQL Visual Studio Plugin is delivered as a MSI package that can be used to install, uninstall or reinstall the Provider. If you are not using Windows XP or Windows Server 2003 you upgrade the Windows Installer system to the latest version (see http://support.microsoft.com/default.aspx?scid=kb;EN-US;292539 for details).

The MSI-package is named `MySQL.VisualStudio.msi`. To install the MySQL Visual Studio Plugin, right click on the MSI file and select Install. The installation process is as follow:

1. The standard Welcome dialog is opened. Click Next to continue installation.
2. The License agreement (GNU GPL) window is opened. Accept the agreement and click Next to continue.
3. The destination folder choice dialog is opened. Here you can point out the folder where the MySQL Visual Studio Plugin will be installed. The default destination folder is `%ProgramFilesDir%\MySQL \MySQL DDEX Data Provider`, where `%ProgramFilesDir%` is the Program Files folder of the installation machine. After choosing the destination folder, click Next to continue.
4. The installer will ask to confirm that installation. Click Install to start installation process.
5. The installation will now take place. At the end of this step the Visual Studio command table is rebuilt (this process may take several minutes).
6. Once installation is complete, click Finish to end the installation process.

To uninstall the MySQL Visual Studio Plugin, you can use either Add/Remove Programs component of the Control Panel or the same MSI-package. Choose the **Remove** option, and the Provider will be uninstalled automatically.

To repair the Provider, right click the MSI-package and choose the Repair option. The MySQL Visual Studio Plugin will be repaired automatically.

The installation package includes the following files:

- `MySQL.VisualStudio.dll` — the MySQL DDEX Provider assembly.
- `MySQL.Data.dll` — the assembly containing the MySQL Connector .NET which is used by the Provider.
- `MySql.VisualStudio.dll.config` — the configuration file for the MySQL Visual Studio Plugin. This file contains default values for the provider GUI layout.

> **Nota**
>
> Do not remove this file before the first use of the Provider.

- `Register.reg` — the file with registry entries that can be used to register the MySQL DDEX Provider in the case of the manual installation.
- `Install.js` — the script used to register the Connector .NET as an ADO.NET data provider in the machine.config file.
- `Release notes.doc` — the document with release notes.

To install the Provider manually, copy all files of the installation package in a desired folder, then set the full path to the Provider assembly as a value of the CodeBase entry. For example:

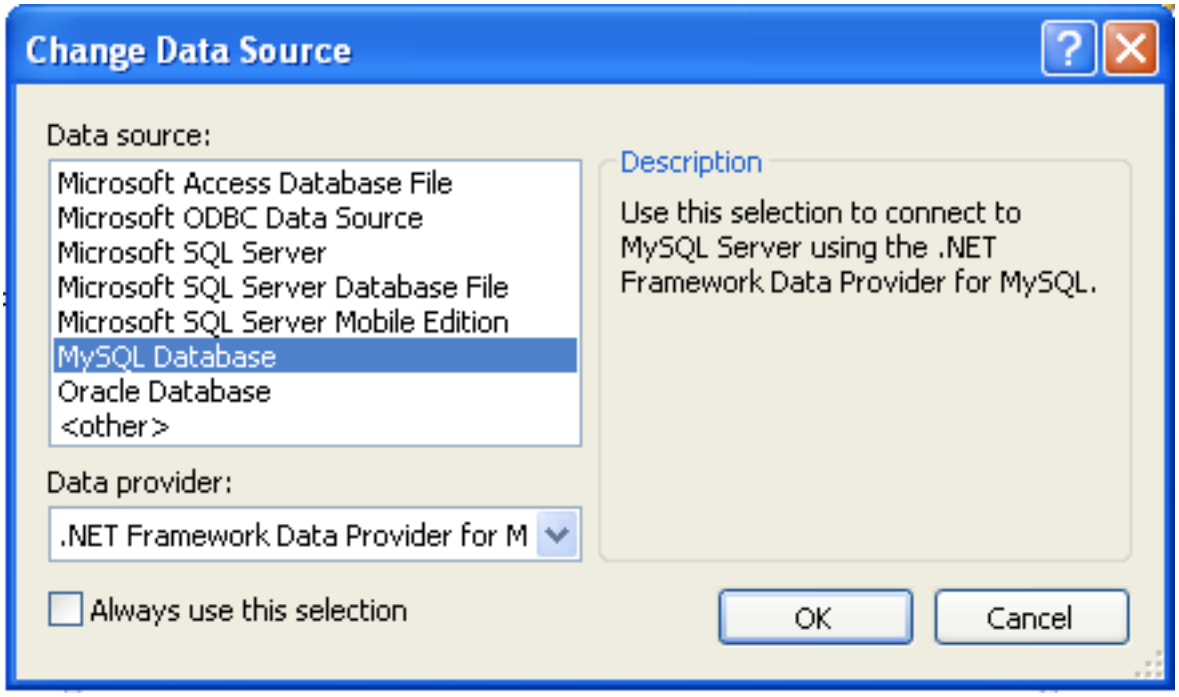
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Packages\{79A115C9-B133-4891-9E7B-242509DAD272}]@=
"InprocServer32"="C:\\WINNT\\system32\\mscoree.dll"
"Class"="MySql.Data.VisualStudio.MySqlDataProviderPackage"
"CodeBase"="C:\\MySqlDdexProvider\\MySql.VisualStudio.dll"
```

Then import information from the Register.reg file to the registry by clicking of the file. At the confirmation dialog choose Yes. Next you must run the command devenv.exe /setup within a Command Prompt to rebuild the Visual Studio command table.

25.3.2. Creating a connection to the MySQL server

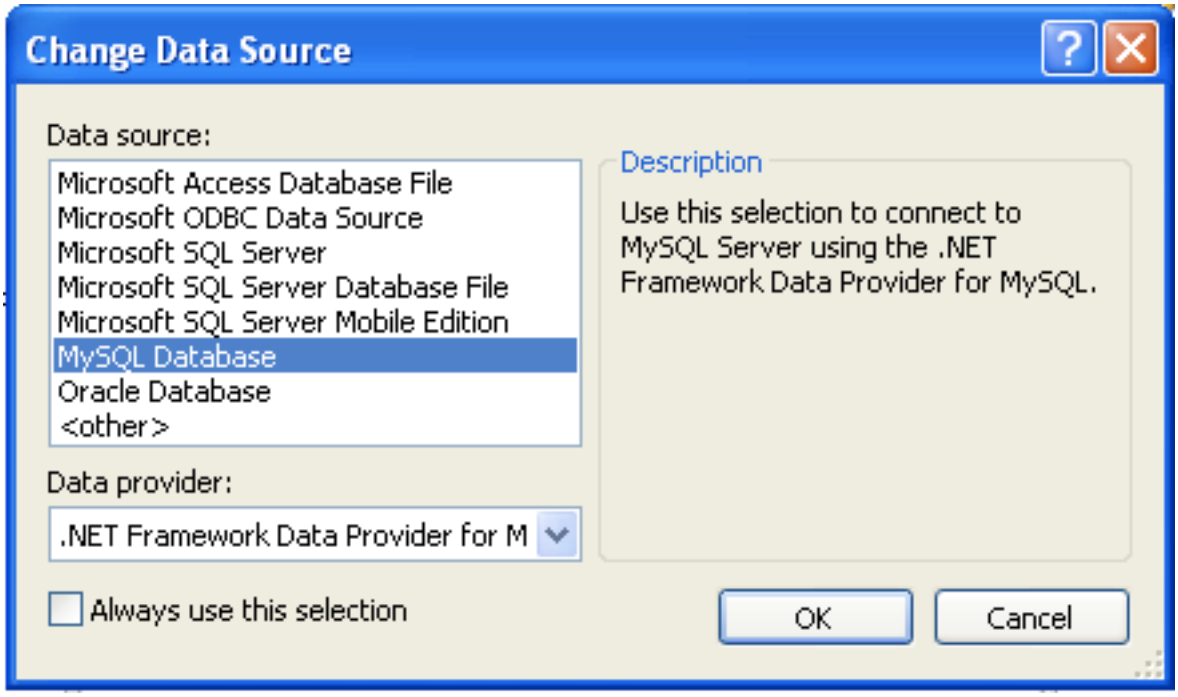
Once the MySQL Visual Studio Plugin is installed, you can use it to create, modify and delete connections to MySQL databases. To create a connection with a MySQL database, perform the following steps:

1. Start Visual Studio 2005 and open Server Explorer window by choosing the Server Explorer option from the View menu.
2. Right click on the **Data Connections** node and choose the Add Connection button.
3. The Add Connection dialog is opened. Press the Change button to choose MySQL Database as a data source.
4. Change Data Source dialog is opened. Choose MySQL Database in the list of data sources (or the `other` option, if MySQL Database is absent), and then choose **.NET Framework Data Provider for MySQL** in the combo box of data providers.



Press OK to confirm your choice.

5. Enter the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default database schema. Note that you must specify the default schema name to open the connection.



6. You can also set the port to connect with the MySQL server by pressing the Advanced button. To test a connection with the MySQL server, ser the server host name, the user name, and the password, and press the Test Connection button. If the test fails, check the connection values that you have supplied are correct and that the corresponding user and privileges have been configured on the MySQL server.

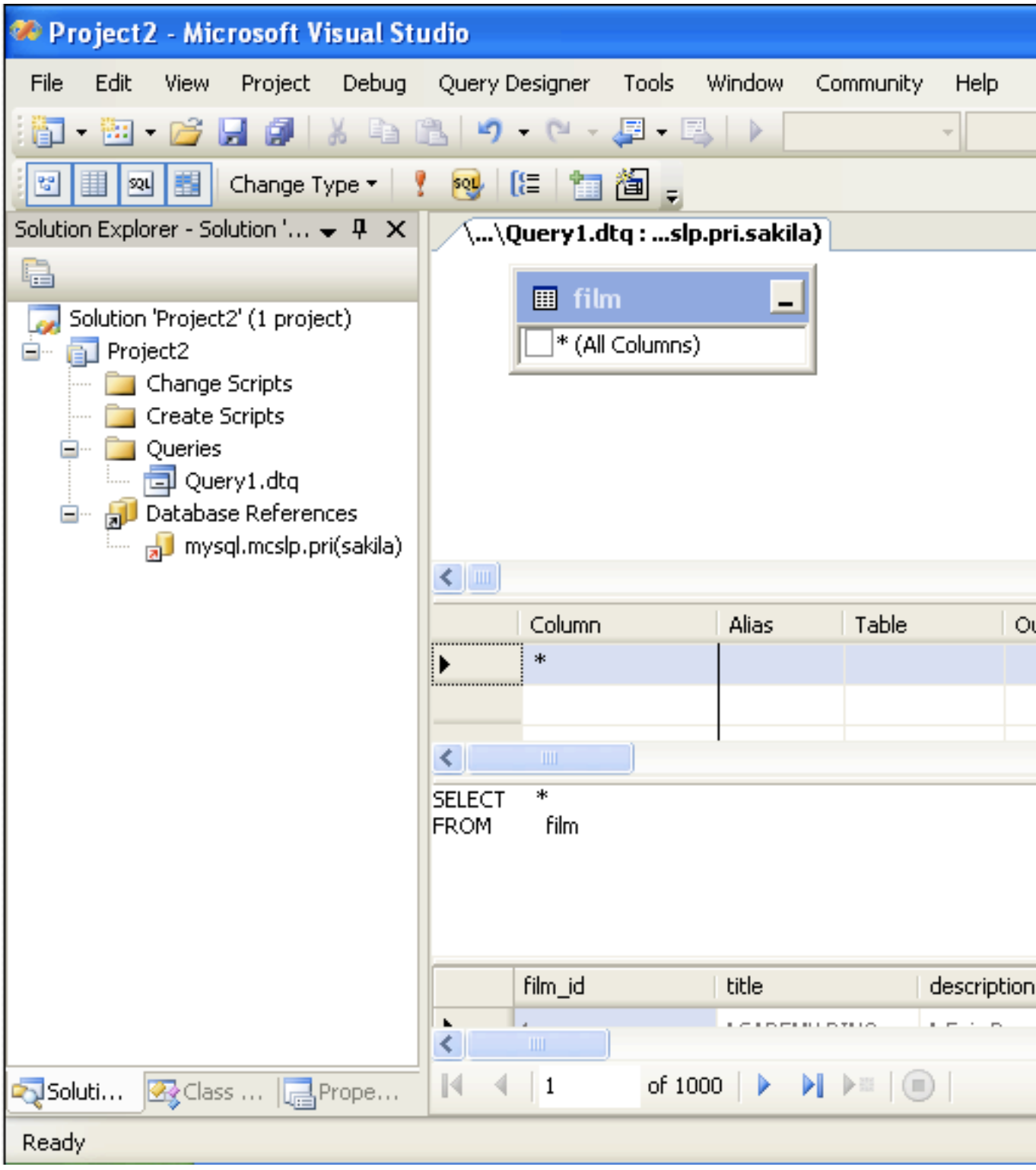
7. After you set all settings and test the connection, press OK. The newly created connection is displayed in Server Explorer. Now you can work with the MySQL server through standard Server Explorer interface.

After a connection is successfully established, all the connection settings are saved. When you next open Visual Studio, the connection to the MySQL server will appear within Server Explorer so that you can re-establish a connection to the MySQL server.

To modify and delete a connection, use the Server Explorer context menu for the corresponding node. You can modify any of the settings just by overwriting the existing values with new ones. Note that a connection should be modified or deleted only if no active editor for it's objects is opened. Otherwise your data could be lost.

25.3.3. Using the MySQL Visual Studio Plugin

To work with a MySQL server using the MySQL Visual Studio Plugin, open the Visual Studio 2005, open the **Server Explorer**, and select the required connection. The working area of the MySQL Visual Studio Plugin consists of three parts.

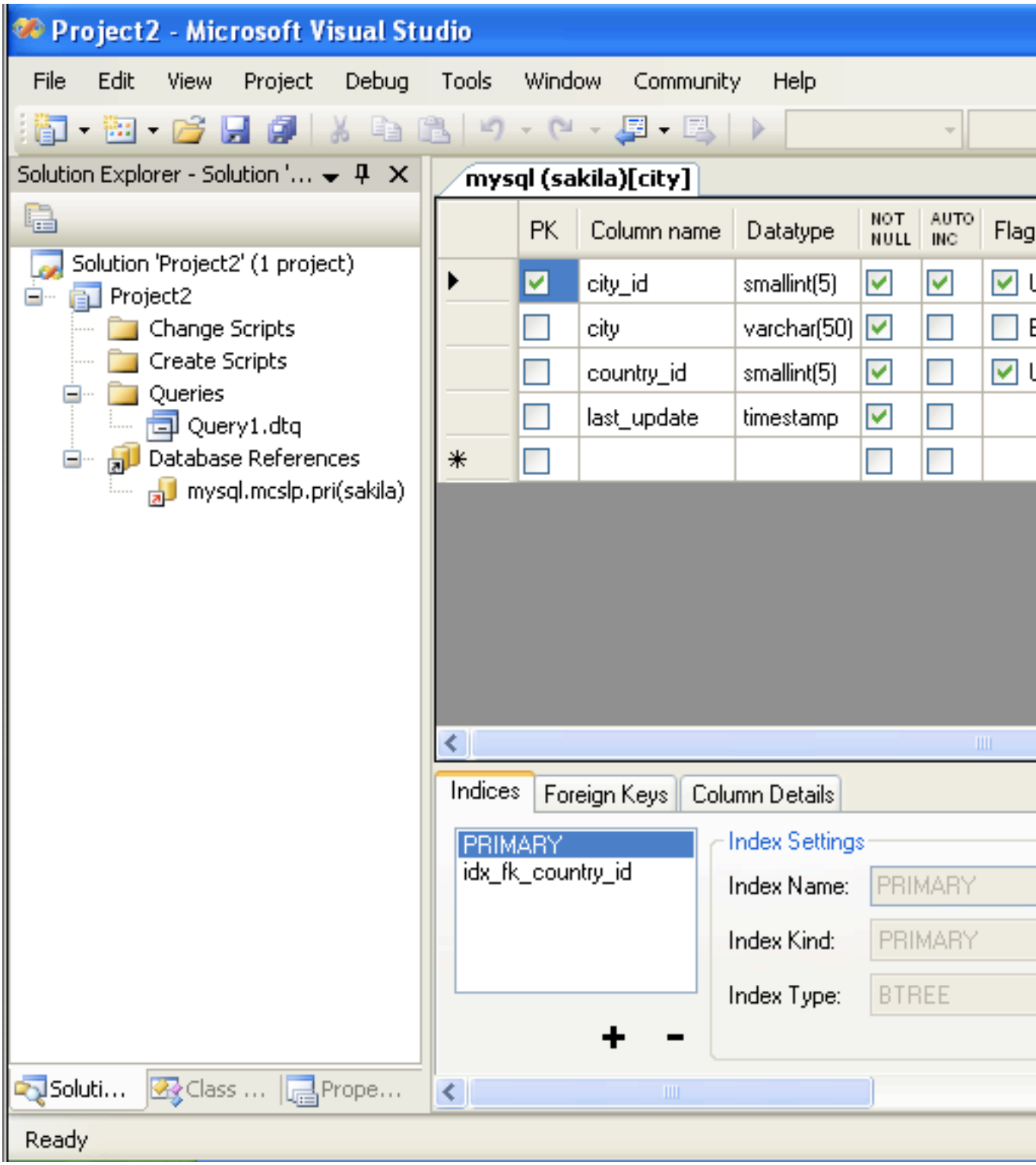


- Database objects (tables, views, stored routines, triggers, and user defined functions) are displayed in the Server Explorer tree. Here you can choose an object and edit its properties and definition.
- Properties of a selected database object are displayed in the **Properties** panel. Certain properties can be edited directly within this window.

- The editor panel provides direct access to the SQL statement and definition of specific objects. Fore example, the SQL statements within a stored procedure definition are shown and edited within this panel.

25.3.3.1. Editing Tables

The Table Editor can be accessed through a mouse action on table-type node of Server Explorer. To create a new table, right click on the **Tables** node (under the connection node) and choose the Create Table command from a context menu. To modify an existing table, double click on a node of the table you wish to modify, or right click on this node and choose the Alter Table command from a context menu. Either of the commands opens the Table Editor.



The MySQL Visual Studio Plugin Table Editor is implemented in a similar fashion to the standard Query Browser Table Editor, but with minor differences.

The Table Editor consists of the following parts:

- Columns Editor — for column creation, modification and deletion.

- Indexes tab — for table/column index management.
- Foreign Keys tab — for configuration of foreign keys.
- Column Details tab — used to set advanced column options.
- Properties window — used to set table properties.

To save changes you have made in the Table Editor, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. Before changes are saved, a confirmation dialog will be displayed to confirm that you want to update the corresponding object within the MySQL database.

Column Editor

You can use the Column Editor to set or change the name, data type, default value and other properties of a table column. To set the properties of an individual column, select the column using the mouse. Alternatively, you can move through the grid using **Tab** and **Shift+Tab** keys.

- To set or change the name, data type, default value and comment of a column, select the appropriate cell and edit the desired value.
- To set or unset flag-type column properties (i.e., primary key, `NOT NULL`, auto incremented, flags), check or uncheck the corresponding checkboxes. Note that the available column flags will depend on the columns data type.
- To reorder columns, index columns or foreign key columns in the Column Editor, select the whole column you wish to reorder by clicking on the selector column at the left of the column grid. Then move the column by using **Ctrl+Up** (to move the column up) and **Ctrl+Down** (to move the column down) keys.
- To delete a column, select it by clicking on the selector column at the left of the column grid, then press the **Delete** button on a keyboard.

Indexes tab

Index management is performed via the Indexes tab.

- To add an index, press the `+` button and set the properties in the **Index Settings** groupbox at the right. You can set the index name, index kind, index type and a set of index columns.
- To remove an index, select the index from the list and press the `-` button.
- To change index settings, select the index from the list; detailed information about the index is displayed in the **Index Settings** panel.

You cannot change a table column to an index column using drag and drop. Instead, you can add new index columns to a table and set their table columns by using the embedded editor within the Indexes tab

Foreign Keys tab

Foreign Key management is performed via the Foreign Keys tab.

- To add a foreign key, press the `+` button and set properties in the **Foreign Keys Settings** panel. You can set the foreign key name, referenced table name, foreign key columns and actions on update and delete.
- To remove a foreign key, select the foreign key and press the `-` button.
- To change foreign key settings, select the foreign key and use the **Foreign Keys Settings** panel to edit the properties.

- When a foreign key is changed, the MySQL Visual Studio Plugin generates two queries: the first query drops the changed keys and the second one recreates the new values. The reason for such a behavior is to avoid the Bug #8377 and Bug #8919.

> **Nota**
>
> If changed values are for some reason inconsistent and cause the second query to fail, all affected foreign keys will be dropped. If this is the case, the MySQL Visual Studio Plugin will mark them as new in the Table Editor, and you will have to recreate them later. But if you close the Table Editor without saving, these foreign keys will be lost.

Column Details tab

The Column Details tab can be used to set column options. Besides the main column properties that are presented in the Column Editor, in the Column Details tab you can set two additional properties options: the character set and the collation sequence.

Table Properties window

There is no separate tab for table options and advanced options. All table options can be browsed and changed using the **Properties** window of Visual Studio 2005.

The following table properties can be set:

- **Auto Increment**
- **Average Row Length**
- **Character Set**
- **Checksum for Rows**
- **Collation**
- **Comment**
- **Connection**
- **Data Directory**
- **Delay Key Updates**
- **Engine**
- **Index Directory**
- **Insert Method**
- **Maximum Rows**
- **Minimum Rows**
- **Name**
- **Pack Keys**
- **Password**
- **Row Format**

- **Union**

Some of these properties can have arbitrary text values, others accept values from a predefined set.

The properties **Schema** and **Server** are read only.

25.3.3.2. Editing Table Data

The Table Data Editor, allows a user to browse, create and edit data of tables. The Table Data Editor is implemented as a simple data grid with auto generated columns.

To access the Table Data Editor, right click on a node representing the table or view in Server Explorer. From the nodes context menu, choose the Browse or Edit Data command. For tables and updatable views, this command opens the Table Data Editor in edit mode. For non-updatable views, this command opens the Table Data Editor in read-only mode.

When in the edit mode, you can modify table data by modifying the displayed table contents directly. To add a row, set desired values in the last row of the grid. To modify values, set new values in appropriate cells. To delete a row, select it by clicking on the selector column at the left of the grid, then press the Delete button.

To save changes you have made in the Table Data Editor, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

25.3.3.3. Editing Views

To create a new view, right click the Views node under the connection node in Server Explorer. From the nodes context menu, choose the Create View command. This command opens the SQL Editor.

To modify an existing view, double click on a node of the view you wish to modify, or right click on this node and choose the Alter View command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the view definition using SQL Editor, type the appropriate SQL statement in the SQL Editor.

> **Nota**
>
> You should enter only the defining statement itself, without the `CREATE VIEW AS` preface.

All other view properties can be set in the **Properties** window. These properties are:

- **Algorithm**
- **Check Option**
- **Definer**
- **Name**
- **Security Type**

Some of these properties can have arbitrary text values, others accept values from a predefined set.

The properties **Is Updatable**, **Schema** and **Server** are readonly.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

25.3.3.4. Editing Stored Procedures and Functions

To create a new stored procedure, right click the Stored Procedures node under the connection node in Server Explorer. From the nodes context menu, choose the Create Routine command. This command opens the SQL Editor.

To create a new stored function, right click the **Functions** node under the connection node in Server Explorer. From the node's context menu, choose the Create Routine command.

To modify an existing stored routine (procedure or function), double click on a node of the routine you wish to modify, or right click on this node and choose the Alter Routine command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the routine definition using SQL Editor, type this definition in the SQL Editor using standard SQL.

All other routine properties can be set in the **Properties** window. These properties are:

- Comment
- Data Access
- Definer
- Is Deterministic
- Security Type

Some of these properties can have arbitrary text values, others accept values only from a predefined set.

Also you can set all the options directly in the SQL Editor, using the standard `CREATE PROCEDURE` or `CREATE FUNCTION` statement. However, it is recommended to use the **Properties** window instead.

> **Nota**
>
> You should never add the `CREATE` preface to the routine definition.

The properties **Name**, **Schema** and **Server** in the **Properties** window are read-only. Set or change the procedure name in the SQL editor.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database..

25.3.3.5. Editing Triggers

To create a new trigger, right click on a node of a table for which you wish to add a trigger. From the node's context menu, choose the Create Trigger command. This command opens the SQL Editor.

To modify an existing trigger, double click on a node of the trigger you wish to modify, or right click on this node and choose the Alter Trigger command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the trigger definition using SQL Editor, type the trigger statement in the SQL Editor using standard SQL.

> **Nota**
>
> You should enter only the trigger statement, that is the part of the `CREATE TRIGGER` query that is placed after the `FOR EACH ROW` clause.

All other trigger properties are set in the **Properties** window. These properties are:

- **Definer**
- **Event Manipulation**
- **Name**
- **Timing**

Some of these properties can have arbitrary text values, others accept values only from a predefined set.

The properties **Event Table**, **Schema** and **Server** in the **Properties** window are read-only.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

25.3.3.6. Editing User Defined Functions (UDF)

To create a new User Defined Function (UDF), right click the UDFs node under the connection node in Server Explorer. From the node's context menu, choose the Create UDF command. This command opens the UDF Editor.

To modify an existing UDF, double click on a node of the UDF you wish to modify, or right click on this node and choose the Alter UDF command from a context menu. Either of the commands opens the UDF Editor.

The UDF editor allows you to set the following properties through the properties panel:

- **Name**
- **So-name (DLL name)**
- **Return type**
- **Is Aggregate**

The property Server in the **Properties** window is read-only.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

25.3.3.7. Dropping database objects

Tables, views, stored routines, triggers, an UDFs can be dropped with the appropriate Drop command from its context menu: Drop Table, Drop View, Drop Routine, Drop Trigger, Drop UDF.

You will be asked to confirm the execution of the corresponding drop query in a confirmation dialog.

Dropping of multiple objects is not supported.

25.3.3.8. Cloning database objects

Tables, views, stored procedures and functions can be cloned with the appropriate Clone command from its context menu: Clone Table, Clone View, Clone Routine. The clone commands open the corresponding editor for a new object: the **Table Editor** for cloning a table and the SQL Editor for cloning a view or a routine.

To save the cloned object, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

25.3.4. Visual Studio Plugin Support

If you have a comment, or if you discover a bug, please, use our MySQL bug tracking system (http://bugs.mysql.com) to report problem or add your suggestion.

25.3.4.1. Visual Studio Plugin FAQ

Questions

- 25.3.4.1.1: [1463] When creating a connection, typing the connection details causes the connection window to immediately close.

Questions and Answers

25.3.4.1.1: When creating a connection, typing the connection details causes the connection window to immediately close.

There are known issues with versions of Connector/NET earlier than 5.0.2. Connector/NET 1.0.x is known not to work. If you have any of these versions installed, or have previously upgraded from an earlier version, uninstall Connector/NET completely and then install Connector/NET 5.0.2.

25.4. MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language via a JDBC driver, which is called MySQL Connector/J.

MySQL Connector/J is a JDBC-3.0 Type 4 driver, which means that is pure Java, implements version 3.0 of the JDBC specification, and communicates directly with the MySQL server using the MySQL protocol.

Although JDBC is useful by itself, we would hope that if you are not familiar with JDBC that after reading the first few sections of this manual, that you would avoid using naked JDBC for all but the most trivial problems and consider using one of the popular persistence frameworks such as Hibernate, Spring's JDBC templates or Ibatis SQL Maps to do the majority of repetitive work and heavier lifting that is sometimes required with JDBC.

This section is not designed to be a complete JDBC tutorial. If you need more information about using JDBC you might be interested in the following online tutorials that are more in-depth than the information presented here:

- JDBC Basics — A tutorial from Sun covering beginner topics in JDBC
- JDBC Short Course — A more in-depth tutorial from Sun and JGuru

25.4.1. Connector/J Versions

There are currently three versions of MySQL Connector/J available:

- Connector/J 3.0 provides core functionality and was designed with connectivity to MySQL 3.x or MySQL 4.1 servers, although it will provide basic compatibility with later versions of MySQL. Connector/J 3.0 does not support server-side prepared statements, and does not support any of the features in versions of MySQL later than 4.1.
- Connector/J 3.1 was designed for connectivity to MySQL 4.1 and MySQL 5.0 servers and provides support for all the functionality in MySQL 5.0 except distributed transaction (XA) support.

- Connector/J 5.0 provides support for all the functionality offered by Connector/J 3.1 and includes distributed transaction (XA) support.

The current recommended version for Connector/J is 5.0. This guide covers all three connector versions, with specific notes given where a setting applies to a specific option.

25.4.1.1. Java Versions Supported

MySQL Connector/J supports Java-2 JVMs, including:

- JDK 1.2.x (only for Connector/J 3.1.x or earlier)
- JDK 1.3.x
- JDK 1.4.x
- JDK 1.5.x

If you are building Connector/J from source using the source distribution (see Sección 25.4.2.4, “Installing from the Development Source Tree”) then you must use JDK 1.4.x or newer to compiler the Connector package.

MySQL Connector/J does not support JDK-1.1.x or JDK-1.0.x.

Because of the implementation of `java.sql.Savepoint`, Connector/J 3.1.0 and newer will not run on JDKs older than 1.4 unless the class verifier is turned off (by setting the `-Xverify:none` option to the Java runtime). This is because the class verifier will try to load the class definition for `java.sql.Savepoint` even though it is not accessed by the driver unless you actually use savepoint functionality.

Caching functionality provided by Connector/J 3.1.0 or newer is also not available on JVMs older than 1.4.x, as it relies on `java.util.LinkedHashMap` which was first available in JDK-1.4.0.

25.4.2. Connector/J Installation

You can install the Connector/J package using two methods, using either the binary or source distribution. The binary distribution provides the easiest methods for installation; the source distribution enables you to customize your installation further. With either solution, you must manually add the Connector/J location to your Java `CLASSPATH`.

25.4.2.1. Installing Connector/J from a Binary Distribution

The easiest method of installation is to use the binary distribution of the Connector/J package. The binary distribution is available either as a Tar/Gzip or Zip file which you must extract to a suitable location and then optionally make the information about the package available by changing your `CLASSPATH` (see Sección 25.4.2.2, “Installing the Driver and Configuring the CLASSPATH”).

MySQL Connector/J is distributed as a .zip or .tar.gz archive containing the sources, the class files, and the JAR archive named `mysql-connector-java-[version]-bin.jar`, and starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar`.

Starting with Connector/J 3.1.9, the `.class` files that constitute the JAR files are only included as part of the driver JAR file.

You should not use the debug build of the driver unless instructed to do so when reporting a problem ors bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

You will need to use the appropriate graphical or command-line utility to un-archive the distribution (for example, WinZip for the .zip archive, and `tar` for the .tar.gz archive). Because there are potentially long filenames in the distribution, we use the GNU tar archive format. You will need to use GNU tar (or an application that understands the GNU tar archive format) to unpack the .tar.gz variant of the distribution.

25.4.2.2. Installing the Driver and Configuring the `CLASSPATH`

Once you have extracted the distribution archive, you can install the driver by placing `mysql-connector-java-[version]-bin.jar` in your classpath, either by adding the full path to it to your `CLASSPATH` environment variable, or by directly specifying it with the command line switch -cp when starting your JVM.

If you are going to use the driver with the JDBC DriverManager, you would use `com.mysql.jdbc.Driver` as the class that implements java.sql.Driver.

You can set the `CLASSPATH` environment variableunder UNIX, Linux or Mac OS X either locally for a user within their `.profile`, `.login` or other login file. You can also set it globally by editing the global `/etc/profile` file.

For example, under a C shell (csh, tcsh) you would add the Connector/J driver to your `CLASSPATH` using the following:

```
shell> setenv CLASSPATH /path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Or with a Bourne-compatible shell (sh, ksh, bash):

```
export set CLASSPATH=/path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Within Windows 2000, Windows XP and Windows Server 2003, you must set the environment variable through the System control panel.

If you want to use MySQL Connector/J with an application server such as Tomcat or JBoss, you will have to read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the `CLASSPATH` environment variable. For configuration examples for some J2EE application servers, see Sección 25.4.5.2, “Using Connector/J with J2EE and Other Java Frameworks”. However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's .jar file in the WEB-INF/lib subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the MysqlDataSource or MysqlConnectionPoolDataSource classes in the `com.mysql.jdbc.jdbc2.optional` package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the `javax.sql.XADataSource` interface is implemented via the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

The various MysqlDataSource classes support the following parameters (through standard set mutators):

- user
- password
- serverName (see the previous section about fail-over hosts)
- databaseName
- port

25.4.2.3. Upgrading from an Older Version

MySQL AB tries to keep the upgrade process as easy as possible, however as is the case with any software, sometimes changes need to be made in new versions to support new features, improve existing functionality, or comply with new standards.

This section has information about what users who are upgrading from one version of Connector/J to another (or to a new version of the MySQL server, with respect to JDBC functionality) should be aware of.

Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes Unicode character sets, server-side prepared statements, SQLState codes returned in error messages by the server and various performance enhancements that can be enabled or disabled via configuration properties.

- **Unicode Character Sets** — See the next section, as well as Capítulo 10, *Soporte de conjuntos de caracteres*, for information on this new feature of MySQL. If you have something misconfigured, it will usually show up as an error with a message similar to `Illegal mix of collations`.

- **Server-side Prepared Statements** — Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

  Starting with version 3.1.7, the driver scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing emulateUnsupportedPstmts=false in your JDBC URL.

  If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the connection property useServerPrepStmts=false

- **Datetimes** with all-zero components (`0000-00-00 ...`) — These values can not be represented reliably in Java. Connector/J 3.0.x always converted them to NULL when being read from a ResultSet.

  Connector/J 3.1 throws an exception by default when these values are encountered as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the zeroDateTimeBehavior configuration property. The allowable values are:

  - `exception` (the default), which throws an SQLException with an SQLState of `S1009`.
  - `convertToNull`, which returns `NULL` instead of the date.
  - `round`, which rounds the date to the nearest closest value which is `0001-01-01`.

  Starting with Connector/J 3.1.7, `ResultSet.getString()` can be decoupled from this behavior via noDatetimeStringSync=true (the default value is `false`) so that you can get retrieve the unaltered all-zero value as a String. It should be noted that this also precludes using any time zone conversions, therefore the driver will not allow you to enable noDatetimeStringSync and useTimezone at the same time.

- **New SQLState Codes** — Connector/J 3.1 uses SQL:1999 SQLState codes returned by the MySQL server (if supported), which are different from the legacy X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return SQLStates as part

of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the configuration property useSqlStateCodes=false.

- **`ResultSet.getString()`** — Calling `ResultSet.getString()` on a BLOB column will now return the address of the byte[] array that represents it, instead of a String representation of the BLOB. BLOBs have no character set, so they can't be converted to java.lang.Strings without data loss or corruption.

  To store strings in MySQL with LOB behavior, use one of the TEXT types, which the driver will treat as a java.sql.Clob.

- **Debug builds** — Starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar` is shipped alongside the normal binary jar file that is named `mysql-connector-java-[version]-bin.jar`.

  Starting with Connector/J 3.1.9, we don't ship the .class files unbundled, they are only available in the JAR archives that ship with the driver.

  You should not use the debug build of the driver unless instructed to do so when reporting a problem or bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in latin1 tables on the server.

  Starting with MySQL-4.1, this functionality is deprecated. If you have applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, you should add the following property to your connection URL:

  ```
  useOldUTF8Behavior=true
  ```

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

  ```
  useServerPrepStmts=false
  ```

25.4.2.4. Installing from the Development Source Tree

Caution. You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/J up and running on your system, you should use a standard release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- Subversion, to check out the sources from our repository (available from http://subversion.tigris.org/).
- Apache Ant version 1.6 or newer (available from http://ant.apache.org/).
- JDK-1.4.2 or later. Although MySQL Connector/J can be installed on older JDKs, to compile it from source you must have at least JDK-1.4.2.

The Subversion source code repository for MySQL Connector/J is located at http://svn.mysql.com/svnpublic/connector-j. In general, you should not check out the entire repository because it contains every branch and tag for MySQL Connector/J and is quite large.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. At the time of this writing, there are three active branches of Connector/J: `branch_3_0`, `branch_3_1` and `branch_5_0`. Check out the latest code from the branch that you want with the following command (replacing *`[major]`* and *`[minor]`* with appropriate version numbers):

   ```
   shell> svn co »
   http://svn.mysql.com/svnpublic/connector-j/branches/branch_[major]_[minor]/connector-j
   ```

   This creates a `connector-j` subdirectory in the current directory that contains the latest sources for the requested branch.

2. Change location to the `connector-j` directory to make it your current working directory:

   ```
   shell> cd connector-j
   ```

3. Issue the following command to compile the driver and create a `.jar` file suitable for installation:

   ```
   shell> ant dist
   ```

   This creates a `build` directory in the current directory, where all build output will go. A directory is created in the `build` directory that includes the version number of the sources you are building from. This directory contains the sources, compiled `.class` files, and a `.jar` file suitable for deployment. For other possible targets, including ones that will create a fully packaged distribution, issue the following command:

   ```
   shell> ant --projecthelp
   ```

4. A newly created `.jar` file containing the JDBC driver will be placed in the directory `build/mysql-connector-java-[version]`.

   Install the newly created JDBC driver as you would a binary `.jar` file that you download from MySQL by following the instructions in Sección 25.4.2.2, “Installing the Driver and Configuring the CLASSPATH”.

25.4.3. Connector/J Examples

Examples of using Connector/J are located throughout this document, this section provides a summary and links to these examples.

- Ejemplo 25.9, “Retrieving `AUTO_INCREMENT` column values using `SELECT LAST_INSERT_ID()`”
- Ejemplo 25.10, “Retrieving `AUTO_INCREMENT` column values in `Updatable ResultSets`”
- Ejemplo 25.11, “Using a connection pool with a J2EE application server”
- Ejemplo 25.12, “Example of transaction with retry logic”

25.4.4. Connector/J (JDBC) Reference

This section of the manual contains reference material for MySQL Connector/J, some of which is automatically generated during the Connector/J build process.

25.4.4.1. Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J

The name of the class that implements java.sql.Driver in MySQL Connector/J is `com.mysql.jdbc.Driver`. The `org.gjt.mm.mysql.Driver` class name is also usable to remain backward-compatible with MM.MySQL. You should use this class name when registering the driver, or when otherwise configuring software to use MySQL Connector/J.

The JDBC URL format for MySQL Connector/J is as follows, with items in square brackets ([,]) being optional:

```
jdbc:mysql://[host][,failoverhost...][:port]/[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

If the hostname is not specified, it defaults to 127.0.0.1. If the port is not specified, it defaults to 3306, the default port number for MySQL servers.

```
jdbc:mysql://[host:port],[host:port].../[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

If the database is not specified, the connection will be made with no default database. In this case, you will need to either call the `setCatalog()` method on the Connection instance or fully-specify table names using the database name (i.e. `SELECT dbname.tablename.colname FROM dbname.tablename...`) in your SQL. Not specifying the database to use upon connection is generally only useful when building tools that work with multiple databases, such as GUI database managers.

MySQL Connector/J has fail-over support. This allows the driver to fail-over to any number of slave hosts and still perform read-only queries. Fail-over only happens when the connection is in an `autoCommit(true)` state, because fail-over can not happen reliably when a transaction is in progress. Most application servers and connection pools set `autoCommit` to `true` at the end of every transaction/connection use.

The fail-over functionality has the following behavior:

- If the URL property autoReconnect is false: Failover only happens at connection initialization, and failback occurs when the driver determines that the first host has become available again.
- If the URL property autoReconnect is true: Failover happens when the driver determines that the connection has failed (before *every* query), and falls back to the first host when it determines that the host has become available again (after `queriesBeforeRetryMaster` queries have been issued).

In either case, whenever you are connected to a "failed-over" server, the connection will be set to read-only state, so queries that would modify data will have exceptions thrown (the query will **never** be processed by the MySQL server).

Configuration properties define how Connector/J will make a connection to a MySQL server. Unless otherwise noted, properties can be set for a DataSource object or for a Connection object.

Configuration Properties can be set in one of the following ways:

- Using the set*() methods on MySQL implementations of java.sql.DataSource (which is the preferred method when using implementations of java.sql.DataSource):
  - com.mysql.jdbc.jdbc2.optional.MysqlDataSource
  - com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
- As a key/value pair in the java.util.Properties instance passed to `DriverManager.getConnection()` or `Driver.connect()`
- As a JDBC URL parameter in the URL given to `java.sql.DriverManager.getConnection()`, `java.sql.Driver.connect()` or the MySQL implementations of the `javax.sql.DataSource setURL()` method.

  Note. If the mechanism you use to configure a JDBC URL is XML-based, you will need to use the XML character literal & to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following tables.

Connection/Authentication.

Property Name	Definition	Default Value	Since Version
user	The user to connect as		all versions
password	The password to use when connecting		all versions
socketFactory	The name of the class that the driver should use for creating socket connections to the server. This class must implement the interface 'com.mysql.jdbc.SocketFactory' and have public no-args constructor.	com.mysql.jdbc.StandardSocketF	3.0.3
connectTimeout	Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to '0'.	0	3.0.1
socketTimeout	Timeout on network socket operations (0, the default means no timeout).	0	3.0.1
connectionLifecycleInterceptors	A comma-delimited list of classes that implement "com.mysql.jdbc.ConnectionLifecycleInterceptor" that should notified of connection lifecycle events (creation, destruction, commit, rollback, setCatalog and setAutoCommit) and potentially alter the execution of these commands. ConnectionLifecycleInterceptors are "stackable", more than one interceptor may be specified via the configuration property as a comma-delimited list, with the interceptors executed in order from left to right.		5.1.4

useConfigs	Load the comma-delimited list of configuration properties before parsing the URL or applying user-specified properties. These configurations are explained in the 'Configurations' of the documentation.		3.1.5
interactiveClient	Set the CLIENT_INTERACTIVE flag, which tells MySQL to timeout connections based on INTERACTIVE_TIMEOUT instead of WAIT_TIMEOUT	false	3.1.0
localSocketAddress	Hostname or IP address given to explicitly configure the interface that the driver will bind the client side of the TCP/IP connection to when connecting.		5.0.5
propertiesTransform	An implementation of com.mysql.jdbc.ConnectionPropertiesTransform that the driver will use to modify URL properties passed to the driver before attempting a connection		3.1.4
useCompression	Use zlib compression when communicating with the server (true/false)? Defaults to 'false'.	false	3.0.17

Networking.

Property Name	Definition	Default Value	Since Version
tcpKeepAlive	If connecting using TCP/IP, should the driver set SO_KEEPALIVE?	true	5.0.7
tcpNoDelay	If connecting using TCP/IP, should the driver set SO_TCP_NODELAY (disabling the Nagle Algorithm)?	true	5.0.7
tcpRcvBuf	If connecting using TCP/IP, should the driver set SO_RCV_BUF to the given value? The default value of '0', means use the platform default value for this property)	0	5.0.7
tcpSndBuf	If connecting using TCP/IP, shuold the driver set SO_SND_BUF to the given value? The default value of '0', means use the platform default value for this property)	0	5.0.7
tcpTrafficClass	If connecting using TCP/IP, should the driver set traffic class or type-of-service fields ?See the documentation for java.net.Socket.setTrafficClass() for more information.	0	5.0.7

High Availability and Clustering.

Property Name	Definition	Default Value	Since Version
autoReconnect	Should the driver try to re-establish stale and/or dead connections? If enabled the driver will throw an exception for a queries issued on a stale or dead connection, which belong to the current transaction, but will attempt reconnect before the next query issued on the connection in a new transaction. The	false	1.1

	use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don't handle SQLExceptions properly, and is only designed to be used when you are unable to configure your application to handle SQLExceptions resulting from dead and stale connections properly. Alternatively, investigate setting the MySQL server variable "wait_timeout" to some high value rather than the default of 8 hours.		
autoReconnectForPools	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	false	3.1.3
failOverReadOnly	When failing over in autoReconnect mode, should the connection be set to 'read-only'?	true	3.0.12
maxReconnects	Maximum number of reconnects to attempt if autoReconnect is true, default is '3'.	3	1.1
reconnectAtTxEnd	If autoReconnect is set to true, should the driver attempt reconnections at the end of every transaction?	false	3.0.10
initialTimeout	If autoReconnect is enabled, the initial time to wait between re-connect attempts (in seconds, defaults to '2').	2	1.1
roundRobinLoadBalance	When autoReconnect is enabled, and failoverReadonly is false, should we pick hosts to connect to on a round-robin basis?	false	3.1.2
queriesBeforeRetryMaster	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.	50	3.0.2
secondsBeforeRetryMaster	How long should the driver wait, when failed over, before attempting	30	3.0.2
resourceId	A globally unique name that identifies the resource that this datasource or connection is connected to, used for XAResource.isSameRM() when the driver can't determine this value based on hostnames used in the URL		5.0.1

Security.

Property Name	Definition	Default Value	Since Version
allowMultiQueries	Allow the use of ';' to delimit multiple queries during one statement (true/false), defaults to 'false'	false	3.1.1
useSSL	Use SSL when communicating with the server (true/false), defaults to 'false'	false	3.0.2
requireSSL	Require SSL connection if useSSL=true? (defaults to 'false').	false	3.1.0

allowLoadLocalInfile	Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true').	true	3.0.3
allowUrlInLocalInfile	Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements?	false	3.1.4
clientCertificateKeyStorePassword	Password for the client certificates KeyStore		5.1.0
clientCertificateKeyStoreType	KeyStore type for client certificates (NULL or empty means use default, standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.		5.1.0
clientCertificateKeyStoreUrl	URL to the client certificate KeyStore (if not specified, use defaults)		5.1.0
trustCertificateKeyStorePassword	Password for the trusted root certificates KeyStore		5.1.0
trustCertificateKeyStoreType	KeyStore type for trusted root certificates (NULL or empty means use default, standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.		5.1.0
trustCertificateKeyStoreUrl	URL to the trusted root certificate KeyStore (if not specified, use defaults)		5.1.0
paranoid	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	false	3.0.1

Performance Extensions.

Property Name	Definition	Default Value	Since Version
callableStmtCacheSize	If 'cacheCallableStmts' is enabled, how many callable statements should be cached?	100	3.1.2
metadataCacheSize	The number of queries to cache ResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)	50	3.1.1
prepStmtCacheSize	If prepared statement caching is enabled, how many prepared statements should be cached?	25	3.0.10
prepStmtCacheSqlLimit	If prepared statement caching is enabled, what's the largest SQL the driver will cache the parsing for?	256	3.0.10
alwaysSendSetIsolation	Should the driver always communicate with the database when Connection.setTransactionIsolation() is called? If set to false, the driver will only communicate with the database when the requested transaction isolation is different than the whichever is newer, the last value that was set via Connection.setTransactionIsolation(), or the value	true	3.1.7

	that was read from the server when the connection was established.		
maintainTimeStats	Should the driver maintain various internal timers to enable idle time calculations as well as more verbose error messages when the connection to the server fails? Setting this property to false removes at least two calls to System.getCurrentTimeMillis() per query.	true	3.1.9
useCursorFetch	If connected to MySQL > 5.0.2, and setFetchSize() > 0 on a statement, should that statement use cursor-based fetching to retrieve rows?	false	5.0.0
blobSendChunkSize	Chunk to use when sending BLOB/CLOBs via ServerPreparedStatements	1048576	3.1.9
cacheCallableStmts	Should the driver cache the parsing stage of CallableStatements	false	3.1.2
cachePrepStmts	Should the driver cache the parsing stage of PreparedStatements of client-side prepared statements, the "check" for suitability of server-side prepared and server-side prepared statements themselves?	false	3.0.10
cacheResultSetMetadata	Should the driver cache ResultSetMetaData for Statements and PreparedStatements? (Req. JDK-1.4+, true/false, default 'false')	false	3.1.1
cacheServerConfiguration	Should the driver cache the results of 'SHOW VARIABLES' and 'SHOW COLLATION' on a per-URL basis?	false	3.1.5
defaultFetchSize	The driver will call setFetchSize(n) with this value on all newly-created Statements	0	3.1.9
dontTrackOpenResources	The JDBC specification requires the driver to automatically track and close resources, however if your application doesn't do a good job of explicitly calling close() on statements or result sets, this can cause memory leakage. Setting this property to true relaxes this constraint, and can be more memory efficient for some applications.	false	3.1.7
dynamicCalendars	Should the driver retrieve the default calendar when required, or cache it per connection/session?	false	3.1.5
elideSetAutoCommits	If using MySQL-4.1 or newer, should the driver only issue 'set autocommit=n' queries when the server's state doesn't match the requested state by Connection.setAutoCommit(boolean)?	false	3.1.3
enableQueryTimeouts	When enabled, query timeouts set via Statement.setQueryTimeout() use a shared java.util.Timer instance for scheduling. Even if the timeout doesn't expire before the query is processed, there will be memory used by the TimerTask for the given timeout which won't be reclaimed until the time the timeout would have expired if it hadn't been cancelled by the driver.	true	5.0.6

	High-load environments might want to consider disabling this functionality.		
holdResultsOpenOverStatementClose	Should the driver close result sets on Statement.close() as required by the JDBC specification?	false	3.1.7
largeRowSizeThreshold	What size result set row should the JDBC driver consider "large", and thus use a more memory-efficient way of representing the row internally?	2048	5.1.1
loadBalanceStrategy	If using a load-balanced connection to connect to SQL nodes in a MySQL Cluster/NDB configuration (by using the URL prefix "jdbc:mysql:loadbalance://"), which load balancing algorithm should the driver use: (1) "random" - the driver will pick a random host for each request. This tends to work better than round-robin, as the randomness will somewhat account for spreading loads where requests vary in response time, while round-robin can sometimes lead to overloaded nodes if there are variations in response times across the workload. (2) "bestResponseTime" - the driver will route the request to the host that had the best response time for the previous transaction.	random	5.0.6
locatorFetchBufferSize	If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for getBinaryInputStream?	1048576	3.2.1
rewriteBatchedStatements	Should the driver use multiqueries (irregardless of the setting of "allowMultiQueries") as well as rewriting of prepared statements for INSERT into multi-value inserts when executeBatch() is called? Notice that this has the potential for SQL injection if using plain java.sql.Statements and your code doesn't sanitize input correctly. Notice that for prepared statements, server-side prepared statements can not currently take advantage of this rewrite option, and that if you don't specify stream lengths when using PreparedStatement.set*Stream(), the driver won't be able to determine the optimum number of parameters per batch and you might receive an error from the driver that the resultant packet is too large. Statement.getGeneratedKeys() for these rewritten statements only works when the entire batch includes INSERT statements.	false	3.1.13
useDirectRowUnpack	Use newer result set row unpacking code that skips a copy from network buffers to a MySQL packet instance and instead reads directly into the result set row data buffers.	true	5.1.1
useDynamicCharsetInfo	Should the driver use a per-connection cache of character set information queried from the server when necessary, or use a built-in static mapping that is more efficient, but isn't aware of custom	true	5.0.6

	character sets or character sets implemented after the release of the JDBC driver?		
useFastDateParsing	Use internal String->Date/Time/Timestamp conversion routines to avoid excessive object creation?	true	5.0.5
useFastIntParsing	Use internal String->Integer conversion routines to avoid excessive object creation?	true	3.1.4
useJvmCharsetConverters	Always use the character encoding routines built into the JVM, rather than using lookup tables for single-byte character sets?	false	5.0.1
useLocalSessionState	Should the driver refer to the internal values of autocommit and transaction isolation that are set by Connection.setAutoCommit() and Connection.setTransactionIsolation() and transaction state as maintained by the protocol, rather than querying the database or blindly sending commands to the database for commit() or rollback() method calls?	false	3.1.7
useReadAheadInput	Use newer, optimized non-blocking, buffered input stream when reading from the server?	true	3.1.5

Debugging/Profiling.

Property Name	Definition	Default Value	Since Version
logger	The name of a class that implements "com.mysql.jdbc.log.Log" that will be used to log messages to. (default is "com.mysql.jdbc.log.StandardLogger", which logs to STDERR)	com.mysql.jdbc.log.StandardLogg	3.1.1
gatherPerfMetrics	Should the driver gather performance metrics, and report them via the configured logger every 'reportMetricsIntervalMillis' milliseconds?	false	3.1.2
profileSQL	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	false	3.1.0
profileSql	Deprecated, use 'profileSQL' instead. Trace queries and their execution/fetch times on STDERR (true/false) defaults to 'false'		2.0.14
reportMetricsIntervalMillis	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	30000	3.1.2
maxQuerySizeToLog	Controls the maximum length/size of a query that will get logged when profiling or tracing	2048	3.1.3
packetDebugBufferSize	The maximum number of packets to retain when 'enablePacketDebug' is true	20	3.1.3
slowQueryThresholdMillis	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	2000	3.1.2
slowQueryThresholdNanos	If 'useNanosForElapsedTime' is set to true, and this property is set to a non-zero value, the driver	0	5.0.7

	will use this threshold (in nanosecond units) to determine if a query was slow.		
useUsageAdvisor	Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and MySQL Connector/J to the log (true/false, defaults to 'false')?	false	3.1.1
autoGenerateTestcaseScript	Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR?	false	3.1.9
autoSlowLog	Instead of using slowQueryThreshold* to determine if a query is slow enough to be logged, maintain statistics that allow the driver to determine queries that are outside the 99th percentile?	true	5.1.4
clientInfoProvider	The name of a class that implements the com.mysql.jdbc.JDBC4ClientInfoProvider interface in order to support JDBC-4.0's Connection.get/setClientInfo() methods	com.mysql.jdbc.JDBC4Comm	5.1.0
dumpMetadataOnColumnNotFound	Should the driver dump the field-level metadata of a result set into the exception message when ResultSet.findColumn() fails?	false	3.1.13
dumpQueriesOnException	Should the driver dump the contents of the query sent to the server in the message for SQLExceptions?	false	3.1.3
enablePacketDebug	When enabled, a ring-buffer of 'packetDebugBufferSize' packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code	false	3.1.3
explainSlowQueries	If 'logSlowQueries' is enabled, should the driver automatically issue an 'EXPLAIN' on the server and send the results to the configured log at a WARN level?	false	3.1.2
includeInnodbStatusInDeadlockExceptions	Include the output of "SHOW ENGINE INNODB STATUS" in exception messages when deadlock exceptions are detected?	false	5.0.7
logSlowQueries	Should queries that take longer than 'slowQueryThresholdMillis' be logged?	false	3.1.2
logXaCommands	Should the driver log XA commands sent by MysqlXaConnection to the server, at the DEBUG level of logging?	false	5.0.5
resultSetSizeThreshold	If the usage advisor is enabled, how many rows should a result set contain before the driver warns that it is suspiciously large?	100	5.0.5
traceProtocol	Should trace-level network protocol be logged?	false	3.1.2
useNanosForElapsedTime	For profiling/debugging functionality that measures elapsed time, should the driver try to use nanoseconds resolution if available (JDK >= 1.5)?	false	5.0.7

Miscellaneous.

Property Name	Definition	Default Value	Since Version
useUnicode	Should the driver use Unicode character encodings when handling strings? Should only be used when the driver can't determine the character set mapping, or you are trying to 'force' the driver to use a character set that MySQL either doesn't natively support (such as UTF-8), true/false, defaults to 'true'	true	1.1g
characterEncoding	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')		1.1g
characterSetResults	Character set to tell the server to return results as.		3.0.13
connectionCollation	If set, tells the server to use this collation via 'set collation_connection'		3.0.13
useBlobToStoreUTF8OutsideBMP	Tells the driver to treat [MEDIUM/LONG]BLOB columns as [LONG]VARCHAR columns holding text encoded in UTF-8 that has characters outside the BMP (4-byte encodings), which MySQL server can't handle natively.	false	5.1.3
utf8OutsideBmpExcludedColumnNamePattern	When "useBlobToStoreUTF8OutsideBMP" is set to "true", column names matching the given regex will still be treated as BLOBs unless they match the regex specified for "utf8OutsideBmpIncludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
utf8OutsideBmpIncludedColumnNamePattern	Used to specify exclusion rules to "utf8OutsideBmpExcludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
sessionVariables	A comma-separated list of name/value pairs to be sent as SET SESSION ... to the server when the driver connects.		3.1.8
allowNanAndInf	Should the driver allow NaN or +/- INF values in PreparedStatement.setDouble()?	false	3.1.5
autoClosePStmtStreams	Should the driver automatically call .close() on streams/readers passed as arguments via set*() methods?	false	3.1.12
autoDeserialize	Should the driver automatically detect and de-serialize objects stored in BLOB fields?	false	3.1.5
blobsAreStrings	Should the driver always treat BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?	false	5.0.8
capitalizeTypeNames	Capitalize type names in DatabaseMetaData? (usually only useful when using WebObjects, true/false, defaults to 'false')	false	2.0.7
clobCharacterEncoding	The character encoding to use for sending and retrieving TEXT, MEDIUMTEXT and LONGTEXT		5.0.0

	values instead of the configured connection characterEncoding		
clobberStreamingResults	This will cause a 'streaming' ResultSet to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.	false	3.0.9
continueBatchOnError	Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true').	true	3.0.3
createDatabaseIfNotExist	Creates the database given in the URL if it doesn't yet exist. Assumes the configured user has permissions to create databases.	false	3.1.9
emptyStringsConvertToZero	Should the driver allow conversions from empty string fields to numeric values of '0'?	true	3.1.8
emulateLocators	Should the driver emulate java.sql.Blobs with locators? With this feature enabled, the driver will delay loading the actual Blob data until the one of the retrieval methods (getInputStream(), getBytes(), and so forth) on the blob data stream has been accessed. For this to work, you must use a column alias with the value of the column to the actual name of the Blob. The feature also has the following restrictions: The SELECT that created the result set must reference only one table, the table must have a primary key; the SELECT must alias the original blob column name, specified as a string, to an alternate name; the SELECT must cover all columns that make up the primary key.	false	3.1.0
emulateUnsupportedPstmts	Should the driver detect prepared statements that are not supported by the server, and replace them with client-side emulated versions?	true	3.1.7
functionsNeverReturnBlobs	Should the driver always treat data from functions returning BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?	false	5.0.8
generateSimpleParameterMetadata	Should the driver generate simplified parameter metadata for PreparedStatements when no metadata is available either because the server couldn't support preparing the statement, or server-side prepared statements are disabled?	false	5.0.5
ignoreNonTxTables	Ignore non-transactional table warning for rollback? (defaults to 'false').	false	3.0.9
jdbcCompliantTruncation	Should the driver throw java.sql.DataTruncation exceptions when data is truncated as is required by the JDBC specification when connected to a server that supports warnings (MySQL 4.1.0 and newer)? This property has no effect if the server sql-mode includes STRICT_TRANS_TABLES.	true	3.1.2

maxRows	The maximum number of rows to return (0, the default means return all rows).	-1	all versions
netTimeoutForStreamingResults	What value should the driver automatically set the server setting 'net_write_timeout' to when the streaming result sets feature is in use? (value has unit of seconds, the value '0' means the driver will not try and adjust this value)	600	5.1.0
noAccessToProcedureBodies	When determining procedure parameter types for CallableStatements, and the connected user can't access procedure bodies through "SHOW CREATE PROCEDURE" or select on mysql.proc should the driver instead create basic metadata (all parameters reported as IN VARCHARs, but allowing registerOutParameter() to be called on them anyway) instead of throwing an exception?	false	5.0.3
noDatetimeStringSync	Don't ensure that ResultSet.getDatetimeType().toString().equals(ResultSet.getString())	false	3.1.7
noTimezoneConversionForTimeType	Don't convert TIME values using the server timezone if 'useTimezone'='true'	false	5.0.0
nullCatalogMeansCurrent	When DatabaseMetadataMethods ask for a 'catalog' parameter, does the value null mean use the current catalog? (this is not JDBC-compliant, but follows legacy behavior from earlier versions of the driver)	true	3.1.8
nullNamePatternMatchesAll	Should DatabaseMetaData methods that accept *pattern parameters treat null the same as '%' (this is not JDBC-compliant, however older versions of the driver accepted this departure from the specification)	true	3.1.8
overrideSupportsIntegrityEnhancementFacility	Should the driver return "true" for DatabaseMetaData.supportsIntegrityEnhancementFacility() even if the database doesn't support it to workaround applications that require this method to return "true" to signal support of foreign keys, even though the SQL specification states that this facility contains much more than just foreign key support (one such application being OpenOffice)?	false	3.1.12
padCharsWithSpace	If a result set column has the CHAR type and the value does not fill the amount of characters specified in the DDL for the column, should the driver pad the remaining characters with space (for ANSI compliance)?	false	5.0.6
pedantic	Follow the JDBC spec to the letter.	false	3.0.0
pinGlobalTxToPhysicalConnection	When using XAConnections, should the driver ensure that operations on a given XID are always routed to the same physical connection? This allows the XAConnection to support "XA START ... JOIN" after "XA END" has been called	false	5.0.1
populateInsertRowWithDefaultValues	When using ResultSets that are CONCUR_UPDATABLE, should the driver pre-	false	5.0.5

	populate the "insert" row with default values from the DDL for the table used in the query so those values are immediately available for ResultSet accessors? This functionality requires a call to the database for metadata each time a result set of this type is created. If disabled (the default), the default values will be populated by the an internal call to refreshRow() which pulls back default values and/or values changed by triggers.		
processEscapeCodesForPrepStmts	Should the driver process escape codes in queries that are prepared?	true	3.1.12
relaxAutoCommit	If the version of MySQL the driver connects to does not support transactions, still allow calls to commit(), rollback() and setAutoCommit() (true/false, defaults to 'false')?	false	2.0.13
retainStatementAfterResultSetClose	Should the driver retain the Statement reference in a ResultSet after ResultSet.close() has been called. This is not JDBC-compliant after JDBC-4.0.	false	3.1.11
rollbackOnPooledClose	Should the driver issue a rollback() when the logical connection in a pool is closed?	true	3.0.15
runningCTS13	Enables workarounds for bugs in Sun's JDBC compliance testsuite version 1.3	false	3.1.7
serverTimezone	Override detection/mapping of timezone. Used when timezone from server doesn't map to Java timezone		3.0.2
statementInterceptors	A comma-delimited list of classes that implement "com.mysql.jdbc.StatementInterceptor" that should be placed "in between" query execution to influence the results. StatementInterceptors are "chainable", the results returned by the "current" interceptor will be passed on to the next in in the chain, from left-to-right order, as specified in this property.		5.1.1
strictFloatingPoint	Used only in older versions of compliance test	false	3.0.0
strictUpdates	Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')?	true	3.0.4
tinyInt1isBit	Should the driver treat the datatype TINYINT(1) as the BIT type (because the server silently converts BIT -> TINYINT(1) when creating tables)?	true	3.0.16
transformedBitIsBoolean	If the driver converts TINYINT(1) to a different type, should it use BOOLEAN instead of BIT for future compatibility with MySQL-5.0, as MySQL-5.0 has a BIT type?	false	3.1.9
treatUtilDateAsTimestamp	Should the driver treat java.util.Date as a TIMESTAMP for the purposes of PreparedStatement.setObject()?	true	5.0.5
ultraDevHack	Create PreparedStatements for prepareCall() when required, because UltraDev is broken and issues	false	2.0.3

	a prepareCall() for _all_ statements? (true/false, defaults to 'false')		
useGmtMillisForDatetimes	Convert between session timezone and GMT before creating Date and Timestamp instances (value of "false" is legacy behavior, "true" leads to more JDBC-compliant behavior.	false	3.1.12
useHostsInPrivileges	Add '@hostname' to users in DatabaseMetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.	true	3.0.2
useInformationSchema	When connected to MySQL-5.0.7 or newer, should the driver use the INFORMATION_SCHEMA to derive information used by DatabaseMetaData?	false	5.0.0
useJDBCCompliantTimezoneShift	Should the driver use JDBC-compliant rules when converting TIME/TIMESTAMP/DATETIME values' timezone information for those JDBC arguments which take a java.util.Calendar argument? (Notice that this option is exclusive of the "useTimezone=true" configuration option.)	false	5.0.0
useOldAliasMetadataBehavior	Should the driver use the legacy behavior for "AS" clauses on columns and tables, and only return aliases (if any) for ResultSetMetaData.getColumnName() or ResultSetMetaData.getTableName() rather than the original column/table name?	false	5.0.4
useOldUTF8Behavior	Use the UTF-8 behavior the driver did when communicating with 4.0 and older servers	false	3.1.6
useOnlyServerErrorMessages	Don't prepend 'standard' SQLState error messages to error messages returned by the server.	true	3.0.15
useSSPSCompatibleTimezoneShift	If migrating from an environment that was using server-side prepared statements, and the configuration property "useJDBCCompliantTimeZoneShift" set to "true", use compatible behavior when not using server-side prepared statements when sending TIMESTAMP values to the MySQL server.	false	5.0.5
useServerPrepStmts	Use server-side prepared statements if the server supports them?	false	3.1.0
useSqlStateCodes	Use SQL Standard state codes instead of 'legacy' X/Open/SQL state codes (true/false), default is 'true'	true	3.1.3
useStreamLengthsInPrepStmts	Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')?	true	3.0.2
useTimezone	Convert time/date types between client and server timezones (true/false, defaults to 'false')?	false	3.0.2
useUnbufferedInput	Don't use BufferedInputStream for reading data from the server	true	3.0.11
yearIsDateType	Should the JDBC driver treat the MySQL type "YEAR" as a java.sql.Date, or as a SHORT?	true	3.1.9

zeroDateTimeBehavior	What should happen when the driver encounters DATETIME values that are composed entirely of zeroes (used by MySQL to represent invalid dates)? Valid values are "exception", "round" and "convertToNull".	exception	3.1.4

Connector/J also supports access to MySQL via named pipes on Windows NT/2000/XP using the NamedPipeSocketFactory as a plugin-socket factory via the socketFactory property. If you don't use a namedPipePath property, the default of '\\.\pipe\MySQL' will be used. If you use the `NamedPipeSocketFactory`, the hostname and port number values in the JDBC url will be ignored. You can enable this feature using:

```
socketFactory=com.mysql.jdbc.NamedPipeSocketFactory
```

Named pipes only work when connecting to a MySQL server on the same physical machine as the one the JDBC driver is being used on. In simple performance tests, it appears that named pipe access is between 30%-50% faster than the standard TCP/IP access.

You can create your own socket factories by following the example code in `com.mysql.jdbc.NamedPipeSocketFactory`, or `com.mysql.jdbc.StandardSocketFactory`.

25.4.4.2. JDBC API Implementation Notes

MySQL Connector/J passes all of the tests in the publicly-available version of Sun's JDBC compliance test suite. However, in many places the JDBC specification is vague about how certain functionality should be implemented, or the specification allows leeway in implementation.

This section gives details on a interface-by-interface level about how certain implementation decisions may affect how you use MySQL Connector/J.

- **Blob**

  Starting with Connector/J version 3.1.0, you can emulate Blobs with locators by adding the property 'emulateLocators=true' to your JDBC URL. Using this method, the driver will delay loading the actual Blob data until you retrieve the other data and then use retrieval methods (`getInputStream()`, `getBytes()`, and so forth) on the blob data stream.

  For this to work, you must use a column alias with the value of the column to the actual name of the Blob, for example:

  ```
  SELECT id, data as 'data' from blobtable
  ```

  For this to work, you must also follow follow these rules:

  - The `SELECT` must also reference only one table, the table must have a primary key.
  - The `SELECT` must cover all columns that make up the primary key.

  The Blob implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the corresponding `PreparedStatement.setBlob()` or `ResultSet.updateBlob()` (in the case of updatable result sets) methods to save changes back to the database.

- **CallableStatement**

Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer via the `CallableStatement` interface. Currently, the `getParameterMetaData()` method of `CallableStatement` is not supported.

- **Clob**

  The Clob implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.

- **Connection**

  Unlike older versions of MM.MySQL the `isClosed()` method does not ping the server to determine if it is alive. In accordance with the JDBC specification, it only returns true if `closed()` has been called on the connection. If you need to determine if the connection is still valid, you should issue a simple query, such as `SELECT 1`. The driver will throw an exception if the connection is no longer valid.

- **DatabaseMetaData**

  Foreign Key information (`getImportedKeys()`/`getExportedKeys()` and `getCrossReference()`) is only available from InnoDB tables. However, the driver uses `SHOW CREATE TABLE` to retrieve this information, so when other storage engines support foreign keys, the driver will transparently support them as well.

- **PreparedStatement**

  PreparedStatements are implemented by the driver, as MySQL does not have a prepared statement feature. Because of this, the driver does not implement `getParameterMetaData()` or `getMetaData()` as it would require the driver to have a complete SQL parser in the client.

  Starting with version 3.1.0 MySQL Connector/J, server-side prepared statements and binary-encoded result sets are used when the server supports them.

  Take care when using a server-side prepared statement with **large** parameters that are set via `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()`, or `setClob()`. If you want to re-execute the statement with any large parameter changed to a non-large parameter, it is necessary to call `clearParameters()` and set all parameters again. The reason for this is as follows:

  - During both server-side prepared statements and client-side emulation, large data is exchanged only when `PreparedStatement.execute()` is called.
  - Once that has been done, the stream used to read the data on the client side is closed (as per the JDBC spec), and can't be read from again.
  - If a parameter changes from large to non-large, the driver must reset the server-side state of the prepared statement to allow the parameter that is being changed to take the place of the prior large value. This removes all of the large data that has already been sent to the server, thus requiring the data to be re-sent, via the `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()` or `setClob()` methods.

  Consequently, if you want to change the type of a parameter to a non-large one, you must call `clearParameters()` and set all parameters of the prepared statement again before it can be re-executed.

- **ResultSet**

By default, ResultSets are completely retrieved and stored in memory. In most cases this is the most efficient way to operate, and due to the design of the MySQL network protocol is easier to implement. If you are working with ResultSets that have a large number of rows or large values, and can not allocate heap space in your JVM for the memory required, you can tell the driver to stream the results back one row at a time.

To enable this functionality, you need to create a Statement instance in the following manner:

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
             java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
```

The combination of a forward-only, read-only result set, with a fetch size of `Integer.MIN_VALUE` serves as a signal to the driver to stream result sets row-by-row. After this any result sets created with the statement will be retrieved row-by-row.

There are some caveats with this approach. You will have to read all of the rows in the result set (or close it) before you can issue any other queries on the connection, or an exception will be thrown.

The earliest the locks these statements hold can be released (whether they be `MyISAM` table-level locks or row-level locks in some other storage engine such as `InnoDB`) is when the statement completes.

If the statement is within scope of a transaction, then locks are released when the transaction completes (which implies that the statement needs to complete first). As with most other databases, statements are not complete until all the results pending on the statement are read or the active result set for the statement is closed.

Therefore, if using streaming results, you should process them as quickly as possible if you want to maintain concurrent access to the tables referenced by the statement producing the result set.

- **ResultSetMetaData**

  The `isAutoIncrement()` method only works when using MySQL servers 4.0 and newer.

- **Statement**

  When using versions of the JDBC driver earlier than 3.2.1, and connected to server versions earlier than 5.0.3, the `setFetchSize()` method has no effect, other than to toggle result set streaming as described above.

  Connector/J 5.0.0 and later include support for both `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL 5.0.0 or newer server, and require a separate connection to issue the `KILL QUERY` statement. In the case of `setQueryTimeout()`, the implementation creates an additional thread to handle the timeout functionality.

  > **Nota**
  >
  > Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeException` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

  MySQL does not support SQL cursors, and the JDBC driver doesn't emulate them, so "setCursorName()" has no effect.

25.4.4.3. Java, JDBC and MySQL Types

MySQL Connector/J is flexible in the way it handles conversions between MySQL data types and Java data types.

In general, any MySQL data type can be converted to a java.lang.String, and any numerical type can be converted to any of the Java numerical types, although round-off, overflow, or loss of precision may occur.

Starting with Connector/J 3.1.0, the JDBC driver will issue warnings or throw DataTruncation exceptions as is required by the JDBC specification unless the connection was configured not to do so by using the property jdbcCompliantTruncation and setting it to `false`.

The conversions that are always guaranteed to work are listed in the following table:

Connection Properties - Miscellaneous.

These MySQL Data Types	Can always be converted to these Java types
`CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET`	`java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob`
`FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT`	`java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal`
`DATE, TIME, DATETIME, TIMESTAMP`	`java.lang.String, java.sql.Date, java.sql.Timestamp`

Note: round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.

The `ResultSet.getObject()` method uses the type conversions between MySQL and Java types, following the JDBC specification where appropriate. The value returned by `ResultSetMetaData.GetColumnClassName()` is also shown below. For more information on the `java.sql.Types` classes see Java 2 Platform Types.

MySQL Types to Java Types for ResultSet.getObject().

MySQL Type Name	Return value of `GetColumnClassName`	Returned as Java Class
BIT(1) (new in MySQL-5.0)	BIT	`java.lang.Boolean`
BIT(> 1) (new in MySQL-5.0)	BIT	`byte[]`
TINYINT	TINYINT	`java.lang.Boolean` if the configuration property `tinyInt1isBit` is set to `true` (the default) and the storage size is 1, or `java.lang.Integer` if not.
BOOL, BOOLEAN	TINYINT	See TINYINT, above as these are aliases for TINYINT(1), currently.
SMALLINT[(M)] [UNSIGNED]	SMALLINT [UNSIGNED]	`java.lang.Integer` (regardless if UNSIGNED or not)
MEDIUMINT[(M)] [UNSIGNED]	MEDIUMINT [UNSIGNED]	`java.lang.Integer,` if UNSIGNED `java.lang.Long`

INT,INTEGER[(M)] [UNSIGNED]	INTEGER [UNSIGNED]	`java.lang.Integer`, if UNSIGNED `java.lang.Long`
BIGINT[(M)] [UNSIGNED]	BIGINT [UNSIGNED]	`java.lang.Long`, if UNSIGNED `java.math.BigInteger`
FLOAT[(M,D)]	FLOAT	`java.lang.Float`
DOUBLE[(M,B)]	DOUBLE	`java.lang.Double`
DECIMAL[(M[,D])]	DECIMAL	`java.math.BigDecimal`
DATE	DATE	`java.sql.Date`
DATETIME	DATETIME	`java.sql.Timestamp`
TIMESTAMP[(M)]	TIMESTAMP	`java.sql.Timestamp`
TIME	TIME	`java.sql.Time`
YEAR[(2|4)]	YEAR	If `yearIsDateType` configuration property is set to false, then the returned object type is `java.sql.Short`. If set to true (the default) then an object of type `java.sql.Date` (with the date set to January 1st, at midnight).
CHAR(M)	CHAR	`java.lang.String` (unless the character set for the column is BINARY, then `byte[]` is returned.
VARCHAR(M) [BINARY]	VARCHAR	`java.lang.String` (unless the character set for the column is BINARY, then `byte[]` is returned.
BINARY(M)	BINARY	`byte[]`
VARBINARY(M)	VARBINARY	`byte[]`
TINYBLOB	TINYBLOB	`byte[]`
TINYTEXT	VARCHAR	`java.lang.String`
BLOB	BLOB	`byte[]`
TEXT	VARCHAR	`java.lang.String`
MEDIUMBLOB	MEDIUMBLOB	`byte[]`
MEDIUMTEXT	VARCHAR	`java.lang.String`
LONGBLOB	LONGBLOB	`byte[]`
LONGTEXT	VARCHAR	`java.lang.String`
ENUM('value1','value2',...)	CHAR	`java.lang.String`
SET('value1','value2',...)	CHAR	`java.lang.String`

25.4.4.4. Using Character Sets and Unicode

All strings sent from the JDBC driver to the server are converted automatically from native Java Unicode form to the client character encoding, including all queries sent via `Statement.execute()`, `Statement.executeUpdate()`, `Statement.executeQuery()` as well as all `PreparedStatement` and `CallableStatement` parameters with the exclusion of parameters set using `setBytes()`, `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()` and `setBlob()` .

Prior to MySQL Server 4.1, Connector/J supported a single character encoding per connection, which could either be automatically detected from the server configuration, or could be configured by the user through the *`useUnicode`* and *`characterEncoding`* properties.

Starting with MySQL Server 4.1, Connector/J supports a single character encoding between client and server, and any number of character encodings for data returned by the server to the client in `ResultSets`.

The character encoding between client and server is automatically detected upon connection. The encoding used by the driver is specified on the server via the `character_set` system variable for server versions older than 4.1.0 and `character_set_server` for server versions 4.1.0 and newer. For more information, see Sección 10.3.1, “Conjunto de caracteres y colación del servidor”.

To override the automatically-detected encoding on the client side, use the *`characterEncoding`* property in the URL used to connect to the server.

When specifying character encodings on the client side, Java-style names should be used. The following table lists Java-style names for MySQL character sets:

MySQL to Java Encoding Name Translations.

MySQL Character Set Name	Java-Style Character Encoding Name
ascii	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
cp932	Cp932 or MS932 (MySQL Server > 4.1.11)
gb2312	EUC_CN
ujis	EUC_JP
euckr	EUC_KR
latin1	ISO8859_1
latin2	ISO8859_2
greek	ISO8859_7
hebrew	ISO8859_8
cp866	Cp866
tis620	TIS620
cp1250	Cp1250
cp1251	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

Warning. Do not issue the query 'set names' with Connector/J, as the driver will not detect that the character set has changed, and will continue to use the character set detected during the initial connection setup.

To allow multiple character sets to be sent from the client, the UTF-8 encoding should be used, either by configuring `utf8` as the default server character set, or by configuring the JDBC driver to use UTF-8 through the *`characterEncoding`* property.

25.4.4.5. Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. The performance penalty for enabling SSL is an increase in query processing time between 35% and 50%, depending on the size of the query, and the amount of data it returns.

For SSL Support to work, you must have the following:

- A JDK that includes JSSE (Java Secure Sockets Extension), like JDK-1.4.1 or newer. SSL does not currently work with a JDK that you can add JSSE to, like JDK-1.2.x or JDK-1.3.x due to the following JSSE bug: http://developer.java.sun.com/developer/bugParade/bugs/4273544.html
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL-4.0.4 or later, see Sección 5.7.7, “Usar conexiones seguras”, for more information.
- A client certificate (covered later in this section)

You will first need to import the MySQL server CA Certificate into a Java truststore. A sample MySQL server CA Certificate is located in the `SSL` subdirectory of the MySQL source distribution. This is what SSL will use to determine if you are communicating with a secure MySQL server.

To use Java's `keytool` to create a truststore in the current directory , and import the server's CA certificate (`cacert.pem`), you can do the following (assuming that `keytool` is in your path. The `keytool` should be located in the `bin` subdirectory of your JDK or JRE):

```
shell> keytool -import -alias mysqlServerCACert \
                                  -file cacert.pem -keystore truststore
```

Keytool will respond with the following information:

```
Enter keystore password:  *********
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus,
       O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus,
       O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Serial number: 0
Valid from:
   Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
    MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
    SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

You will then need to generate a client certificate, so that the MySQL server knows that it is talking to a secure client:

```
 shell> keytool -genkey -keyalg rsa \
     -alias mysqlClientCertificate -keystore keystore
```

Keytool will prompt you for the following information, and create a keystore named `keystore` in the current directory.

You should respond with information that is appropriate for your situation:

```
Enter keystore password:  *********
What is your first and last name?
  [Unknown]:  Matthews
What is the name of your organizational unit?
  [Unknown]:  Software Development
What is the name of your organization?
  [Unknown]:  MySQL AB
What is the name of your City or Locality?
```

```
  [Unknown]:  Flossmoor
What is the name of your State or Province?
  [Unknown]:  IL
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
 L=Flossmoor, ST=IL, C=US> correct?
  [no]:  y

Enter key password for <mysqlClientCertificate>
        (RETURN if same as keystore password):
```

Finally, to get JSSE to use the keystore and truststore that you have generated, you need to set the following system properties when you start your JVM, replacing path_to_keystore_file with the full path to the keystore file you created, path_to_truststore_file with the path to the truststore file you created, and using the appropriate password values for each property.

```
-Djavax.net.ssl.keyStore=path_to_keystore_file
-Djavax.net.ssl.keyStorePassword=*********
-Djavax.net.ssl.trustStore=path_to_truststore_file
-Djavax.net.ssl.trustStorePassword=*********
```

You will also need to set useSSL to `true` in your connection parameters for MySQL Connector/J, either by adding `useSSL=true` to your URL, or by setting the property useSSL to `true` in the `java.util.Properties` instance you pass to `DriverManager.getConnection()`.

You can test that SSL is working by turning on JSSE debugging (as detailed below), and look for the following key events:

```
...
*** ClientHello, v3.1
RandomCookie:  GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, »
  54, 244, 0, 168, 55, 103, 215, 64, 16, 138, 225, 190, 132, 153, 2, »
  217, 219, 239, 202, 19, 121, 78 }
Session ID:  {}
Cipher Suites:  { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
Compression Methods:  { 0 }
***
[write] MD5 and SHA1 hashes:  len = 59
0000: 01 00 00 37 03 01 3D B6 90 FA C7 94 B4 D7 4A 0C  ...7..=.......J.
0010: 36 F4 00 A8 37 67 D7 40 10 8A E1 BE 84 99 02 D9  6...7g.@........
0020: DB EF CA 13 79 4E 00 00 10 00 05 00 04 00 09 00  ....yN..........
0030: 0A 00 12 00 13 00 03 00 11 01 00                 ...........
main, WRITE:  SSL v3.1 Handshake, length = 59
main, READ:  SSL v3.1 Handshake, length = 74
*** ServerHello, v3.1
RandomCookie:  GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, »
   202, 79, 185, 178, 100, 215, 66, 254, 21, 83, 187, 190, 42, 170, 3, »
   132, 110, 82, 148, 160, 92 }
Session ID:  {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, »
   182, 158, 30, 11, 150, 79, 170, 76, 255, 92, 15, 226, 24, 17, 177, »
   219, 158, 177, 187, 143}
Cipher Suite:  { 0, 5 }
Compression Method: 0
***
%% Created:  [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes:  len = 74
0000: 02 00 00 46 03 01 3D B6 43 98 74 32 04 67 19 64  ...F..=.C.t2.g.d
0010: 3A CA 4F B9 B2 64 D7 42 FE 15 53 BB BE 2A AA 03  :.O..d.B..S..*..
0020: 84 6E 52 94 A0 5C 20 A3 E3 54 35 51 7F FC FE B2  .nR..\ ..T5Q....
0030: B3 44 3F B6 9E 1E 0B 96 4F AA 4C FF 5C 0F E2 18  .D?.....O.L.\...
0040: 11 B1 DB 9E B1 BB 8F 00 05 00                    ..........
main, READ:  SSL v3.1 Handshake, length = 1712
...
```

JSSE provides debugging (to STDOUT) when you set the following system property: `-Djavax.net.debug=all` This will tell you what keystores and truststores are being used, as well as what is going on during the SSL handshake and certificate exchange. It will be helpful when trying to determine what is not working when trying to get an SSL connection to happen.

25.4.4.6. Using Master/Slave Replication with ReplicationConnection

Starting with Connector/J 3.1.7, we've made available a variant of the driver that will automatically send queries to a read/write master, or a failover or round-robin loadbalanced set of slaves based on the state of `Connection.getReadOnly()` .

An application signals that it wants a transaction to be read-only by calling `Connection.setReadOnly(true)`, this replication-aware connection will use one of the slave connections, which are load-balanced per-vm using a round-robin scheme (a given connection is sticky to a slave unless that slave is removed from service). If you have a write transaction, or if you have a read that is time-sensitive (remember, replication in MySQL is asynchronous), set the connection to be not read-only, by calling `Connection.setReadOnly(false)` and the driver will ensure that further calls are sent to the master MySQL server. The driver takes care of propagating the current state of autocommit, isolation level, and catalog between all of the connections that it uses to accomplish this load balancing functionality.

To enable this functionality, use the " `com.mysql.jdbc.ReplicationDriver` " class when configuring your application server's connection pool or when creating an instance of a JDBC driver for your standalone application. Because it accepts the same URL format as the standard MySQL JDBC driver, `ReplicationDriver` does not currently work with `java.sql.DriverManager` -based connection creation unless it is the only MySQL JDBC driver registered with the `DriverManager` .

Here is a short, simple example of how ReplicationDriver might be used in a standalone application.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.util.Properties;

import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {

  public static void main(String[] args) throws Exception {
    ReplicationDriver driver = new ReplicationDriver();

    Properties props = new Properties();

    // We want this for failover on the slaves
    props.put("autoReconnect", "true");

    // We want to load balance between the slaves
    props.put("roundRobinLoadBalance", "true");

    props.put("user", "foo");
    props.put("password", "bar");

    //
    // Looks like a normal MySQL JDBC url, with a
    // comma-separated list of hosts, the first
    // being the 'master', the rest being any number
    // of slaves that the driver will load balance against
    //

    Connection conn =
        driver.connect("jdbc:mysql://master,slave1,slave2,slave3/test",
            props);
```

```
    //
    // Perform read/write work on the master
    // by setting the read-only flag to "false"
    //

    conn.setReadOnly(false);
    conn.setAutoCommit(false);
    conn.createStatement().executeUpdate("UPDATE some_table ....");
    conn.commit();

    //
    // Now, do a query from a slave, the driver automatically picks one
    // from the list
    //

    conn.setReadOnly(true);

    ResultSet rs =
      conn.createStatement().executeQuery("SELECT a,b FROM alt_table");

     .......
  }
}
```

25.4.5. Connector/J Notes and Tips

25.4.5.1. Basic JDBC Concepts

This section provides some general JDBC background.

Connecting to MySQL Using the `DriverManager` Interface

When you are using JDBC outside of an application server, the `DriverManager` class manages the establishment of Connections.

The `DriverManager` needs to be told which JDBC drivers it should try to make Connections with. The easiest way to do this is to use `Class.forName()` on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is `com.mysql.jdbc.Driver`. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

The following section of Java code shows how you might register MySQL Connector/J from the `main()` method of your application:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
}
```

After the driver has been registered with the `DriverManager`, you can obtain a `Connection` instance that is connected to a particular database by calling `DriverManager.getConnection()`:

Ejemplo 25.1. Obtaining a connection from the `DriverManager`

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. You should see the API documentation that comes with your JDK for more specific information on how to use them.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

...
try {
    Connection conn =
       DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                                   "user=monty&password=greatsqldb");

    // Do something with the Connection

   ...
} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

Once a `Connection` is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database. This is explained in the following sections.

Using Statements to Execute SQL

`Statement` objects allow you to execute basic SQL queries and retrieve the results through the `ResultSet` class which is described later.

To create a `Statement` instance, you call the `createStatement()` method on the `Connection` object you have retrieved via one of the `DriverManager.getConnection()` or `DataSource.getConnection()` methods described earlier.

Once you have a `Statement` instance, you can execute a `SELECT` query by calling the `executeQuery(String)` method with the SQL you want to use.

To update data in the database, use the `executeUpdate(String SQL)` method. This method returns the number of rows affected by the update statement.

If you don't know ahead of time whether the SQL statement will be a `SELECT` or an `UPDATE`/`INSERT`, then you can use the `execute(String SQL)` method. This method will return true if the SQL query was a `SELECT`, or false if it was an `UPDATE`, `INSERT`, or `DELETE` statement. If the statement was a `SELECT` query, you can retrieve the results by calling the `getResultSet()` method. If the statement was an `UPDATE`, `INSERT`, or `DELETE` statement, you can retrieve the affected rows count by calling `getUpdateCount()` on the `Statement` instance.

Ejemplo 25.2. Using java.sql.Statement to execute a `SELECT` query

```
// assume that conn is an already created JDBC connection
Statement stmt = null;
ResultSet rs = null;

try {
```

```
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...

    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }

    // Now do something with the ResultSet ....
} finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { // ignore }

        rs = null;
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { // ignore }

        stmt = null;
    }
}
```

Using `CallableStatements` to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

See Capítulo 19, *Procedimientos almacenados y funciones*, for more information on MySQL stored procedures.

Connector/J exposes stored procedure functionality through JDBC's `CallableStatement` interface.

Note. Current versions of MySQL server do not return enough information for the JDBC driver to provide result set metadata for callable statements. This means that when using `CallableStatement`, `ResultSetMetaData` may return `NULL`.

The following example shows a stored procedure that returns the value of `inOutParam` incremented by 1, and the string passed in via `inputParam` as a `ResultSet`:

Ejemplo 25.3. Stored Procedures

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                                      INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using `Connection.prepareCall()` .

   Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

   Ejemplo 25.4. Using `Connection.prepareCall()`

   ```
   import java.sql.CallableStatement;

   ...

       //
       // Prepare a call to the stored procedure 'demoSp'
       // with two parameters
       //
       // Notice the use of JDBC-escape syntax ({call ...})
       //

       CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");



       cStmt.setString(1, "abcdefg");
   ```

   Note. `Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, you should try to minimize unnecessary calls to `Connection.prepareCall()` by reusing `CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

   To retrieve the values of output parameters (parameters specified as `OUT` or `INOUT` when you created the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

   Ejemplo 25.5. Registering output parameters

   ```
   import java.sql.Types;
   ...
   //
   // Connector/J supports both named and indexed
   // output parameters. You can register output
   // parameters using either method, as well
   // as retrieve output parameters using either
   // method, regardless of what method was
   // used to register them.
   //
   // The following examples show how to use
   // the various methods of registering
   // output parameters (you should of course
   // use only one registration per parameter).
   //

   //
   // Registers the second parameter as output, and
   // uses the type 'INTEGER' for values returned from
   // getObject()
   //

   cStmt.registerOutParameter(2, Types.INTEGER);
   ```

```
//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...
```

3. Set the input parameters (if any exist)

   Input and in/out parameters are set as for `PreparedStatement` objects. However, `CallableStatement` also supports setting parameters by name:

   Ejemplo 25.6. Setting `CallableStatement` input parameters

```
...

    //
    // Set a parameter by index
    //

    cStmt.setString(1, "abcdefg");

    //
    // Alternatively, set a parameter using
    // the parameter name
    //

    cStmt.setString("inputParameter", "abcdefg");

    //
    // Set the 'in/out' parameter using an index
    //

    cStmt.setInt(2, 1);

    //
    // Alternatively, set the 'in/out' parameter
    // by name
    //

    cStmt.setInt("inOutParam", 1);

...
```

4. Execute the `CallableStatement`, and retrieve any result sets or output parameters.

   Although `CallableStatement` supports calling any of the `Statement` execute methods (`executeUpdate()`, `executeQuery()` or `execute()`), the most flexible method to call is `execute()`, as you do not need to know ahead of time if the stored procedure returns result sets:

   Ejemplo 25.7. Retrieving results and output parameter values

```
...

    boolean hadResults = cStmt.execute();

    //
    // Process all returned result sets
    //

    while (hadResults) {
        ResultSet rs = cStmt.getResultSet();
```

```
        // process result set
        ...

        hadResults = rs.getMoreResults();
    }

    //
    // Retrieve output parameters
    //
    // Connector/J supports both index-based and
    // name-based retrieval
    //

    int outputValue = cStmt.getInt(2); // index-based

    outputValue = cStmt.getInt("inOutParam"); // name-based

...
```

Retrieving `AUTO_INCREMENT` Column Values

Before version 3.0 of the JDBC API, there was no standard way of retrieving key values from databases that supported auto increment or identity columns. With older JDBC drivers for MySQL, you could always use a MySQL-specific method on the `Statement` interface, or issue the query `SELECT LAST_INSERT_ID()` after issuing an `INSERT` to a table that had an `AUTO_INCREMENT` key. Using the MySQL-specific method call isn't portable, and issuing a `SELECT` to get the `AUTO_INCREMENT` key's value requires another round-trip to the database, which isn't as efficient as possible. The following code snippets demonstrate the three different ways to retrieve `AUTO_INCREMENT` values. First, we demonstrate the use of the new JDBC-3.0 method `getGeneratedKeys()` which is now the preferred method to use if you need to retrieve `AUTO_INCREMENT` keys and have access to JDBC-3.0. The second example shows how you can retrieve the same value using a standard `SELECT LAST_INSERT_ID()` query. The final example shows how updatable result sets can retrieve the `AUTO_INCREMENT` value when using the `insertRow()` method.

Ejemplo 25.8. Retrieving `AUTO_INCREMENT` column values using `Statement.getGeneratedKeys()`

```
Statement stmt = null;
ResultSet rs = null;

try {

 //
 // Create a Statement instance that we can use for
 // 'normal' result sets assuming you have a
 // Connection 'conn' to a MySQL database already
 // available

 stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                             java.sql.ResultSet.CONCUR_UPDATABLE);

 //
 // Issue the DDL queries for the table for this example
 //

 stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
 stmt.executeUpdate(
         "CREATE TABLE autoIncTutorial ("
         + "priKey INT NOT NULL AUTO_INCREMENT, "
         + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

 //
 // Insert one row that will generate an AUTO INCREMENT
 // key in the 'priKey' field
```

```
    //

    stmt.executeUpdate(
            "INSERT INTO autoIncTutorial (dataField) "
            + "values ('Can I Get the Auto Increment Field?')",
            Statement.RETURN_GENERATED_KEYS);

    //
    // Example of using Statement.getGeneratedKeys()
    // to retrieve the value of an auto-increment
    // value
    //

    int autoIncKeyFromApi = -1;

    rs = stmt.getGeneratedKeys();

    if (rs.next()) {
        autoIncKeyFromApi = rs.getInt(1);
    } else {

        // throw an exception from here
    }

    rs.close();

    rs = null;

    System.out.println("Key returned from getGeneratedKeys():"
        + autoIncKeyFromApi);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

Ejemplo 25.9. Retrieving `AUTO_INCREMENT` column values using `SELECT LAST_INSERT_ID()`

```
   Statement stmt = null;
   ResultSet rs = null;

   try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.

    stmt = conn.createStatement();

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
```

```
    stmt.executeUpdate(
            "CREATE TABLE autoIncTutorial ("
            + "priKey INT NOT NULL AUTO_INCREMENT, "
            + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
            "INSERT INTO autoIncTutorial (dataField) "
            + "values ('Can I Get the Auto Increment Field?')");

    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //

    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();

    System.out.println("Key returned from " +
                       "'SELECT LAST_INSERT_ID()': " +
                       autoIncKeyFromFunc);

} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

Ejemplo 25.10. Retrieving `AUTO_INCREMENT` column values in `Updatable ResultSets`

```
   Statement stmt = null;
   ResultSet rs = null;

   try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //
```

```
    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
            "CREATE TABLE autoIncTutorial ("
            + "priKey INT NOT NULL AUTO_INCREMENT, "
            + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //

    rs = stmt.executeQuery("SELECT priKey, dataField "
       + "FROM autoIncTutorial");

    rs.moveToInsertRow();

    rs.updateString("dataField", "AUTO INCREMENT here?");
    rs.insertRow();

    //
    // the driver adds rows at the end
    //

    rs.last();

    //
    // We should now be on the row we just inserted
    //

    int autoIncKeyFromRS = rs.getInt("priKey");

    rs.close();

    rs = null;

    System.out.println("Key returned for inserted row: "
        + autoIncKeyFromRS);

} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

When you run the preceding example code, you should get the following output: Key returned from `getGeneratedKeys()`: 1 Key returned from `SELECT LAST_INSERT_ID()`: 1 Key returned for inserted row: 2 You should be aware, that at times, it can be tricky to use the `SELECT LAST_INSERT_ID()` query, as that function's value is scoped to a connection. So, if some other query happens on the same connection, the value will be overwritten. On the other hand, the `getGeneratedKeys()` method is scoped by the `Statement` instance, so it can be used even if other queries happen on the same connection, but not on the same `Statement` instance.

25.4.5.2. Using Connector/J with J2EE and Other Java Frameworks

This section describes how to use Connector/J in several contexts.

General J2EE Concepts

This section provides general background on J2EE concepts that pertain to use of Connector/J.

Understanding Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them.

This technique of pooling connections is based on the fact that most applications only need a thread to have access to a JDBC connection when they are actively processing a transaction, which usually take only milliseconds to complete. When not processing a transaction, the connection would otherwise sit idle. Instead, connection pooling allows the idle connection to be used by some other thread to do useful work.

In practice, when a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool. When the thread is finished using the connection, it returns it to the pool, so that it may be used by any other threads that want to use it.

When the connection is loaned out from the pool, it is used exclusively by the thread that requested it. From a programming point of view, it is the same as if your thread called `DriverManager.getConnection()` every time it needed a JDBC connection, however with connection pooling, your thread may end up using either a new, or already-existing connection.

Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage. The main benefits to connection pooling are:

- Reduced connection creation time

  Although this is not usually an issue with the quick connection setup that MySQL offers compared to other databases, creating new JDBC connections still incurs networking and JDBC driver overhead that will be avoided if connections are recycled.

- Simplified programming model

  When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straight-forward JDBC programming techniques.

- Controlled resource usage

  If you don't use connection pooling, and instead create a new connection every time a thread needs one, your application's resource usage can be quite wasteful and lead to unpredictable behavior under load.

Remember that each connection to MySQL has overhead (memory, CPU, context switches, and so forth) on both the client and server side. Every connection limits how many resources there are available to your application as well as the MySQL server. Many of these resources will be used whether or not the connection is actually doing any useful work!

Connection pools can be tuned to maximize performance, while keeping resource utilization below the point where your application will start to fail rather than just run slower.

Luckily, Sun has standardized the concept of connection pooling in JDBC through the JDBC-2.0 Optional interfaces, and all major application servers have implementations of these APIs that work fine with MySQL Connector/J.

Generally, you configure a connection pool in your application server configuration files, and access it via the Java Naming and Directory Interface (JNDI). The following code shows how you might use a connection pool from an application deployed in a J2EE application server:

Ejemplo 25.11. Using a connection pool with a J2EE application server

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;


public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         *  lookup  the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */

        InitialContext ctx = new InitialContext();

         /*
          * Lookup the DataSource, which will be backed by a pool
          * that the application server provides. DataSource instances
          * are also a good candidate for caching as an instance
          * variable, as JNDI lookups can be expensive as well.
          */

        DataSource ds =
          (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");

        /*
         * The following code is what would actually be in your
         * Servlet, JSP or EJB 'service' method...where you need
         * to work with a JDBC connection.
         */

        Connection conn = null;
        Statement stmt = null;

        try {
            conn = ds.getConnection();

            /*
```

```
             * Now, use normal JDBC programming to work with
             * MySQL, making sure to close each resource when you're
             * finished with it, which allows the connection pool
             * resources to be recovered as quickly as possible
             */

            stmt = conn.createStatement();
            stmt.execute("SOME SQL QUERY");

            stmt.close();
            stmt = null;

            conn.close();
            conn = null;
        } finally {
            /*
             * close any jdbc instances here that weren't
             * explicitly closed during normal code path, so
             * that we don't 'leak' resources...
             */

            if (stmt != null) {
                try {
                    stmt.close();
                } catch (sqlexception sqlex) {
                    // ignore -- as we can't do anything about it here
                }

                stmt = null;
            }

            if (conn != null) {
                try {
                    conn.close();
                } catch (sqlexception sqlex) {
                    // ignore -- as we can't do anything about it here
                }

                conn = null;
            }
        }
    }
}
```

As shown in the example above, after obtaining the JNDI InitialContext, and looking up the DataSource, the rest of the code should look familiar to anyone who has done JDBC programming in the past.

The most important thing to remember when using connection pooling is to make sure that no matter what happens in your code (exceptions, flow-of-control, and so forth), connections, and anything created by them (such as statements or result sets) are closed, so that they may be re-used, otherwise they will be stranded, which in the best case means that the MySQL server resources they represent (such as buffers, locks, or sockets) may be tied up for some time, or worst case, may be tied up forever.

What's the Best Size for my Connection Pool?

As with all other configuration rules-of-thumb, the answer is: it depends. Although the optimal size depends on anticipated load and average database transaction time, the optimum connection pool size is smaller than you might expect. If you take Sun's Java Petstore blueprint application for example, a connection pool of 15-20 connections can serve a relatively moderate load (600 concurrent users) using MySQL and Tomcat with response times that are acceptable.

To correctly size a connection pool for your application, you should create load test scripts with tools such as Apache JMeter or The Grinder, and load test your application.

An easy way to determine a starting point is to configure your connection pool's maximum number of connections to be unbounded, run a load test, and measure the largest amount of concurrently used connections. You can then work backward from there to determine what values of minimum and maximum pooled connections give the best performance for your particular application.

Using Connector/J with Tomcat

The following instructions are based on the instructions for Tomcat-5.x, available at http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html which is current at the time this document was written.

First, install the .jar file that comes with Connector/J in `$CATALINA_HOME/common/lib` so that it is available to all applications installed in the container.

Next, Configure the JNDI DataSource by adding a declaration resource to `$CATALINA_HOME/conf/server.xml` in the context that defines your web application:

```
<Context ....>

  ...

  <Resource name="jdbc/MySQLDB"
              auth="Container"
              type="javax.sql.DataSource"/>

  <!-- The name you used above, must match _exactly_ here!

       The connection pool will be bound into JNDI with the name
       "java:/comp/env/jdbc/MySQLDB"
  -->

  <ResourceParams name="jdbc/MySQLDB">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

    <parameter>
      <name>maxActive</name>
      <value>10</value>
    </parameter>

    <!-- You don't want to many idle connections hanging around
         if you can avoid it, only enough to soak up a spike in
         the load -->

    <parameter>
      <name>maxIdle</name>
      <value>5</value>
    </parameter>

    <!-- Don't use autoReconnect=true, it's going away eventually
         and it's a crutch for older connection pools that couldn't
         test connections. You need to decide whether your application
         is supposed to deal with SQLExceptions (hint, it should), and
         how much of a performance penalty you're willing to pay
         to ensure 'freshness' of the connection -->

    <parameter>
      <name>validationQuery</name>
      <value>SELECT 1</value>
```

```
 </parameter>

<!-- The most conservative approach is to test connections
     before they're given to your application. For most applications
     this is okay, the query used above is very small and takes
     no real server resources to process, other than the time used
     to traverse the network.

     If you have a high-load application you'll need to rely on
     something else. -->

 <parameter>
   <name>testOnBorrow</name>
   <value>true</value>
 </parameter>

<!-- Otherwise, or in addition to testOnBorrow, you can test
     while connections are sitting idle -->

 <parameter>
   <name>testWhileIdle</name>
   <value>true</value>
 </parameter>

 <!-- You have to set this value, otherwise even though
      you've asked connections to be tested while idle,
      the idle evicter thread will never run -->

 <parameter>
   <name>timeBetweenEvictionRunsMillis</name>
   <value>10000</value>
 </parameter>

 <!-- Don't allow connections to hang out idle too long,
      never longer than what wait_timeout is set to on the
      server...A few minutes or even fraction of a minute
      is sometimes okay here, it depends on your application
      and how much spikey load it will see -->

 <parameter>
   <name>minEvictableIdleTimeMillis</name>
   <value>60000</value>
 </parameter>

 <!-- Username and password used when connecting to MySQL -->

 <parameter>
  <name>username</name>
  <value>someuser</value>
 </parameter>

 <parameter>
  <name>password</name>
  <value>somepass</value>
 </parameter>

 <!-- Class name for the Connector/J driver -->

 <parameter>
    <name>driverClassName</name>
    <value>com.mysql.jdbc.Driver</value>
 </parameter>

 <!-- The JDBC connection url for connecting to MySQL, notice
      that if you want to pass any other MySQL-specific parameters
      you should pass them here in the URL, setting them using the
      parameter tags above will have no effect, you will also
```

```
         need to use &amp;amp; to separate parameter values as the
         ampersand is a reserved character in XML -->

    <parameter>
      <name>url</name>
      <value>jdbc:mysql://localhost:3306/test</value>
    </parameter>

  </ResourceParams>
</Context>
```

In general, you should follow the installation instructions that come with your version of Tomcat, as the way you configure datasources in Tomcat changes from time-to-time, and unfortunately if you use the wrong syntax in your XML file, you will most likely end up with an exception similar to the following:

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null ' SQL
state: null
```

Using Connector/J with JBoss

These instructions cover JBoss-4.x. To make the JDBC driver classes available to the application server, copy the .jar file that comes with Connector/J to the `lib` directory for your server configuration (which is usually called `default`). Then, in the same configuration directory, in the subdirectory named deploy, create a datasource configuration file that ends with "-ds.xml", which tells JBoss to deploy this file as a JDBC Datasource. The file should have the following contents:

```
<datasources>
    <local-tx-datasource>
        <!-- This connection pool will be bound into JNDI with the name
             "java:/MySQLDB" -->

        <jndi-name>MySQLDB</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>user</user-name>
        <password>pass</password>

        <min-pool-size>5</min-pool-size>

        <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

        <max-pool-size>20</max-pool-size>

        <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->

        <idle-timeout-minutes>5</idle-timeout-minutes>

        <!-- If you're using Connector/J 3.1.8 or newer, you can use
             our implementation of these to increase the robustness
             of the connection pool. -->

        <exception-sorter-class-name>com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter</exception-
        <valid-connection-checker-class-name>com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker</val

    </local-tx-datasource>
</datasources>
```

25.4.5.3. Common Problems and Solutions

There are a few issues that seem to be commonly encountered often by users of MySQL Connector/J. This section deals with their symptoms, and their resolutions.

Questions

- 25.4.5.3.1: [1507] When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

  What's going on? I can connect just fine with the MySQL command-line client.

- 25.4.5.3.2: [1508] My application throws an SQLException 'No Suitable Driver'. Why is this happening?

- 25.4.5.3.3: [1508] I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

- 25.4.5.3.4: [1508] I have a servlet/application that works fine for a day, and then stops working overnight

- 25.4.5.3.5: [1511] I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.

- 25.4.5.3.6: [1511] I cannot connect to the MySQL server using Connector/J, and I'm sure the connection paramters are correct.

Questions and Answers

25.4.5.3.1: When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What's going on? I can connect just fine with the MySQL command-line client.

MySQL Connector/J must use TCP/IP sockets to connect to MySQL, as Java does not support Unix Domain Sockets. Therefore, when MySQL Connector/J connects to MySQL, the security manager in MySQL server will use its grant tables to determine whether the connection should be allowed.

You must add the necessary security credentials to the MySQL server for this to happen, using the `GRANT` statement to your MySQL Server. See Sección 13.5.1.3, “Sintaxis de GRANT y REVOKE”, for more information.

Note. Testing your connectivity with the `mysql` command-line client will not work unless you add the `--host` flag, and use something other than `localhost` for the host. The `mysql` command-line client will

use Unix domain sockets if you use the special hostname `localhost`. If you are testing connectivity to `localhost`, use `127.0.0.1` as the hostname instead.

Warning. Changing privileges and permissions improperly in MySQL can potentially cause your server installation to not have optimal security properties.

25.4.5.3.2: My application throws an SQLException 'No Suitable Driver'. Why is this happening?

There are three possible causes for this error:

- The Connector/J driver is not in your `CLASSPATH`, see Sección 25.4.2, “Connector/J Installation”.
- The format of your connection URL is incorrect, or you are referencing the wrong JDBC driver.
- When using DriverManager, the `jdbc.drivers` system property has not been populated with the location of the Connector/J driver.

25.4.5.3.3: I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

Either you're running an Applet, your MySQL server has been installed with the "--skip-networking" option set, or your MySQL server has a firewall sitting in front of it.

Applets can only make network connections back to the machine that runs the web server that served the .class files for the applet. This means that MySQL must run on the same machine (or you must have some sort of port re-direction) for this to work. This also means that you will not be able to test applets from your local file system, you must always deploy them to a web server.

MySQL Connector/J can only communicate with MySQL using TCP/IP, as Java does not support Unix domain sockets. TCP/IP communication with MySQL might be affected if MySQL was started with the "--skip-networking" flag, or if it is firewalled.

If MySQL has been started with the "--skip-networking" option set (the Debian Linux package of MySQL server does this for example), you need to comment it out in the file /etc/mysql/my.cnf or /etc/my.cnf. Of course your my.cnf file might also exist in the `data` directory of your MySQL server, or anywhere else (depending on how MySQL was compiled for your system). Binaries created by MySQL AB always look in /etc/my.cnf and [datadir]/my.cnf. If your MySQL server has been firewalled, you will need to have the firewall configured to allow TCP/IP connections from the host where your Java code is running to the MySQL server on the port that MySQL is listening to (by default, 3306).

25.4.5.3.4: I have a servlet/application that works fine for a day, and then stops working overnight

MySQL closes connections after 8 hours of inactivity. You either need to use a connection pool that handles stale connections or use the "autoReconnect" parameter (see Sección 25.4.4.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”).

Also, you should be catching SQLExceptions in your application and dealing with them, rather than propagating them all the way until your application exits, this is just good programming practice. MySQL Connector/J will set the SQLState (see `java.sql.SQLException.getSQLState()` in your APIDOCS)

to "08S01" when it encounters network-connectivity issues during the processing of a query. Your application code should then attempt to re-connect to MySQL at this point.

The following (simplistic) example shows what code that can handle these exceptions might look like:

Ejemplo 25.12. Example of transaction with retry logic

```
public void doBusinessOp() throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    //
    // How many times do you want to retry the transaction
    // (or at least _getting_ a connection)?
    //
    int retryCount = 5;

    boolean transactionCompleted = false;

    do {
        try {
            conn = getConnection(); // assume getting this from a
                                    // javax.sql.DataSource, or the
                                    // java.sql.DriverManager

            conn.setAutoCommit(false);

            //
            // Okay, at this point, the 'retry-ability' of the
            // transaction really depends on your application logic,
            // whether or not you're using autocommit (in this case
            // not), and whether you're using transacational storage
            // engines
            //
            // For this example, we'll assume that it's _not_ safe
            // to retry the entire transaction, so we set retry
            // count to 0 at this point
            //
            // If you were using exclusively transaction-safe tables,
            // or your application could recover from a connection going
            // bad in the middle of an operation, then you would not
            // touch 'retryCount' here, and just let the loop repeat
            // until retryCount == 0.
            //
            retryCount = 0;

            stmt = conn.createStatement();

            String query = "SELECT foo FROM bar ORDER BY baz";

            rs = stmt.executeQuery(query);

            while (rs.next()) {
            }

            rs.close();
            rs = null;

            stmt.close();
            stmt = null;

            conn.commit();
            conn.close();
            conn = null;
```

```
            transactionCompleted = true;
        } catch (SQLException sqlEx) {

            //
            // The two SQL states that are 'retry-able' are 08S01
            // for a communications error, and 40001 for deadlock.
            //
            // Only retry if the error was due to a stale connection,
            // communications problem or deadlock
            //

            String sqlState = sqlEx.getSQLState();

            if ("08S01".equals(sqlState) || "40001".equals(sqlState)) {
                retryCount--;
            } else {
                retryCount = 0;
            }
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this . . .
                }
            }

            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this as well . . .
                }
            }

            if (conn != null) {
                try {
                    //
                    // If we got here, and conn is not null, the
                    // transaction should be rolled back, as not
                    // all work has been done

                    try {
                        conn.rollback();
                    } finally {
                        conn.close();
                    }
                } catch (SQLException sqlEx) {
                    //
                    // If we got an exception here, something
                    // pretty serious is going on, so we better
                    // pass it up the stack, rather than just
                    // logging it. . .

                    throw sqlEx;
                }
            }
        }
    } while (!transactionCompleted && (retryCount > 0));
}
```

Note. Use of the `autoReconnect` option is not recommended because there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information. Instead, you should use a connection pool which will enable your application to connect to the MySQL server using an available connection from the pool. The `autoReconnect` facility is deprecated, and may be removed in a future release.

25.4.5.3.5: I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.

Because MySQL does not have row identifiers, MySQL Connector/J can only update result sets that have come from queries on tables that have at least one primary key, the query must select every primary key and the query can only span one table (that is, no joins). This is outlined in the JDBC specification.

Note that this issue only occurs when using updatable result sets, and is caused because Connector/J is unable to guarantee that it can identify the correct rows within the result set to be updated without having a unique reference to each row. There is no requirement to have a unique field on a table if you are using `UPDATE` or `DELETE` statements on a table where you can individually specify the criteria to be matched using a `WHERE` clause.

25.4.5.3.6: I cannot connect to the MySQL server using Connector/J, and I'm sure the connection paramters are correct.

Make sure that the `skip-networking` option has not been enabled on your server. Connector/J must be able to communicate with your server over TCP/IP, named sockets are not supported. Also ensure that you are not filtering connections through a Firewall or other network security system. For more informaiton, see Sección A.2.2, “`Can't connect to [local] MySQL server`”.

25.4.6. Connector/J Support

25.4.6.1. Connector/J Community Support

MySQL AB provides assistance to the user community by means of its mailing lists. For Connector/J related issues, you can get help from experienced users by using the MySQL and Java mailing list. Archives and subscription information is available online at http://lists.mysql.com/java.

For information about subscribing to MySQL mailing lists or to browse list archives, visit http://lists.mysql.com/. See Sección 1.6.1.1, “Las listas de correo de MySQL”.

Community support from experienced users is also available through the JDBC Forum. You may also find help from other users in the other MySQL Forums, located at http://forums.mysql.com. See Sección 1.6.3, “Soporte por parte de la comunidad en los foros de MySQL”.

25.4.6.2. How to Report Connector/J Bugs or Problems

The normal place to report bugs is 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 will also be able to enter new reports.

If you have found a sensitive security bug in MySQL, you can send email to security_at_mysql.com.

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 will help you write your report correctly so that you don't waste your time doing things that may not help us much or at all.

If you have a repeatable bug report, please report it to the bugs database at http://bugs.mysql.com/. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message 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 don't matter.

A good principle is this: 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 Connector/J or MySQL used, and (b) not fully describing the platform on which Connector/J is installed (including the JVM version, and the platform type and version number that MySQL itself is installed on).

This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. 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 already been fixed in newer MySQL versions.

Sometimes the error is 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 at all possible, you should create a repeatable, stanalone testcase that doesn't involve any third-party classes.

To streamline this process, we ship a base class for testcases with Connector/J, named '`com.mysql.jdbc.util.BaseBugReport`'. To create a testcase for Connector/J using this class, create your own class that inherits from `com.mysql.jdbc.util.BaseBugReport` and override the methods `setUp()`, `tearDown()` and `runTest()`.

In the `setUp()` method, create code that creates your tables, and populates them with any data needed to demonstrate the bug.

In the `runTest()` method, create code that demonstrates the bug using the tables and data you created in the `setUp` method.

In the `tearDown()` method, drop any tables you created in the `setUp()` method.

In any of the above three methods, you should use one of the variants of the `getConnection()` method to create a JDBC connection to MySQL:

- `getConnection()` - Provides a connection to the JDBC URL specified in `getUrl()`. If a connection already exists, that connection is returned, otherwise a new connection is created.
- `getNewConnection()` - Use this if you need to get a new connection for your bug report (i.e. there's more than one connection involved).
- `getConnection(String url)` - Returns a connection using the given URL.
- `getConnection(String url, Properties props)` - Returns a connection using the given URL and properties.

If you need to use a JDBC URL that is different from 'jdbc:mysql:///test', override the method `getUrl()` as well.

Use the `assertTrue(boolean expression)` and `assertTrue(String failureMessage, boolean expression)` methods to create conditions that must be met in your testcase demonstrating the behavior you are expecting (vs. the behavior you are observing, which is why you are most likely filing a bug report).

Finally, create a `main()` method that creates a new instance of your testcase, and calls the `run` method:

```
public static void main(String[] args) throws Exception {
      new MyBugReport().run();
 }
```

Once you have finished your testcase, and have verified that it demonstrates the bug you are reporting, upload it with your bug report to http://bugs.mysql.com/.

25.4.6.3. Connector/J Change History

The Connector/J Change History (Changelog) is located with the main Changelog for MySQL. See Sección C.5, “MySQL Connector/J Change History”.

25.5. MySQL Connector/MXJ

MySQL Connector/MXJ is a solution for deploying the MySQL database engine (`mysqld`) intelligently from within a Java package.

25.5.1. Introduction to Connector/MXJ

MySQL Connector/MXJ is a Java Utility package for deploying and managing a MySQL database. Deploying and using MySQL can be as easy as adding an additional parameter to the JDBC connection url, which will result in the database being started when the first connection is made. This makes it easy for Java developers to deploy applications which require a database by reducing installation barriers for their end-users.

MySQL Connector/MXJ makes the MySQL database appear to be a java-based component. It does this by determining what platform the system is running on, selecting the appropriate binary, and launching the executable. It will also optionally deploy an initial database, with any specified parameters.

Included are instructions for use with a JDBC driver and deploying as a JMX MBean to JBoss.

You can download sources and binaries from: http://dev.mysql.com/downloads/connector/mxj/

This a beta release and feedback is welcome and encouraged.

Please send questions or comments to the MySQL and Java mailing list.

25.5.1.1. Connector/MXJ Versions

Connector/MX

- Connector/MXJ 5.x, currently in beta status, includes `mysqld` version 5.x and includes binaries for Linux x86, Mac OS X PPC, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 5.x requires the Connector/J 5.x package.

  The exact version of `mysqld` included depends on the version of Connector/MXJ

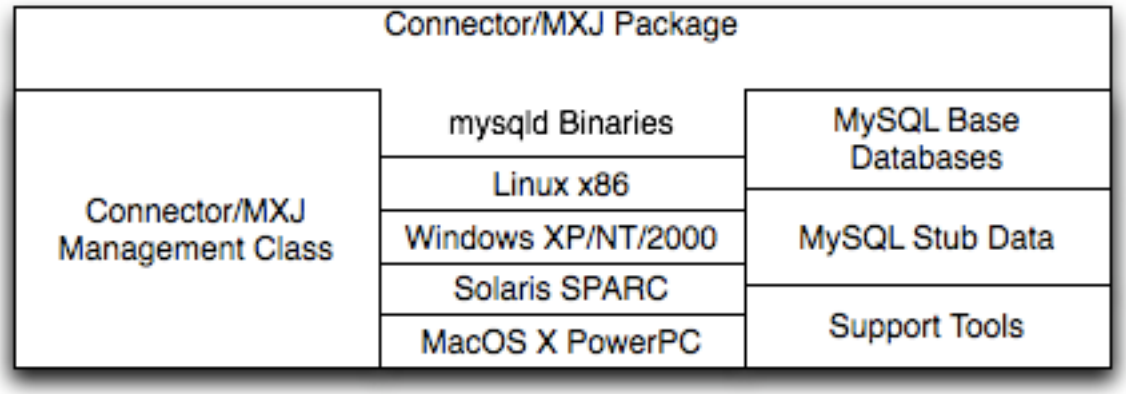
  1. Connector/MXJ v5.0.3 included MySQL v5.0.22
  2. Connector/MXJ v5.0.4 includes MySQL v5.0.27 (Community) or MySQL v5.0.32 (Enterprise)

- Connector/MXJ 1.x includes `mysqld` version 4.1.13 and includes binaries for Linux x86, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 1.x requires the Connector/J 3.x package.

This guide provides information on the Connector/MXJ 5.x release. For information on using the older releases, please see the documentation included with the appropriate distribution.

25.5.1.2. Connector/MXJ Overview

Connector/MXJ consists of a Java class, a copy of the `mysqld` binary for a specific list of platforms, and associated files and support utilities. The Java class controls the initialization of an instance of the embedded `mysqld` binary, and the ongoing management of the `mysqld` process. The entire sequence and management can be controlled entirely from within Java using the Connector/MXJ Java classes. You can see an overview of the contents of the Connector/MXJ package in the figure below.



It is important to note that Connector/MXJ is not an embedded version of MySQL, or a version of MySQL written as part of a Java class. Connector/MXJ works through the use of an embedded, compiled binary of `mysqld` as would normally be used when deploying a standard MySQL installation.

It is the Connector/MXJ wrapper, support classes and tools, that enable Connector/MXJ to appear as a MySQL instance.

When Connector/MXJ is initialized, the corresponding `mysqld` binary for the current platform is extracted, along with a pre-configured data directed. Both are contained within the Connector/MXJ JAR file. The `mysqld` instance is then started, with any additional options as specified during the initialization, and the MySQL database becomes accessible.

Because Connector/MXJ works in combination with Connector/J, you can access and integrate with the MySQL instance through a JDBC connection. When you have finished with the server, the instance is terminated, and, by default, any data created during the session is retained within the temporary directory created when the instance was started.

Connector/MXJ and the embedded `mysqld` instance can be deployed in a number of environments where relying on an existing database, or installing a MySQL instance would be impossible, including CD-ROM embedded database applications and temporary database requirements within a Java-based application environment.

25.5.2. Connector/MXJ Installation

Connector/MXJ does not have a installation application or process, but there are some steps you can follow to make the installation and deployment of Connector/MXJ easier.

Before you start, there are some baseline requirements for

- Java Runtime Environment (v1.4.0 or newer) if you are only going to deploy the package.
- Java Development Kit (v1.4.0 or newer) if you want to build Connector/MXJ from source.

- Connector/J 5.0 or newer.

Depending on your target installation/deployment environment you may also require:

- JBoss - 4.0rc1 or newer
- Apache Tomcat - 5.0 or newer
- Sun's JMX reference implementation version 1.2.1 (from http://java.sun.com/products/JavaManagement/)

25.5.2.1. Supported Platforms

Connector/MXJ is compatible with any platform supporting Java and MySQL. By default, Connector/MXJ incorporates the `mysqld` binary for a select number of platforms, as outlined below.

- Linux, i386
- Windows NT, Windows 2000, Windows XP, x86
- Solaris 8, SPARC 32 (compatible with Solaris 8, Solaris 9 and Solaris 10 on SPARC 32-bit and 64-bit platforms)
- Mac OS X, PowerPC

For more information on packaging your own Connector/MXJ with the platforms you require, see Sección 25.5.5.1, “Creating your own Connector/MXJ Package”

25.5.2.2. Connector/MXJ Base Installation

Because there is no formal installation process, the method, installation directory, and access methods you use for Connector/MXJ are entirely up to your individual requirements.

To perform a basic installation, choose a target directory for the files included in the Connector/MXJ package. On Unix/Linux systems you may opt to use a directory such as `/usr/local/connector-mxj`; On Windows, you may want to install the files in the base directory, `C:\Connector-MXJ`, or within the `Program Files` directory.

To install the files, for a Connector/MXJ 5.0.4 installation:

1. Download the Connector/MXJ package, either in Tar/Gzip format (ideal for Unix/Linux systems) or Zip format (Windows).
2. Extract the files from the package. This will create a directory `connector-mxj`. Copy and optionally rename this directory to your desired location.
3. For best results, you should update your global `CLASSPATH` variable with the location of the required `jar` files.

   Within Unix/Linux you can do this globally by editing the global shell profile, or on a user by user basis by editing their individual shell profile.

   On Windows 2000, Windows NT and Windows XP, you can edit the global `CLASSPATH` by editing the `Environment Variables` configured through the `System` control panel.

For Connector/MXJ 5.0.4 and later you need the following JAR files in your `CLASSPATH`

1. `connector-mxj.jar` — contains the main Connector/MXJ classes.
2. `connector-mxj-db-files.jar` — contains the embedded `mysqld` and database files.

3. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.

4. `connector-j.jar` — Connector/J, see Sección 25.4, “MySQL Connector/J”.

For Connector/MXJ 5.0.3 and earlier, you need the following JAR files:

1. `connector-mxj.jar`

2. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.

3. `connector-j.jar` — Connector/J, see Sección 25.4, “MySQL Connector/J”.

25.5.2.3. Connector/MXJ Quick Start Guide

Once you have extracted the Connector/MXJ and Connector/J components you can run one of the sample applications that initiates a MySQL instance. You can test the installation by running the `ConnectorMXJUrlTestExample`:

```
java ConnectorMXJUrlTestExample
jdbc:mysql:mxj://localhost:3336/test?server.basedir=/var/tmp/test-mxj
[MysqldResource] launching mysqld (getOptions)
[/var/tmp/test-mxj/bin/mysqld][--no-defaults] »
[--pid-file=/var/tmp/test-mxj/data/MysqldResource.pid] »
[--socket=mysql.sock][--datadir=/var/tmp/test-mxj/data] »
[--port=3336][--basedir=/var/tmp/test-mxj]
[MysqldResource] launching mysqld (driver_launched_mysqld_1)
InnoDB: The first specified data file ./ibdata1 did not exist:
InnoDB: a new database to be created!
060726 15:40:42  InnoDB: Setting file ./ibdata1 size to 10 MB
InnoDB: Database physically writes the file full: wait...
060726 15:40:43  InnoDB: Log file ./ib_logfile0 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile0 size to 5 MB
InnoDB: Database physically writes the file full: wait...
060726 15:40:43  InnoDB: Log file ./ib_logfile1 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
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
060726 15:40:44  InnoDB: Started; log sequence number 0 0
060726 15:40:44 [Note] /var/tmp/test-mxj/bin/mysqld: ready for connections.
Version: '5.0.22-max'  socket: 'mysql.sock'  port: 3336  MySQL Community Edition - Experimental (GPL)
[MysqldResource] mysqld running as process: 1210
------------------------

5.0.22-max

------------------------
[MysqldResource] stopping mysqld (process: 1210)
060726 15:40:44 [Note] /var/tmp/test-mxj/bin/mysqld: Normal shutdown

060726 15:40:45  InnoDB: Starting shutdown...
060726 15:40:48  InnoDB: Shutdown completed; log sequence number 0 43655
060726 15:40:48 [Note] /var/tmp/test-mxj/bin/mysqld: Shutdown complete

[MysqldResource] clearing options
[MysqldResource] shutdown complete
```

The above output shows an instance of MySQL starting, the necessary files being created (log files, InnoDB data files) and the MySQL database entering the running state. The instance is then shutdown by Connector/MXJ before the example terminates.

25.5.2.4. Deploying Connector/MXJ using Driver Launch

Connector/MXJ and Connector/J work together to enable you to launch an instance of the `mysqld` server through the use of a keyword in the JDBC connection string. Deploying Connector/MXJ within a Java application can be automated through this method, making the deployment of Connector/MXJ a simple process:

1. Download and unzip Connector/MXJ, add `connector-mxj.jar` to the `CLASSPATH`.

   If you are using Connector/MXJ v5.0.4 or later you will also need to add the `connector-mxj-db-files.jar` file to your `CLASSPATH`.

2. To the JDBC connection string, embed the `mxj` keyword, for example: `jdbc:mysql:mxj://localhost:PORT/DBNAME`.

For more details, see Sección 25.5.3, “Connector/MXJ Configuration”.

25.5.2.5. Deploying Connector/MXJ within JBoss

For deployment within a JBoss environment, you must configure the JBoss environment to use the Connector/MXJ component within the JDBC parameters:

1. Download Connector/MXJ and copy the `connector-mxj.jar` file to the `$JBOSS_HOME/server/default/lib` directory.

   If you are using Connector/MXJ v5.0.4 or later you will also need to copy the `connector-mxj-db-files.jar` file to `$JBOSS_HOME/server/default/lib`.

2. Download Connector/J and copy the `mysql-connector-java-5.0.4-bin.jar` file to the `$JBOSS_HOME/server/default/lib` directory.

3. Create an MBean service xml file in the `$JBOSS_HOME/server/default/deploy` directory with any attributes set, for instance the `datadir` and `autostart`.

4. Set the JDBC parameters of your web application to use:

```
String driver = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql:///test?propertiesTransform="+
             "com.mysql.management.jmx.ConnectorMXJPropertiesTransform";
String user = "root";
String password = "";
Class.forName(driver);
Connection conn = DriverManager.getConnection(url, user, password);
```

You may wish to create a separate users and database table spaces for each application, rather than using "root and test".

We highly suggest having a routine backup procedure for backing up the database files in the `datadir`.

25.5.2.6. Verifying Installation using JUnit

The best way to ensure that your platform is supported is to run the JUnit tests. These will test the Connector/MXJ classes and the associated components.

JUnit Test Requirements

The first thing to do is make sure that the components will work on the platform. The `MysqldResource` class is really a wrapper for a native version of MySQL, so not all platforms are supported. At the time of

this writing, Linux on the i386 architecture has been tested and seems to work quite well, as does OS X v10.3. There has been limited testing on Windows and Solaris.

Requirements:

1. JDK-1.4 or newer (or the JRE if you aren't going to be compiling the source or JSPs).
2. MySQL Connector/J version 5.0 or newer (from http://dev.mysql.com/downloads/connector/j/) installed and available via your CLASSPATH.
3. The `javax.management` classes for JMX version 1.2.1, these are present in the following application servers:
   - JBoss - 4.0rc1 or newer.
   - Apache Tomcat - 5.0 or newer.
   - Sun's JMX reference implementation version 1.2.1 (from http://java.sun.com/products/JavaManagement/).
4. JUnit 3.8.1 (from http://www.junit.org/).

If building from source, All of the requirements from above, plus:

1. Ant version 1.5 or newer (download from http://ant.apache.org/).

Running the JUnit Tests

1. The tests attempt to launch MySQL on the port 3336. If you have a MySQL running, it may conflict, but this isn't very likely because the default port for MySQL is 3306. However, You may set the "c-mxj_test_port" Java property to a port of your choosing. Alternatively, you may wish to start by shutting down any instances of MySQL you have running on the target machine.

   The tests suppress output to the console by default. For verbose output, you may set the "c-mxj_test_silent" Java property to "false".

2. To run the JUnit test suite, the $CLASSPATH must include the following:
   - JUnit
   - JMX
   - Connector/J
   - MySQL Connector/MXJ
3. If `connector-mxj.jar` is not present in your download, unzip MySQL Connector/MXJ source archive.

   ```
   cd mysqldjmx
   ant dist
   ```

   Then add `$TEMP/cmxj/stage/connector-mxj/connector-mxj.jar` to the CLASSPATH.

4. if you have `junit`, execute the unit tests. From the command line, type:

   ```
   java junit.textui.TestRunner com.mysql.management.AllTestsSuite
   ```

The output should look something like this:

```
.....................................
.....................................
.........
Time: 259.438

OK (101 tests)
```

Note that the tests are a bit slow near the end, so please be patient.

25.5.3. Connector/MXJ Configuration

25.5.3.1. Running as part of the JDBC Driver

A feature of the MySQL Connector/J JDBC driver is the ability to specify a connection to an embedded Connector/MXJ instance through the use of the mxj keyword in the JDBC connection string.

In the following example, we have a program which creates a connection, executes a query, and prints the result to the System.out. The MySQL database will be deployed and started as part of the connection process, and shutdown as part of the finally block.

You can find this file in the Connector/MXJ package as `src/ConnectorMXJUrlTestExample.java`.

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

import com.mysql.management.driverlaunched.ServerLauncherSocketFactory;

public class ConnectorMXJUrlTestExample {
    public static String DRIVER = "com.mysql.jdbc.Driver";

    public static String JAVA_IO_TMPDIR = "java.io.tmpdir";

    public static void main(String[] args) throws Exception {
        File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR));
        File databaseDir = new File(ourAppDir, "test-mxj");
        int port = 3336;

        String url = "jdbc:mysql:mxj://localhost:" + port + "/test" + "?"
                + "server.basedir=" + databaseDir;

        System.out.println(url);

        String userName = "root";
        String password = "";

        Class.forName(DRIVER);
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url, userName, password);
            printQueryResults(conn, "SELECT VERSION()");
        } finally {
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception e) {
                e.printStackTrace();
```

```
            }

            ServerLauncherSocketFactory.shutdown(databaseDir, null);
        }
    }

    public static void printQueryResults(Connection conn, String SQLquery)
            throws Exception {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(SQLquery);
        int columns = rs.getMetaData().getColumnCount();
        System.out.println("------------------------");
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= columns; i++) {
                System.out.println(rs.getString(i));
            }
            System.out.println();
        }
        rs.close();
        stmt.close();
        System.out.println("------------------------");
        System.out.flush();
        Thread.sleep(100); // wait for System.out to finish flush
    }
}
```

To run the above program, be sure to have connector-mxj.jar and Connector/J in the CLASSPATH. Then type:

```
java ConnectorMXJTestExample
```

25.5.3.2. Running within a Java Object

If you have a java application and wish to “embed” a MySQL database, make use of the com.mysql.management.MysqldResource class directly. This class may be instantiated with the default (no argument) constructor, or by passing in a java.io.File object representing the directory you wish the server to be "unzipped" into. It may also be instantiated with printstreams for "stdout" and "stderr" for logging.

Once instantiated, a java.util.Map, the object will be able to provide a java.util.Map of server options appropriate for the platform and version of MySQL which you will be using.

The MysqldResource enables you to "start" MySQL with a java.util.Map of server options which you provide, as well as "shutdown" the database. The following example shows a simplistic way to embed MySQL in an application using plain java objects.

You can find this file in the Connector/MXJ package as `src/ConnectorMXJObjectTestExample.java`.

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;

import com.mysql.management.MysqldResource;

public class ConnectorMXJObjectTestExample {
    public static String DRIVER = "com.mysql.jdbc.Driver";
```

```
    public static String JAVA_IO_TMPDIR = "java.io.tmpdir";

    public static void main(String[] args) throws Exception {
        File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR));
        File databaseDir = new File(ourAppDir, "mysql-mxj");
        int port = 3336;

        MysqldResource mysqldResource = startDatabase(databaseDir, port);

        String userName = "root";
        String password = "";

        Class.forName(DRIVER);
        Connection conn = null;
        try {
            String url = "jdbc:mysql://localhost:" + port + "/test";
            conn = DriverManager.getConnection(url, userName, password);
            printQueryResults(conn, "SELECT VERSION()");
        } finally {
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                mysqldResource.shutdown();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static MysqldResource startDatabase(File databaseDir, int port) {
        MysqldResource mysqldResource = new MysqldResource(databaseDir);

        Map database_options = new HashMap();
        database_options.put("port", Integer.toString(port));
        mysqldResource.start("test-mysqld-thread", database_options);

        if (!mysqldResource.isRunning()) {
            throw new RuntimeException("MySQL did not start.");
        }

        System.out.println("MySQL is running.");

        return mysqldResource;
    }

    public static void printQueryResults(Connection conn, String SQLquery)
            throws Exception {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(SQLquery);
        int columns = rs.getMetaData().getColumnCount();
        System.out.println("------------------------");
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= columns; i++) {
                System.out.println(rs.getString(i));
            }
            System.out.println();
        }
        rs.close();
        stmt.close();
        System.out.println("------------------------");
        System.out.flush();
```

```
        Thread.sleep(100); // wait for System.out to finish flush
    }
}
```

25.5.3.3. Setting server options

Of course there are many options we may wish to set for a MySQL database. These options may be specified as part of the JDBC connection string simply by prefixing each server option with "server.". In the following example we set two driver parameters and two server parameters:

```
        String url = "jdbc:mysql://" + hostColonPort + "/"
                + "?"
                + "cacheServerConfiguration=true"
                + "&"
                + "useLocalSessionState=true"
                + "&"
                + "server.basedir=/opt/myapp/db"
                + "&"
                + "server.datadir=/mnt/bigdisk/myapp/data";
```

25.5.4. Connector/MXJ Reference

25.5.4.1. MysqldResource API

MysqldResource Constructors

The `MysqldResource` class supports three different constructor forms:

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err, Utils util)`

  The most detailed constructor, enables you to set the base directory, data directory, select a server by its version string, standard out and standard error and MySQL utilities class.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err)`

  Enables you to set the base directory, data directory, select a server by its version string, standard out and standard error.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString)`

  Enables you to set the base directory, data directory and select a server by its version string. Output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource(File baseDir, File dataDir)`

  Enables you to set the base directory and data directory. The default MySQL version is selected, and output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource(File baseDir);`

  Allows the setting of the "basedir" to deploy the MySQL files to. Output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource();`

The basedir is defaulted to a subdirectory of the java.io.tempdir. Output for standard out and standard err are directed to System.out and System.err;

MysqldResource Methods

MysqldResource API includes the following methods:

- `void start(String threadName, Map mysqldArgs);`

  Deploys and starts MySQL. The "threadName" string is used to name the thread which actually performs the execution of the MySQL command line. The map is the set of arguments and their values to be passed to the command line.

- `void shutdown();`

  Shuts down the MySQL instance managed by the MysqldResource object.

- `Map getServerOptions();`

  Returns a map of all the options and their current (or default, if not running) options available for the MySQL database.

- `boolean isRunning();`

  Returns true if the MySQL database is running.

- `boolean isReadyForConnections();`

  Returns true once the database reports that is ready for connections.

- `void setKillDelay(int millis);`

  The default “Kill Delay” is 30 seconds. This represents the amount of time to wait between the initial request to shutdown and issuing a “force kill” if the database has not shutdown by itself.

- `void addCompletionListenser(Runnable listener);`

  Allows for applications to be notified when the server process completes. Each "listener" will be fired off in its own thread.

- `String getVersion();`

  Returns the version of MySQL.

- `void setVersion(int MajorVersion, int minorVersion, int patchLevel);`

  The standard distribution comes with only one version of MySQL packaged. However, it is possible to package multiple versions, and specify which version to use.

25.5.5. Connector/MXJ Notes and Tips

This section contains notes and tips on using the Connector/MXJ component within your applications.

25.5.5.1. Creating your own Connector/MXJ Package

If you want to create a custom Connector/MXJ package that includes a specific `mysqld` version or platform then you must extract and rebuild the `connector-mxj.jar` (Connector/MXJ v5.0.3 or earlier) or `connector-mxj-db-files.jar` (Connector/MXJ v5.0.4 or later) file.

First, you should create a new directory into which you can extract the current `connector-mxj.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj.jar
shell> ls
5-0-22/
ConnectorMXJObjectTestExample.class
ConnectorMXJUrlTestExample.class
META-INF/
TestDb.class
com/
kill.exe
```

If you are using Connector/MXJ v5.0.4 or later, you should unpack the `connector-mxj-db-files.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj-db-files.jar
shell> ls
5-0-27/
META-INF/
connector-mxj.properties
```

The MySQL version directory, `5-0-22` or `5-0-27` in the above examples, contains all of the files used to create an instance of MySQL when Connector/MXJ is executed. All of the files in this directory are required for each version of MySQL that you want to embed. Note as well the format of the version number, which uses hyphens instead of periods to separate the version number components.

Within the version specific directory are the platform specific directories, and archives of the `data` and `share` directory required by MySQL for the various platforms. For example, here is the listing for the default Connector/MXJ package:

```
shell>> ls
Linux-i386/
META-INF/
Mac_OS_X-ppc/
SunOS-sparc/
Win-x86/
com/
data_dir.jar
share_dir.jar
win_share_dir.jar
```

Platform specific directories are listed by their OS and platform - for example the `mysqld` for Mac OS X PowerPC is located within the `Mac_OS_X-ppc` directory. You can delete directories from this location that you do not require, and add new directories for additional platforms that you want to support.

To add a platform specific `mysqld`, create a new directory with the corresponding name for your operating system/platform. For example, you could add a directory for Mac OS X/Intel using the directory `Mac_OS_X-i386`.

On Unix systems, you can determine the platform using `uname`:

```
shell> uname -p
i386
```

Now you need to download or compile `mysqld` for the MySQL version and platform you want to include in your custom `connector-mxj.jar` package into the new directory.

Create a file called `version.txt` in the OS/platform directory you have just created that contains the version string/path of the mysqld binary. For example:

```
mysql-5.0.22-osx10.3-i386/bin/mysqld
```

You can now recreate the `connector-mxj.jar` file with the added `mysqld`:

```
shell> cd custom-mxj
shell> jar -cf ../connector-mxj.jar *
```

For Connector/MXJ v5.0.4 and later, you should repackage to the `connector-mxj-db-files.jar`:

```
shell> cd custom-mxj
shell> jar -cf ../connector-mxj-db-files.jar *
```

You should test this package using the steps outlined in Sección 25.5.2.3, “Connector/MXJ Quick Start Guide”.

> **Nota**
>
> Because the `connector-mxj-db-files.jar` file is separate from the main Connector/MXJ classes you can distribute different `connector-mxj-db-files.jar` files to different hotsts or for different projects without having to create a completely new main `connector-mxj.jar` file for each one.

25.5.5.2. Deploying Connector/MXJ with a pre-configured database

To include a pre-configured/populated database within your Connector/MXJ JAR file you must create a custom `data_dir.jar` file, as included within the main `connector-mxj.jar` (Connector/MXJ 5.0.3 or earlier) or `connector-mxj-db-files.jar` (Connector/MXJ 5.0.4 or later) file:

1. First extract the `connector-mxj.jar` file, as outlined in the previous section (see Sección 25.5.5.1, “Creating your own Connector/MXJ Package”).
2. First, create your database and populate the database with the information you require in an existing instance of MySQL - including Connector/MXJ instances. Data file formats are compatible across platforms.
3. Shutdown the instance of MySQL.
4. Create a JAR file of the data directory and databases that you want to include your Connector/MXJ package. You should include the `mysql` database, which includes user authentication information, in addition to the specific databases you want to include. For example, to create a JAR of the `mysql` and `mxjtest` databases:

   ```
   shell> jar -cf ../data_dir.jar mysql mxjtest
   ```

5. For Connector/MXJ 5.0.3 or earlier, copy the `data_dir.jar` file into the extracted `connector-mxj.jar` directory, and then create an archive for `connector-mxj.jar`.

   For Connector/MXJ 5.0.4 or later, copy the `data_dir.jar` file into the extracted `connector-mxj-db-files.jar` directory, and then create an archive for `connector-mxj-db-files.jar`.

Note that if you are create databases using the InnoDB engine, you must include the `ibdata.*` and `ib_logfile*` files within the `data_dir.jar` archive.

25.5.5.3. Running within a JMX Agent (custom)

As a JMX MBean, MySQL Connector/MXJ requires a JMX v1.2 compliant MBean container, such as JBoss version 4. The MBean will uses the standard JMX management APIs to present (and allow the setting of) parameters which are appropriate for that platform.

If you are not using the SUN Reference implementation of the JMX libraries, you should skip this section. Or, if you are deploying to JBoss, you also may wish to skip to the next section.

We want to see the MysqldDynamicMBean in action inside of a JMX agent. In the `com.mysql.management.jmx.sunri` package is a custom JMX agent with two MBeans:

1. the MysqldDynamicMBean, and
2. a com.sun.jdmk.comm.HtmlAdaptorServer, which provides a web interface for manipulating the beans inside of a JMX agent.

When this very simple agent is started, it will allow a MySQL database to be started and stopped with a web browser.

1. Complete the testing of the platform as above.
   - current JDK, JUnit, Connector/J, MySQL Connector/MXJ
   - this section *requires* the SUN reference implementation of JMX
   - `PATH`, `JAVA_HOME`, `ANT_HOME`, `CLASSPATH`
2. If not building from source, skip to next step

   rebuild with the "sunri.present"

   ```
   ant -Dsunri.present=true dist
   re-run tests:
   java junit.textui.TestRunner com.mysql.management.AllTestsSuite
   ```
3. launch the test agent from the command line:

   ```
   java com.mysql.management.jmx.sunri.MysqldTestAgentSunHtmlAdaptor &
   ```
4. from a browser:

   ```
   http://localhost:9092/
   ```
5. under MysqldAgent,

   ```
   select "name=mysqld"
   ```
6. Observe the MBean View
7. scroll to the bottom of the screen press the startMysqld button
8. click `Back to MBean View`
9. scroll to the bottom of the screen press stopMysqld button
10. kill the java process running the Test Agent (jmx server)

25.5.5.4. Deployment in a standard JMX Agent environment (JBoss)

Once there is confidence that the MBean will function on the platform, deploying the MBean inside of a standard JMX Agent is the next step. Included are instructions for deploying to JBoss.

1. Ensure a current version of java development kit (v1.4.x), see above.

- Ensure `JAVA_HOME` is set (JBoss requires `JAVA_HOME`)
- Ensure `JAVA_HOME/bin` is in the `PATH` (You will NOT need to set your CLASSPATH, nor will you need any of the jars used in the previous tests).

2. Ensure a current version of JBoss (v4.0RC1 or better)

```
http://www.jboss.org/index.html
select "Downloads"
select "jboss-4.0.zip"
pick a mirror
unzip ~/dload/jboss-4.0.zip
create a JBOSS_HOME environment variable set to the unzipped directory
unix only:
cd $JBOSS_HOME/bin
chmod +x *.sh
```

3. Deploy (copy) the `connector-mxj.jar` to `$JBOSS_HOME/server/default/lib`.
4. Deploy (copy) `mysql-connector-java-3.1.4-beta-bin.jar` to `$JBOSS_HOME/server/default/lib`.
5. Create a `mxjtest.war` directory in `$JBOSS_HOME/server/default/deploy`.
6. Deploy (copy) `index.jsp` to `$JBOSS_HOME/server/default/deploy/mxjtest.war`.
7. Create a `mysqld-service.xml` file in `$JBOSS_HOME/server/default/deploy`.

```
<?xml version="1.0" encoding="UTF-8"?>
 <server>
  <mbean code="com.mysql.management.jmx.jboss.JBossMysqldDynamicMBean"
     name="mysql:type=service,name=mysqld">
  <attribute name="datadir">/tmp/xxx_data_xxx</attribute>
  <attribute name="autostart">true</attribute>
  </mbean>
 </server>
```

8. Start jboss:
   - on unix: `$JBOSS_HOME/bin/run.sh`
   - on windows: `%JBOSS_HOME%\bin\run.bat`

   Be ready: JBoss sends a lot of output to the screen.
9. When JBoss seems to have stopped sending output to the screen, open a web browser to: `http://localhost:8080/jmx-console`
10. Scroll down to the bottom of the page in the `mysql` section, select the bulleted `mysqld` link.
11. Observe the JMX MBean View page. MySQL should already be running.
12. (If "autostart=true" was set, you may skip this step.) Scroll to the bottom of the screen. You may press the Invoke button to stop (or start) MySQL observe `Operation completed successfully without a return value.` Click `Back to MBean View`
13. To confirm MySQL is running, open a web browser to `http://localhost:8080/mxjtest/` and you should see that

```
SELECT 1
```

returned with a result of

```
1
```

14. Guided by the `$JBOSS_HOME/server/default/deploy/mxjtest.war/index.jsp` you will be able to use MySQL in your Web Application. There is a `test` database and a `root` user (no password) ready to experiment with. Try creating a table, inserting some rows, and doing some selects.

15. Shut down MySQL. MySQL will be stopped automatically when JBoss is stopped, or: from the browser, scroll down to the bottom of the MBean View press the stop service Invoke button to halt the service. Observe `Operation completed successfully without a return value.` Using `ps` or `task manager` see that MySQL is no longer running

As of 1.0.6-beta version is the ability to have the MBean start the MySQL database upon start up. Also, we've taken advantage of the JBoss life-cycle extension methods so that the database will gracefully shut down when JBoss is shutdown.

25.5.6. Connector/MXJ Support

There are a wide variety of options available for obtaining support for using Connector/MXJ. You should contact the Connector/MXJ community for help before reporting a potential bug or problem. See Sección 25.5.6.1, “Connector/MXJ Community Support”.

25.5.6.1. Connector/MXJ Community Support

MySQL AB provides assistance to the user community by means of a number of mailing lists and web based forums.

You can find help and support through the MySQL and Java mailing list.

For information about subscribing to MySQL mailing lists or to browse list archives, visit http://lists.mysql.com/. See Sección 1.6.1.1, “Las listas de correo de MySQL”.

Community support from experienced users is also available through the MyODBC Forum. You may also find help from other users in the other MySQL Forums, located at http://forums.mysql.com. See Sección 1.6.3, “Soporte por parte de la comunidad en los foros de MySQL”.

25.5.6.2. How to Report Connector/MXJ Problems

If you encounter difficulties or problems with Connector/MXJ, contact the Connector/MXJ community Sección 25.5.6.1, “Connector/MXJ Community Support”.

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version
- Connector/MXJ version
- MySQL server version
- Copies of error messages or other unexpected output
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through http://bugs.mysql.com/.

25.6. Connector/PHP

La distribución y documentación de PHP están disponibles en el sitio Web de PHP. MySQL proporciona las extensiones `mysql` y `mysqli` para el sistema Windows en http://dev.mysql.com/downloads/connector/php/ para MySQL versión 4.1.16 y superiores, MySQL 5.0.18, y MySQL 5.1. Puede encontrar información sobre por qué es preferible usar las extensiones proporcionadas por MySQL en este mísmo sitio. Para el resto de las plataformas, deberá usar las extensiones `mysql` o `mysqli` incluidas con los fuentes de PHP. Ver Sección 24.3, “API PHP de MySQL”.

Capítulo 26. Manejo de errores en MySQL

Este capítulo lista los errores que pueden aparecer cuando llama MySQL desde cualquier idioma de equipo. La primera lista muestra los mensajes de error del servidor. La segunda lista muestra los mensajes de los programas clientes.

La información de error del servidor se obtiene de los siguientes archivos:

- Los valores de error y los símbolos en paréntesis se corresponden con las definiciones en el fichero fuente de MySQL `include/mysqld_error.h`.
- Los valores SQLSTATE se corresponden con las definiciones en el fichero fuente de MySQL `include/sql_state.h`.
- Los valores de Mensaje se corresponden con los mensajes de error que pueden encontrarse listados en el fichero `share/errmsg.txt` . `%d` y `%s` representan números y cadenas de caracteres, respectivamente, que se substituyen en los mensajes cuando se muestran.

Como las actualizaciones son frecuentes, es posible que estos ficheros contengan información de error adicional que no está listada aquí.

- Error: `1000` SQLSTATE: `HY000` (`ER_HASHCHK`)

  Mensaje: hashchk

- Error: `1001` SQLSTATE: `HY000` (`ER_NISAMCHK`)

  Mensaje: isamchk

- Error: `1002` SQLSTATE: `HY000` (`ER_NO`)

  Mensaje: NO

- Error: `1003` SQLSTATE: `HY000` (`ER_YES`)

  Mensaje: SI

- Error: `1004` SQLSTATE: `HY000` (`ER_CANT_CREATE_FILE`)

  Mensaje: No puedo crear archivo '%s' (Error: %d)

- Error: `1005` SQLSTATE: `HY000` (`ER_CANT_CREATE_TABLE`)

  Mensaje: No puedo crear tabla '%s' (Error: %d)

- Error: `1006` SQLSTATE: `HY000` (`ER_CANT_CREATE_DB`)

  Mensaje: No puedo crear base de datos '%s' (Error: %d)

- Error: `1007` SQLSTATE: `HY000` (`ER_DB_CREATE_EXISTS`)

  Mensaje: No puedo crear base de datos '%s'; la base de datos ya existe

- Error: `1008` SQLSTATE: `HY000` (`ER_DB_DROP_EXISTS`)

  Mensaje: No puedo eliminar base de datos '%s'; la base de datos no existe

- Error: `1009` SQLSTATE: `HY000` (`ER_DB_DROP_DELETE`)

  Mensaje: Error eliminando la base de datos(no puedo borrar '%s', error %d)

- Error: 1010 SQLSTATE: HY000 (ER_DB_DROP_RMDIR)

  Mensaje: Error eliminando la base de datos (No puedo borrar directorio '%s', error %d)

- Error: 1011 SQLSTATE: HY000 (ER_CANT_DELETE_FILE)

  Mensaje: Error en el borrado de '%s' (Error: %d)

- Error: 1012 SQLSTATE: HY000 (ER_CANT_FIND_SYSTEM_REC)

  Mensaje: No puedo leer el registro en la tabla del sistema

- Error: 1013 SQLSTATE: HY000 (ER_CANT_GET_STAT)

  Mensaje: No puedo obtener el estado de '%s' (Error: %d)

- Error: 1014 SQLSTATE: HY000 (ER_CANT_GET_WD)

  Mensaje: No puedo acceder al directorio (Error: %d)

- Error: 1015 SQLSTATE: HY000 (ER_CANT_LOCK)

  Mensaje: No puedo bloquear archivo: (Error: %d)

- Error: 1016 SQLSTATE: HY000 (ER_CANT_OPEN_FILE)

  Mensaje: No puedo abrir archivo: '%s' (Error: %d)

- Error: 1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND)

  Mensaje: No puedo encontrar archivo: '%s' (Error: %d)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

  Mensaje: No puedo leer el directorio de '%s' (Error: %d)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

  Mensaje: No puedo cambiar al directorio de '%s' (Error: %d)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD)

  Mensaje: El registro ha cambiado desde la ultima lectura de la tabla '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL)

  Mensaje: Disco lleno (%s). Esperando para que se libere algo de espacio...

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY)

  Mensaje: No puedo escribir, clave duplicada en la tabla '%s'

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

  Mensaje: Error en el cierre de '%s' (Error: %d)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

  Mensaje: Error leyendo el fichero '%s' (Error: %d)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

  Mensaje: Error en el renombrado de '%s' a '%s' (Error: %d)

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)

  Mensaje: Error escribiendo el archivo '%s' (Error: %d)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED)

  Mensaje: '%s' esta bloqueado contra cambios

- Error: 1028 SQLSTATE: HY000 (ER_FILSORT_ABORT)

  Mensaje: Ordeancion cancelada

- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND)

  Mensaje: La vista '%s' no existe para '%s'

- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO)

  Mensaje: Error %d desde el manejador de la tabla

- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL_HA)

  Mensaje: El manejador de la tabla de '%s' no tiene esta opcion

- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND)

  Mensaje: No puedo encontrar el registro en '%s'

- Error: 1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE)

  Mensaje: Informacion erronea en el archivo: '%s'

- Error: 1034 SQLSTATE: HY000 (ER_NOT_KEYFILE)

  Mensaje: Clave de archivo erronea para la tabla: '%s'; intente repararlo

- Error: 1035 SQLSTATE: HY000 (ER_OLD_KEYFILE)

  Mensaje: Clave de archivo antigua para la tabla '%s'; reparelo!

- Error: 1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY)

  Mensaje: '%s' es de solo lectura

- Error: 1037 SQLSTATE: HY001 (ER_OUTOFMEMORY)

  Mensaje: Memoria insuficiente. Reinicie el demonio e intentelo otra vez (necesita %d bytes)

- Error: 1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY)

  Mensaje: Memoria de ordenacion insuficiente. Incremente el tamano del buffer de ordenacion

- Error: 1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF)

  Mensaje: Inesperado fin de ficheroU mientras leiamos el archivo '%s' (Error: %d)

- Error: 1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR)

  Mensaje: Demasiadas conexiones
- Error: 1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES)

  Mensaje: Memoria/espacio de tranpaso insuficiente
- Error: 1042 SQLSTATE: 08S01 (ER_BAD_HOST_ERROR)

  Mensaje: No puedo obtener el nombre de maquina de tu direccion
- Error: 1043 SQLSTATE: 08S01 (ER_HANDSHAKE_ERROR)

  Mensaje: Protocolo erroneo
- Error: 1044 SQLSTATE: 42000 (ER_DBACCESS_DENIED_ERROR)

  Mensaje: Acceso negado para usuario: '%s'@'%s' para la base de datos '%s'
- Error: 1045 SQLSTATE: 28000 (ER_ACCESS_DENIED_ERROR)

  Mensaje: Acceso negado para usuario: '%s'@'%s' (Usando clave: %s)
- Error: 1046 SQLSTATE: 3D000 (ER_NO_DB_ERROR)

  Mensaje: Base de datos no seleccionada
- Error: 1047 SQLSTATE: 08S01 (ER_UNKNOWN_COM_ERROR)

  Mensaje: Comando desconocido
- Error: 1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR)

  Mensaje: La columna '%s' no puede ser nula
- Error: 1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR)

  Mensaje: Base de datos desconocida '%s'
- Error: 1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR)

  Mensaje: La tabla '%s' ya existe
- Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)

  Mensaje: Tabla '%s' desconocida
- Error: 1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR)

  Mensaje: La columna: '%s' en %s es ambigua
- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

  Mensaje: Desconexion de servidor en proceso
- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

  Mensaje: La columna '%s' en %s es desconocida

- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

  Mensaje: Usado '%s' el cual no esta group by

- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD)

  Mensaje: No puedo agrupar por '%s'

- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT)

  Mensaje: El estamento tiene funciones de suma y columnas en el mismo estamento

- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT)

  Mensaje: La columna con count no tiene valores para contar

- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT)

  Mensaje: El nombre del identificador '%s' es demasiado grande

- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME)

  Mensaje: Nombre de columna duplicado '%s'

- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME)

  Mensaje: Nombre de clave duplicado '%s'

- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY)

  Mensaje: Entrada duplicada '%s' para la clave %d

- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC)

  Mensaje: Especificador de columna erroneo para la columna '%s'

- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR)

  Mensaje: %s cerca '%s' en la linea %d

- Error: 1065 SQLSTATE: 42000 (ER_EMPTY_QUERY)

  Mensaje: La query estaba vacia

- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE)

  Mensaje: Tabla/alias: '%s' es no unica

- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT)

  Mensaje: Valor por defecto invalido para '%s'

- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY)

  Mensaje: Multiples claves primarias definidas

- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS)

  Mensaje: Demasiadas claves primarias declaradas. Un maximo de %d claves son permitidas

- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS)

  Mensaje: Demasiadas partes de clave declaradas. Un maximo de %d partes son permitidas

- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

  Mensaje: Declaracion de clave demasiado larga. La maxima longitud de clave es %d

- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS)

  Mensaje: La columna clave '%s' no existe en la tabla

- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

  Mensaje: La columna Blob '%s' no puede ser usada en una declaracion de clave

- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

  Mensaje: Longitud de columna demasiado grande para la columna '%s' (maximo = %lu).Usar BLOB en su lugar

- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

  Mensaje: Puede ser solamente un campo automatico y este debe ser definido como una clave

- Error: 1076 SQLSTATE: HY000 (ER_READY)

  Mensaje: %s: preparado para conexiones Version: '%s' socket: '%s' port: %d

- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)

  Mensaje: %s: Apagado normal

- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)

  Mensaje: %s: Recibiendo signal %d. Abortando!

- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)

  Mensaje: %s: Apagado completado

- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)

  Mensaje: %s: Forzando a cerrar el thread %ld usuario: '%s'

- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)

  Mensaje: No puedo crear IP socket

- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)

  Mensaje: La tabla '%s' no tiene indice como el usado en CREATE INDEX. Crea de nuevo la tabla

- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)

  Mensaje: Los separadores de argumentos del campo no son los especificados. Comprueba el manual

- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)

  Mensaje: No puedes usar longitudes de filas fijos con BLOBs. Por favor usa 'campos terminados por '.

- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)

  Mensaje: El archivo '%s' debe estar en el directorio de la base de datos o ser de lectura por todos

- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

  Mensaje: El archivo '%s' ya existe

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO)

  Mensaje: Registros: %ld Borrados: %ld Saltados: %ld Peligros: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO)

  Mensaje: Registros: %ld Duplicados: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

  Mensaje: Parte de la clave es erronea. Una parte de la clave no es una cadena o la longitud usada es tan grande como la parte de la clave

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

  Mensaje: No puede borrar todos los campos con ALTER TABLE. Usa DROP TABLE para hacerlo

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

  Mensaje: No puedo ELIMINAR '%s'. compuebe que el campo/clave existe

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO)

  Mensaje: Registros: %ld Duplicados: %ld Peligros: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

  Mensaje: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

  Mensaje: Identificador del thread: %lu desconocido

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

  Mensaje: Tu no eres el propietario del thread%lu

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)

  Mensaje: No ha tablas usadas

- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)

  Mensaje: Muchas strings para columna %s y SET

- Error: 1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE)

  Mensaje: No puede crear un unico archivo log %s.(1-999)

- Error: 1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE)

  Mensaje: Tabla '%s' fue trabada con un READ lock y no puede ser actualizada

- Error: 1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED)

  Mensaje: Tabla '%s' no fue trabada con LOCK TABLES

- Error: 1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT)

  Mensaje: Campo Blob '%s' no puede tener valores patron

- Error: 1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME)

  Mensaje: Nombre de base de datos ilegal '%s'

- Error: 1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME)

  Mensaje: Nombre de tabla ilegal '%s'

- Error: 1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT)

  Mensaje: El SELECT puede examinar muchos registros y probablemente con mucho tiempo. Verifique tu WHERE y usa SET SQL_BIG_SELECTS=1 si el SELECT esta correcto

- Error: 1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR)

  Mensaje: Error desconocido

- Error: 1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE)

  Mensaje: Procedimiento desconocido %s

- Error: 1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE)

  Mensaje: Equivocado parametro count para procedimiento %s

- Error: 1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE)

  Mensaje: Equivocados parametros para procedimiento %s

- Error: 1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE)

  Mensaje: Tabla desconocida '%s' in %s

- Error: 1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE)

  Mensaje: Campo '%s' especificado dos veces

- Error: 1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE)

  Mensaje: Invalido uso de función en grupo

- Error: 1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION)

  Mensaje: Tabla '%s' usa una extensión que no existe en esta MySQL versión

- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)

  Mensaje: Una tabla debe tener al menos 1 columna

- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)

  Mensaje: La tabla '%s' está llena

- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)

  Mensaje: Juego de caracteres desconocido: '%s'

- Error: 1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES)

  Mensaje: Muchas tablas. MySQL solamente puede usar %d tablas en un join

- Error: 1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS)

  Mensaje: Muchos campos

- Error: 1118 SQLSTATE: 42000 (ER_TOO_BIG_ROWSIZE)

  Mensaje: Tamaño de línea muy grande. Máximo tamaño de línea, no contando blob, es %ld. Tu tienes que cambiar algunos campos para blob

- Error: 1119 SQLSTATE: HY000 (ER_STACK_OVERRUN)

  Mensaje: Sobrecarga de la pila de thread: Usada: %ld de una %ld pila. Use 'mysqld -O thread_stack=#' para especificar una mayor pila si necesario

- Error: 1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN)

  Mensaje: Dependencia cruzada encontrada en OUTER JOIN. Examine su condición ON

- Error: 1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX)

  Mensaje: Columna '%s' es usada con UNIQUE o INDEX pero no está definida como NOT NULL

- Error: 1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF)

  Mensaje: No puedo cargar función '%s'

- Error: 1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF)

  Mensaje: No puedo inicializar función '%s'; %s

- Error: 1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS)

  Mensaje: No pasos permitidos para librarias conjugadas

- Error: 1125 SQLSTATE: HY000 (ER_UDF_EXISTS)

  Mensaje: Función '%s' ya existe

- Error: 1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY)

  Mensaje: No puedo abrir libraria conjugada '%s' (errno: %d %s)

- Error: 1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY)

  Mensaje: No puedo encontrar función '%s' en libraria

- Error: 1128 SQLSTATE: HY000 (ER_FUNCTION_NOT_DEFINED)

  Mensaje: Función '%s' no está definida

- Error: 1129 SQLSTATE: HY000 (ER_HOST_IS_BLOCKED)

Mensaje: Servidor '%s' está bloqueado por muchos errores de conexión. Desbloquear con 'mysqladmin flush-hosts'

- Error: `1130` SQLSTATE: `HY000` (`ER_HOST_NOT_PRIVILEGED`)

  Mensaje: Servidor '%s' no está permitido para conectar con este servidor MySQL

- Error: `1131` SQLSTATE: `42000` (`ER_PASSWORD_ANONYMOUS_USER`)

  Mensaje: Tu estás usando MySQL como un usuario anonimo y usuarios anonimos no tienen permiso para cambiar las claves

- Error: `1132` SQLSTATE: `42000` (`ER_PASSWORD_NOT_ALLOWED`)

  Mensaje: Tu debes de tener permiso para actualizar tablas en la base de datos mysql para cambiar las claves para otros

- Error: `1133` SQLSTATE: `42000` (`ER_PASSWORD_NO_MATCH`)

  Mensaje: No puedo encontrar una línea correponsdiente en la tabla user

- Error: `1134` SQLSTATE: `HY000` (`ER_UPDATE_INFO`)

  Mensaje: Líneas correspondientes: %ld Cambiadas: %ld Avisos: %ld

- Error: `1135` SQLSTATE: `HY000` (`ER_CANT_CREATE_THREAD`)

  Mensaje: No puedo crear un nuevo thread (errno %d). Si tu está con falta de memoria disponible, tu puedes consultar el Manual para posibles problemas con SO

- Error: `1136` SQLSTATE: `21S01` (`ER_WRONG_VALUE_COUNT_ON_ROW`)

  Mensaje: El número de columnas no corresponde al número en la línea %ld

- Error: `1137` SQLSTATE: `HY000` (`ER_CANT_REOPEN_TABLE`)

  Mensaje: No puedo reabrir tabla: '%s

- Error: `1138` SQLSTATE: `22004` (`ER_INVALID_USE_OF_NULL`)

  Mensaje: Invalido uso de valor NULL

- Error: `1139` SQLSTATE: `42000` (`ER_REGEXP_ERROR`)

  Mensaje: Obtenido error '%s' de regexp

- Error: `1140` SQLSTATE: `42000` (`ER_MIX_OF_GROUP_FUNC_AND_FIELDS`)

  Mensaje: Mezcla de columnas GROUP (MIN(),MAX(),COUNT()...) con no GROUP columnas es ilegal si no hat la clausula GROUP BY

- Error: `1141` SQLSTATE: `42000` (`ER_NONEXISTING_GRANT`)

  Mensaje: No existe permiso definido para usuario '%s' en el servidor '%s'

- Error: `1142` SQLSTATE: `42000` (`ER_TABLEACCESS_DENIED_ERROR`)

  Mensaje: %s comando negado para usuario: '%s'@'%s' para tabla '%s'

- Error: 1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR)

  Mensaje: %s comando negado para usuario: '%s'@'%s' para columna '%s' en la tabla '%s'

- Error: 1144 SQLSTATE: 42000 (ER_ILLEGAL_GRANT_FOR_TABLE)

  Mensaje: Ilegal comando GRANT/REVOKE. Por favor consulte el manual para cuales permisos pueden ser usados.

- Error: 1145 SQLSTATE: 42000 (ER_GRANT_WRONG_HOST_OR_USER)

  Mensaje: El argumento para servidor o usuario para GRANT es demasiado grande

- Error: 1146 SQLSTATE: 42S02 (ER_NO_SUCH_TABLE)

  Mensaje: Tabla '%s.%s' no existe

- Error: 1147 SQLSTATE: 42000 (ER_NONEXISTING_TABLE_GRANT)

  Mensaje: No existe tal permiso definido para usuario '%s' en el servidor '%s' en la tabla '%s'

- Error: 1148 SQLSTATE: 42000 (ER_NOT_ALLOWED_COMMAND)

  Mensaje: El comando usado no es permitido con esta versión de MySQL

- Error: 1149 SQLSTATE: 42000 (ER_SYNTAX_ERROR)

  Mensaje: Algo está equivocado en su sintax

- Error: 1150 SQLSTATE: HY000 (ER_DELAYED_CANT_CHANGE_LOCK)

  Mensaje: Thread de inserción retarda no pudiendo bloquear para la tabla %s

- Error: 1151 SQLSTATE: HY000 (ER_TOO_MANY_DELAYED_THREADS)

  Mensaje: Muchos threads retardados en uso

- Error: 1152 SQLSTATE: 08S01 (ER_ABORTING_CONNECTION)

  Mensaje: Conexión abortada %ld para db: '%s' usuario: '%s' (%s)

- Error: 1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE)

  Mensaje: Obtenido un paquete mayor que 'max_allowed_packet'

- Error: 1154 SQLSTATE: 08S01 (ER_NET_READ_ERROR_FROM_PIPE)

  Mensaje: Obtenido un error de lectura de la conexión pipe

- Error: 1155 SQLSTATE: 08S01 (ER_NET_FCNTL_ERROR)

  Mensaje: Obtenido un error de fcntl()

- Error: 1156 SQLSTATE: 08S01 (ER_NET_PACKETS_OUT_OF_ORDER)

  Mensaje: Obtenido paquetes desordenados

- Error: 1157 SQLSTATE: 08S01 (ER_NET_UNCOMPRESS_ERROR)

  Mensaje: No puedo descomprimir paquetes de comunicación

- Error: 1158 SQLSTATE: 08S01 (ER_NET_READ_ERROR)

  Mensaje: Obtenido un error leyendo paquetes de comunicación

- Error: 1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED)

  Mensaje: Obtenido timeout leyendo paquetes de comunicación

- Error: 1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE)

  Mensaje: Obtenido un error de escribiendo paquetes de comunicación

- Error: 1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED)

  Mensaje: Obtenido timeout escribiendo paquetes de comunicación

- Error: 1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING)

  Mensaje: La string resultante es mayor que max_allowed_packet

- Error: 1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB)

  Mensaje: El tipo de tabla usada no permite soporte para columnas BLOB/TEXT

- Error: 1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT)

  Mensaje: El tipo de tabla usada no permite soporte para columnas AUTO_INCREMENT

- Error: 1165 SQLSTATE: HY000 (ER_DELAYED_INSERT_TABLE_LOCKED)

  Mensaje: INSERT DELAYED no puede ser usado con tablas '%s', porque esta bloqueada con LOCK TABLES

- Error: 1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME)

  Mensaje: Incorrecto nombre de columna '%s'

- Error: 1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN)

  Mensaje: El manipulador de tabla usado no puede indexar columna '%s'

- Error: 1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE)

  Mensaje: Todas las tablas en la MERGE tabla no estan definidas identicamente

- Error: 1169 SQLSTATE: 23000 (ER_DUP_UNIQUE)

  Mensaje: No puedo escribir, debido al único constraint, para tabla '%s'

- Error: 1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH)

  Mensaje: Columna BLOB column '%s' usada en especificación de clave sin tamaño de la clave

- Error: 1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL)

  Mensaje: Todas las partes de un PRIMARY KEY deben ser NOT NULL; Si necesitas NULL en una clave, use UNIQUE

- Error: 1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS)

Mensaje: Resultado compuesto de mas que una línea

- Error: 1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY)

  Mensaje: Este tipo de tabla necesita de una primary key

- Error: 1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED)

  Mensaje: Esta versión de MySQL no es compilada con soporte RAID

- Error: 1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE)

  Mensaje: Tu estás usando modo de actualización segura y tentado actualizar una tabla sin un WHERE que usa una KEY columna

- Error: 1176 SQLSTATE: HY000 (ER_KEY_DOES_NOT_EXITS)

  Mensaje: Clave '%s' no existe en la tabla '%s'

- Error: 1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE)

  Mensaje: No puedo abrir tabla

- Error: 1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED)

  Mensaje: El manipulador de la tabla no permite soporte para %s

- Error: 1179 SQLSTATE: 25000 (ER_CANT_DO_THIS_DURING_AN_TRANSACTION)

  Mensaje: No tienes el permiso para ejecutar este comando en una transición

- Error: 1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT)

  Mensaje: Obtenido error %d durante COMMIT

- Error: 1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK)

  Mensaje: Obtenido error %d durante ROLLBACK

- Error: 1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS)

  Mensaje: Obtenido error %d durante FLUSH_LOGS

- Error: 1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT)

  Mensaje: Obtenido error %d durante CHECKPOINT

- Error: 1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION)

  Mensaje: Abortada conexión %ld para db: '%s' usuario: '%s' servidor: '%s' (%s)

- Error: 1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED)

  Mensaje: El manipulador de tabla no soporta dump para tabla binaria

- Error: 1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED)

  Mensaje: Binlog closed, cannot RESET MASTER

- Error: 1187 SQLSTATE: HY000 (ER_INDEX_REBUILD)

  Mensaje: Falla reconstruyendo el indice de la tabla dumped '%s'

- Error: 1188 SQLSTATE: HY000 (ER_MASTER)

  Mensaje: Error del master: '%s'

- Error: 1189 SQLSTATE: 08S01 (ER_MASTER_NET_READ)

  Mensaje: Error de red leyendo del master

- Error: 1190 SQLSTATE: 08S01 (ER_MASTER_NET_WRITE)

  Mensaje: Error de red escribiendo para el master

- Error: 1191 SQLSTATE: HY000 (ER_FT_MATCHING_KEY_NOT_FOUND)

  Mensaje: No puedo encontrar índice FULLTEXT correspondiendo a la lista de columnas

- Error: 1192 SQLSTATE: HY000 (ER_LOCK_OR_ACTIVE_TRANSACTION)

  Mensaje: No puedo ejecutar el comando dado porque tienes tablas bloqueadas o una transición activa

- Error: 1193 SQLSTATE: HY000 (ER_UNKNOWN_SYSTEM_VARIABLE)

  Mensaje: Desconocida variable de sistema '%s'

- Error: 1194 SQLSTATE: HY000 (ER_CRASHED_ON_USAGE)

  Mensaje: Tabla '%s' está marcada como crashed y debe ser reparada

- Error: 1195 SQLSTATE: HY000 (ER_CRASHED_ON_REPAIR)

  Mensaje: Tabla '%s' está marcada como crashed y la última reparación (automactica?) falló

- Error: 1196 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK)

  Mensaje: Aviso: Algunas tablas no transancionales no pueden tener rolled back

- Error: 1197 SQLSTATE: HY000 (ER_TRANS_CACHE_FULL)

  Mensaje: Multipla transición necesita mas que 'max_binlog_cache_size' bytes de almacenamiento. Aumente esta variable mysqld y tente de nuevo

- Error: 1198 SQLSTATE: HY000 (ER_SLAVE_MUST_STOP)

  Mensaje: Esta operación no puede ser hecha con el esclavo funcionando, primero use STOP SLAVE

- Error: 1199 SQLSTATE: HY000 (ER_SLAVE_NOT_RUNNING)

  Mensaje: Esta operación necesita el esclavo funcionando, configure esclavo y haga el START SLAVE

- Error: 1200 SQLSTATE: HY000 (ER_BAD_SLAVE)

  Mensaje: El servidor no está configurado como esclavo, edite el archivo config file o con CHANGE MASTER TO

- Error: 1201 SQLSTATE: HY000 (ER_MASTER_INFO)

Mensaje: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: `1202` SQLSTATE: `HY000` (`ER_SLAVE_THREAD`)

  Mensaje: No puedo crear el thread esclavo, verifique recursos del sistema

- Error: `1203` SQLSTATE: `42000` (`ER_TOO_MANY_USER_CONNECTIONS`)

  Mensaje: Usario %s ya tiene mas que 'max_user_connections' conexiones activas

- Error: `1204` SQLSTATE: `HY000` (`ER_SET_CONSTANTS_ONLY`)

  Mensaje: Tu solo debes usar expresiones constantes con SET

- Error: `1205` SQLSTATE: `HY000` (`ER_LOCK_WAIT_TIMEOUT`)

  Mensaje: Tiempo de bloqueo de espera excedido

- Error: `1206` SQLSTATE: `HY000` (`ER_LOCK_TABLE_FULL`)

  Mensaje: El número total de bloqueos excede el tamaño de bloqueo de la tabla

- Error: `1207` SQLSTATE: `25000` (`ER_READ_ONLY_TRANSACTION`)

  Mensaje: Bloqueos de actualización no pueden ser adqueridos durante una transición READ UNCOMMITTED

- Error: `1208` SQLSTATE: `HY000` (`ER_DROP_DB_WITH_READ_LOCK`)

  Mensaje: DROP DATABASE no permitido mientras un thread está ejerciendo un bloqueo de lectura global

- Error: `1209` SQLSTATE: `HY000` (`ER_CREATE_DB_WITH_READ_LOCK`)

  Mensaje: CREATE DATABASE no permitido mientras un thread está ejerciendo un bloqueo de lectura global

- Error: `1210` SQLSTATE: `HY000` (`ER_WRONG_ARGUMENTS`)

  Mensaje: Argumentos errados para %s

- Error: `1211` SQLSTATE: `42000` (`ER_NO_PERMISSION_TO_CREATE_USER`)

  Mensaje: '%s`@`%s` no es permitido para crear nuevos usuarios

- Error: `1212` SQLSTATE: `HY000` (`ER_UNION_TABLES_IN_DIFFERENT_DIR`)

  Mensaje: Incorrecta definición de la tabla; Todas las tablas MERGE deben estar en el mismo banco de datos

- Error: `1213` SQLSTATE: `40001` (`ER_LOCK_DEADLOCK`)

  Mensaje: Encontrado deadlock cuando tentando obtener el bloqueo; Tente recomenzar la transición

- Error: `1214` SQLSTATE: `HY000` (`ER_TABLE_CANT_HANDLE_FT`)

  Mensaje: El tipo de tabla usada no soporta índices FULLTEXT

- Error: 1215 SQLSTATE: HY000 (ER_CANNOT_ADD_FOREIGN)

  Mensaje: No puede adicionar clave extranjera constraint

- Error: 1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW)

  Mensaje: No puede adicionar una línea hijo: falla de clave extranjera constraint

- Error: 1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED)

  Mensaje: No puede deletar una línea padre: falla de clave extranjera constraint

- Error: 1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER)

  Mensaje: Error de coneccion a master: %s

- Error: 1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER)

  Mensaje: Error executando el query en master: %s

- Error: 1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND)

  Mensaje: Error de %s: %s

- Error: 1221 SQLSTATE: HY000 (ER_WRONG_USAGE)

  Mensaje: Equivocado uso de %s y %s

- Error: 1222 SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT)

  Mensaje: El comando SELECT usado tiene diferente número de columnas

- Error: 1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK)

  Mensaje: No puedo ejecutar el query porque usted tiene conflicto de traba de lectura

- Error: 1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED)

  Mensaje: Mezla de transancional y no-transancional tablas está deshabilitada

- Error: 1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT)

  Mensaje: Opción '%s' usada dos veces en el comando

- Error: 1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED)

  Mensaje: Usuario '%s' ha excedido el recurso '%s' (actual valor: %ld)

- Error: 1227 SQLSTATE: 42000 (ER_SPECIFIC_ACCESS_DENIED_ERROR)

  Mensaje: Acceso negado. Usted necesita el privilegio %s para esta operación

- Error: 1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE)

  Mensaje: Variable '%s' es una SESSION variable y no puede ser usada con SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

  Mensaje: Variable '%s' es una GLOBAL variable y no puede ser configurada con SET GLOBAL

- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

  Mensaje: Variable '%s' no tiene un valor patrón

- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

  Mensaje: Variable '%s' no puede ser configurada para el valor de '%s'

- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

  Mensaje: Tipo de argumento equivocado para variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

  Mensaje: Variable '%s' solamente puede ser configurada, no leída

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

  Mensaje: Equivocado uso/colocación de '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

  Mensaje: Esta versión de MySQL no soporta todavia '%s'

- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG)

  Mensaje: Recibió fatal error %d: '%s' del master cuando leyendo datos del binary log

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

  Mensaje: Slave SQL thread ignorado el query debido a las reglas de replicación-*-tabla

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

  Mensaje: Variable '%s' es una %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

  Mensaje: Equivocada definición de llave extranjera para '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

  Mensaje: Referencia de llave y referencia de tabla no coinciden

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)

  Mensaje: Operando debe tener %d columna(s)

- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)

  Mensaje: Subconsulta retorna mas que 1 línea

- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)

  Mensaje: Desconocido preparado comando handler (%.*s) dado para %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)

  Mensaje: Base de datos Help está corrupto o no existe

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)

  Mensaje: Cíclica referencia en subconsultas

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)

  Mensaje: Convirtiendo columna '%s' de %s para %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)

  Mensaje: Referencia '%s' no soportada (%s)

- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)

  Mensaje: Cada tabla derivada debe tener su propio alias

- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)

  Mensaje: Select %u fué reducido durante optimización

- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)

  Mensaje: Tabla '%s' de uno de los SELECT no puede ser usada en %s

- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

  Mensaje: Cliente no soporta protocolo de autenticación solicitado por el servidor; considere actualizar el cliente MySQL

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

  Mensaje: Todas las partes de una SPATIAL index deben ser NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

  Mensaje: COLLATION '%s' no es válido para CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

  Mensaje: Slave ya está funcionando

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

  Mensaje: Slave ya fué parado

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

  Mensaje: Tamaño demasiado grande para datos descomprimidos. El máximo tamaño es %d. (probablemente, extensión de datos descomprimidos fué corrompida)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

  Mensaje: Z_MEM_ERROR: No suficiente memoria para zlib

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

  Mensaje: Z_BUF_ERROR: No suficiente espacio en el búfer de salida para zlib (probablemente, extensión de datos descomprimidos fué corrompida)

- Error: 1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR)

Mensaje: ZLIB: Dato de entrada fué corrompido para zlib

- Error: 1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT)

  Mensaje: %d línea(s) fue(fueron) cortadas por group_concat()

- Error: 1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS)

  Mensaje: Línea %ld no contiene datos para todas las columnas

- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS)

  Mensaje: Línea %ld fué truncada; La misma contine mas datos que las que existen en las columnas de entrada

- Error: 1263 SQLSTATE: 22004 (ER_WARN_NULL_TO_NOTNULL)

  Mensaje: Datos truncado, NULL suministrado para NOT NULL columna '%s' en la línea %ld

- Error: 1264 SQLSTATE: 22003 (ER_WARN_DATA_OUT_OF_RANGE)

  Mensaje: Datos truncados, fuera de gama para columna '%s' en la línea %ld

- Error: 1265 SQLSTATE: 01000 (WARN_DATA_TRUNCATED)

  Mensaje: Datos truncados para columna '%s' en la línea %ld

- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER)

  Mensaje: Usando motor de almacenamiento %s para tabla '%s'

- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS)

  Mensaje: Ilegal mezcla de collations (%s,%s) y (%s,%s) para operación '%s'

- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER)

  Mensaje: Cannot drop one or more of the requested users

- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS)

  Mensaje: No puede revocar todos los privilegios, derecho para uno o mas de los usuarios solicitados

- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS)

  Mensaje: Ilegal mezcla de collations (%s,%s), (%s,%s), (%s,%s) para operación '%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS)

  Mensaje: Ilegal mezcla de collations para operación '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT)

  Mensaje: Variable '%s' no es una variable componente (No puede ser usada como XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

  Mensaje: Collation desconocida: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

  Mensaje: Parametros SSL en CHANGE MASTER son ignorados porque este slave MySQL fue compilado sin soporte SSL; pueden ser usados despues cuando el slave MySQL con SSL sea inicializado

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

  Mensaje: Servidor está rodando en modo --secure-auth, pero '%s'@'%s' tiene clave en el antiguo formato; por favor cambie la clave para el nuevo formato

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

  Mensaje: Campo o referencia '%s%s%s%s%s' de SELECT #%d fue resolvido en SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

  Mensaje: Parametro equivocado o combinación de parametros para START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

  Mensaje: Es recomendado rodar con --skip-slave-start cuando haciendo replicación step-by-step con START SLAVE UNTIL, a menos que usted no esté seguro en caso de inesperada reinicialización del mysqld slave

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)

  Mensaje: SQL thread no es inicializado tal que opciones UNTIL son ignoradas

- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)

  Mensaje: Nombre de índice incorrecto '%s'

- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)

  Mensaje: Nombre de catalog incorrecto '%s'

- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)

  Mensaje: Query cache fallada para configurar tamaño %lu, nuevo tamaño de query cache es %lu

- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)

  Mensaje: Columna '%s' no puede ser parte de FULLTEXT index

- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)

  Mensaje: Desconocida key cache '%s'

- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)

  Mensaje: MySQL esta inicializado en modo --skip-name-resolve. Usted necesita reinicializarlo sin esta opción para este derecho funcionar

- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)

  Mensaje: Desconocido motor de tabla '%s'

- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)

Mensaje: '%s' está desaprobado, use '%s' en su lugar

- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)

  Mensaje: La tabla destino %s del %s no es actualizable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

  Mensaje: El recurso '%s' fue deshabilitado; usted necesita construir MySQL con '%s' para tener eso funcionando

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

  Mensaje: El servidor MySQL está rodando con la opción %s tal que no puede ejecutar este comando

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

  Mensaje: Columna '%s' tiene valor doblado '%s' en %s

- Error: 1292 SQLSTATE: 22007 (ER_TRUNCATED_WRONG_VALUE)

  Mensaje: Equivocado truncado %s valor: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS)

  Mensaje: Incorrecta definición de tabla; Solamente debe haber una columna TIMESTAMP con CURRENT_TIMESTAMP en DEFAULT o ON UPDATE cláusula

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE)

  Mensaje: Inválido ON UPDATE cláusula para campo '%s'

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS)

  Mensaje: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG)

  Mensaje: Got error %d '%s' from %s

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG)

  Mensaje: Got temporary error %d '%s' from %s

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE)

  Mensaje: Unknown or incorrect time zone: '%s'

- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP)

  Mensaje: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING)

  Mensaje: Invalid %s character string: '%s'

- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED)

  Mensaje: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS)

  Mensaje: Conflicting declarations: '%s%s' and '%s%s'

- Error: 1303 SQLSTATE: 2F003 (ER_SP_NO_RECURSIVE_CREATE)

  Mensaje: Can't create a %s from within another stored routine

- Error: 1304 SQLSTATE: 42000 (ER_SP_ALREADY_EXISTS)

  Mensaje: %s %s already exists

- Error: 1305 SQLSTATE: 42000 (ER_SP_DOES_NOT_EXIST)

  Mensaje: %s %s does not exist

- Error: 1306 SQLSTATE: HY000 (ER_SP_DROP_FAILED)

  Mensaje: Failed to DROP %s %s

- Error: 1307 SQLSTATE: HY000 (ER_SP_STORE_FAILED)

  Mensaje: Failed to CREATE %s %s

- Error: 1308 SQLSTATE: 42000 (ER_SP_LILABEL_MISMATCH)

  Mensaje: %s with no matching label: %s

- Error: 1309 SQLSTATE: 42000 (ER_SP_LABEL_REDEFINE)

  Mensaje: Redefining label %s

- Error: 1310 SQLSTATE: 42000 (ER_SP_LABEL_MISMATCH)

  Mensaje: End-label %s without match

- Error: 1311 SQLSTATE: 01000 (ER_SP_UNINIT_VAR)

  Mensaje: Referring to uninitialized variable %s

- Error: 1312 SQLSTATE: 0A000 (ER_SP_BADSELECT)

  Mensaje: PROCEDURE %s can't return a result set in the given context

- Error: 1313 SQLSTATE: 42000 (ER_SP_BADRETURN)

  Mensaje: RETURN is only allowed in a FUNCTION

- Error: 1314 SQLSTATE: 0A000 (ER_SP_BADSTATEMENT)

  Mensaje: %s is not allowed in stored procedures

- Error: 1315 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_IGNORED)

  Mensaje: 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)

  Mensaje: 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)

  Mensaje: Query execution was interrupted
- Error: 1318 SQLSTATE: 42000 (ER_SP_WRONG_NO_OF_ARGS)

  Mensaje: Incorrect number of arguments for %s %s; expected %u, got %u
- Error: 1319 SQLSTATE: 42000 (ER_SP_COND_MISMATCH)

  Mensaje: Undefined CONDITION: %s
- Error: 1320 SQLSTATE: 42000 (ER_SP_NORETURN)

  Mensaje: No RETURN found in FUNCTION %s
- Error: 1321 SQLSTATE: 2F005 (ER_SP_NORETURNEND)

  Mensaje: FUNCTION %s ended without RETURN
- Error: 1322 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_QUERY)

  Mensaje: Cursor statement must be a SELECT
- Error: 1323 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_SELECT)

  Mensaje: Cursor SELECT must not have INTO
- Error: 1324 SQLSTATE: 42000 (ER_SP_CURSOR_MISMATCH)

  Mensaje: Undefined CURSOR: %s
- Error: 1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN)

  Mensaje: Cursor is already open
- Error: 1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN)

  Mensaje: Cursor is not open
- Error: 1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR)

  Mensaje: Undeclared variable: %s
- Error: 1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS)

  Mensaje: Incorrect number of FETCH variables
- Error: 1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)

  Mensaje: No data - zero rows fetched, selected, or processed
- Error: 1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM)

  Mensaje: Duplicate parameter: %s
- Error: 1331 SQLSTATE: 42000 (ER_SP_DUP_VAR)

  Mensaje: Duplicate variable: %s

- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

  Mensaje: Duplicate condition: %s

- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

  Mensaje: Duplicate cursor: %s

- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

  Mensaje: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

  Mensaje: Subselect value not supported

- Error: 1336 SQLSTATE: 0A000 (ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG)

  Mensaje: %s is not allowed in stored function or trigger

- Error: 1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

  Mensaje: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

  Mensaje: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

  Mensaje: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

  Mensaje: Configuration file '%s' is too big

- Error: 1341 SQLSTATE: HY000 (ER_FPARSER_BAD_HEADER)

  Mensaje: Malformed file type header in file '%s'

- Error: 1342 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_COMMENT)

  Mensaje: Unexpected end of file while parsing comment '%s'

- Error: 1343 SQLSTATE: HY000 (ER_FPARSER_ERROR_IN_PARAMETER)

  Mensaje: Error while parsing parameter '%s' (line: '%s')

- Error: 1344 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER)

  Mensaje: Unexpected end of file while skipping unknown parameter '%s'

- Error: 1345 SQLSTATE: HY000 (ER_VIEW_NO_EXPLAIN)

  Mensaje: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error: 1346 SQLSTATE: HY000 (ER_FRM_UNKNOWN_TYPE)

  Mensaje: File '%s' has unknown type '%s' in its header

- Error: 1347 SQLSTATE: HY000 (ER_WRONG_OBJECT)

  Mensaje: '%s.%s' is not %s

- Error: 1348 SQLSTATE: HY000 (ER_NONUPDATEABLE_COLUMN)

  Mensaje: Column '%s' is not updatable

- Error: 1349 SQLSTATE: HY000 (ER_VIEW_SELECT_DERIVED)

  Mensaje: View's SELECT contains a subquery in the FROM clause

- Error: 1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE)

  Mensaje: View's SELECT contains a '%s' clause

- Error: 1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE)

  Mensaje: View's SELECT contains a variable or parameter

- Error: 1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE)

  Mensaje: View's SELECT refers to a temporary table '%s'

- Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

  Mensaje: View's SELECT and view's field list have different column counts

- Error: 1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE)

  Mensaje: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error: 1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY)

  Mensaje: View being updated does not have complete key of underlying table in it

- Error: 1356 SQLSTATE: HY000 (ER_VIEW_INVALID)

  Mensaje: 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)

  Mensaje: Can't drop or alter a %s from within another stored routine

- Error: 1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR)

  Mensaje: GOTO is not allowed in a stored procedure handler

- Error: 1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS)

  Mensaje: Trigger already exists

- Error: 1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST)

  Mensaje: Trigger does not exist

- Error: 1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE)

  Mensaje: Trigger's '%s' is view or temporary table

- Error: 1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW)

  Mensaje: Updating of %s row is not allowed in %strigger

- Error: 1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG)

  Mensaje: There is no %s row in %s trigger

- Error: 1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD)

  Mensaje: Field '%s' doesn't have a default value

- Error: 1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO)

  Mensaje: Division by 0

- Error: 1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD)

  Mensaje: Incorrect %s value: '%s' for column '%s' at row %ld

- Error: 1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE)

  Mensaje: Illegal %s '%s' value found during parsing

- Error: 1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK)

  Mensaje: CHECK OPTION on non-updatable view '%s.%s'

- Error: 1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED)

  Mensaje: CHECK OPTION failed '%s.%s'

- Error: 1370 SQLSTATE: 42000 (ER_PROCACCESS_DENIED_ERROR)

  Mensaje: %s command denied to user '%s'@'%s' for routine '%s'

- Error: 1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL)

  Mensaje: Failed purging old relay logs: %s

- Error: 1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH)

  Mensaje: Password hash should be a %d-digit hexadecimal number

- Error: 1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG)

  Mensaje: Target log not found in binlog index

- Error: 1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ)

  Mensaje: I/O error reading log index file

- Error: 1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED)

  Mensaje: Server configuration does not permit binlog purge

- Error: 1376 SQLSTATE: HY000 (ER_FSEEK_FAIL)

  Mensaje: Failed on fseek()

- Error: 1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR)

  Mensaje: Fatal error during log purge
- Error: 1378 SQLSTATE: HY000 (ER_LOG_IN_USE)

  Mensaje: A purgeable log is in use, will not purge
- Error: 1379 SQLSTATE: HY000 (ER_LOG_PURGE_UNKNOWN_ERR)

  Mensaje: Unknown error during log purge
- Error: 1380 SQLSTATE: HY000 (ER_RELAY_LOG_INIT)

  Mensaje: Failed initializing relay log position: %s
- Error: 1381 SQLSTATE: HY000 (ER_NO_BINARY_LOGGING)

  Mensaje: You are not using binary logging
- Error: 1382 SQLSTATE: HY000 (ER_RESERVED_SYNTAX)

  Mensaje: The '%s' syntax is reserved for purposes internal to the MySQL server
- Error: 1383 SQLSTATE: HY000 (ER_WSAS_FAILED)

  Mensaje: WSAStartup Failed
- Error: 1384 SQLSTATE: HY000 (ER_DIFF_GROUPS_PROC)

  Mensaje: Can't handle procedures with different groups yet
- Error: 1385 SQLSTATE: HY000 (ER_NO_GROUP_FOR_PROC)

  Mensaje: Select must have a group with this procedure
- Error: 1386 SQLSTATE: HY000 (ER_ORDER_WITH_PROC)

  Mensaje: Can't use ORDER clause with this procedure
- Error: 1387 SQLSTATE: HY000 (ER_LOGGING_PROHIBIT_CHANGING_OF)

  Mensaje: Binary logging and replication forbid changing the global server %s
- Error: 1388 SQLSTATE: HY000 (ER_NO_FILE_MAPPING)

  Mensaje: Can't map file: %s, errno: %d
- Error: 1389 SQLSTATE: HY000 (ER_WRONG_MAGIC)

  Mensaje: Wrong magic in %s
- Error: 1390 SQLSTATE: HY000 (ER_PS_MANY_PARAM)

  Mensaje: Prepared statement contains too many placeholders
- Error: 1391 SQLSTATE: HY000 (ER_KEY_PART_0)

  Mensaje: Key part '%s' length cannot be 0
- Error: 1392 SQLSTATE: HY000 (ER_VIEW_CHECKSUM)

Mensaje: View text checksum failed

- Error: 1393 SQLSTATE: HY000 (ER_VIEW_MULTIUPDATE)

  Mensaje: Can not modify more than one base table through a join view '%s.%s'

- Error: 1394 SQLSTATE: HY000 (ER_VIEW_NO_INSERT_FIELD_LIST)

  Mensaje: Can not insert into join view '%s.%s' without fields list

- Error: 1395 SQLSTATE: HY000 (ER_VIEW_DELETE_MERGE_VIEW)

  Mensaje: Can not delete from join view '%s.%s'

- Error: 1396 SQLSTATE: HY000 (ER_CANNOT_USER)

  Mensaje: Operation %s failed for %s

- Error: 1397 SQLSTATE: XAE04 (ER_XAER_NOTA)

  Mensaje: XAER_NOTA: Unknown XID

- Error: 1398 SQLSTATE: XAE05 (ER_XAER_INVAL)

  Mensaje: XAER_INVAL: Invalid arguments (or unsupported command)

- Error: 1399 SQLSTATE: XAE07 (ER_XAER_RMFAIL)

  Mensaje: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state

- Error: 1400 SQLSTATE: XAE09 (ER_XAER_OUTSIDE)

  Mensaje: XAER_OUTSIDE: Some work is done outside global transaction

- Error: 1401 SQLSTATE: XAE03 (ER_XAER_RMERR)

  Mensaje: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency

- Error: 1402 SQLSTATE: XA100 (ER_XA_RBROLLBACK)

  Mensaje: XA_RBROLLBACK: Transaction branch was rolled back

- Error: 1403 SQLSTATE: 42000 (ER_NONEXISTING_PROC_GRANT)

  Mensaje: There is no such grant defined for user '%s' on host '%s' on routine '%s'

- Error: 1404 SQLSTATE: HY000 (ER_PROC_AUTO_GRANT_FAIL)

  Mensaje: Failed to grant EXECUTE and ALTER ROUTINE privileges

- Error: 1405 SQLSTATE: HY000 (ER_PROC_AUTO_REVOKE_FAIL)

  Mensaje: Failed to revoke all privileges to dropped routine

- Error: 1406 SQLSTATE: 22001 (ER_DATA_TOO_LONG)

  Mensaje: Data too long for column '%s' at row %ld

- Error: 1407 SQLSTATE: 42000 (ER_SP_BAD_SQLSTATE)

Mensaje: Bad SQLSTATE: '%s'

- Error: 1408 SQLSTATE: HY000 (ER_STARTUP)

  Mensaje: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s

- Error: 1409 SQLSTATE: HY000 (ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR)

  Mensaje: Can't load value from file with fixed size rows to variable

- Error: 1410 SQLSTATE: 42000 (ER_CANT_CREATE_USER_WITH_GRANT)

  Mensaje: You are not allowed to create a user with GRANT

- Error: 1411 SQLSTATE: HY000 (ER_WRONG_VALUE_FOR_TYPE)

  Mensaje: Incorrect %s value: '%s' for function %s

- Error: 1412 SQLSTATE: HY000 (ER_TABLE_DEF_CHANGED)

  Mensaje: Table definition has changed, please retry transaction

- Error: 1413 SQLSTATE: 42000 (ER_SP_DUP_HANDLER)

  Mensaje: Duplicate handler declared in the same block

- Error: 1414 SQLSTATE: 42000 (ER_SP_NOT_VAR_ARG)

  Mensaje: 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)

  Mensaje: Not allowed to return a result set from a %s

- Error: 1416 SQLSTATE: 22003 (ER_CANT_CREATE_GEOMETRY_OBJECT)

  Mensaje: Cannot get geometry object from data you send to the GEOMETRY field

- Error: 1417 SQLSTATE: HY000 (ER_FAILED_ROUTINE_BREAK_BINLOG)

  Mensaje: 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)

  Mensaje: 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)

- Error: 1419 SQLSTATE: HY000 (ER_BINLOG_CREATE_ROUTINE_NEED_SUPER)

  Mensaje: 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)

  Mensaje: 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)

  Mensaje: The statement (%lu) has no open cursor.

- Error: 1422 SQLSTATE: HY000 (ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG)

  Mensaje: Explicit or implicit commit is not allowed in stored function or trigger.

- Error: 1423 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_VIEW_FIELD)

  Mensaje: Field of view '%s.%s' underlying table doesn't have a default value

- Error: 1424 SQLSTATE: HY000 (ER_SP_NO_RECURSION)

  Mensaje: Recursive stored functions and triggers are not allowed.

- Error: 1425 SQLSTATE: 42000 (ER_TOO_BIG_SCALE)

  Mensaje: Too big scale %lu specified for column '%s'. Maximum is %d.

- Error: 1426 SQLSTATE: 42000 (ER_TOO_BIG_PRECISION)

  Mensaje: Too big precision %lu specified for column '%s'. Maximum is %lu.

- Error: 1427 SQLSTATE: 42000 (ER_M_BIGGER_THAN_D)

  Mensaje: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

- Error: 1428 SQLSTATE: HY000 (ER_WRONG_LOCK_OF_SYSTEM_TABLE)

  Mensaje: You can't combine write-locking of system '%s.%s' table with other tables

- Error: 1429 SQLSTATE: HY000 (ER_CONNECT_TO_FOREIGN_DATA_SOURCE)

  Mensaje: Unable to connect to foreign data source: %s

- Error: 1430 SQLSTATE: HY000 (ER_QUERY_ON_FOREIGN_DATA_SOURCE)

  Mensaje: There was a problem processing the query on the foreign data source. Data source error: %s

- Error: 1431 SQLSTATE: HY000 (ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST)

  Mensaje: The foreign data source you are trying to reference does not exist. Data source error: %s

- Error: 1432 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE)

  Mensaje: Can't create federated table. The data source connection string '%s' is not in the correct format

- Error: 1433 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID)

  Mensaje: The data source connection string '%s' is not in the correct format

- Error: 1434 SQLSTATE: HY000 (ER_CANT_CREATE_FEDERATED_TABLE)

  Mensaje: Can't create federated table. Foreign data src error: %s

- Error: 1435 SQLSTATE: HY000 (ER_TRG_IN_WRONG_SCHEMA)

  Mensaje: Trigger in wrong schema

- Error: 1436 SQLSTATE: HY000 (ER_STACK_OVERRUN_NEED_MORE)

  Mensaje: 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.

- Error: 1437 SQLSTATE: 42000 (ER_TOO_LONG_BODY)

  Mensaje: Routine body for '%s' is too long

- Error: 1438 SQLSTATE: HY000 (ER_WARN_CANT_DROP_DEFAULT_KEYCACHE)

  Mensaje: Cannot drop default keycache

- Error: 1439 SQLSTATE: 42000 (ER_TOO_BIG_DISPLAYWIDTH)

  Mensaje: Display width out of range for column '%s' (max = %lu)

- Error: 1440 SQLSTATE: XAE08 (ER_XAER_DUPID)

  Mensaje: XAER_DUPID: The XID already exists

- Error: 1441 SQLSTATE: 22008 (ER_DATETIME_FUNCTION_OVERFLOW)

  Mensaje: Datetime function: %s field overflow

- Error: 1442 SQLSTATE: HY000 (ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG)

  Mensaje: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.

- Error: 1443 SQLSTATE: HY000 (ER_VIEW_PREVENT_UPDATE)

  Mensaje: The definition of table '%s' prevents operation %s on table '%s'.

- Error: 1444 SQLSTATE: HY000 (ER_PS_NO_RECURSION)

  Mensaje: 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

- Error: 1445 SQLSTATE: HY000 (ER_SP_CANT_SET_AUTOCOMMIT)

  Mensaje: Not allowed to set autocommit from a stored function or trigger

- Error: 1446 SQLSTATE: HY000 (ER_MALFORMED_DEFINER)

  Mensaje: Definer is not fully qualified

- Error: 1447 SQLSTATE: HY000 (ER_VIEW_FRM_NO_USER)

  Mensaje: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!

- Error: 1448 SQLSTATE: HY000 (ER_VIEW_OTHER_USER)

  Mensaje: You need the SUPER privilege for creation view with '%s'@'%s' definer

- Error: 1449 SQLSTATE: HY000 (ER_NO_SUCH_USER)

  Mensaje: There is no '%s'@'%s' registered

- Error: 1450 SQLSTATE: HY000 (ER_FORBID_SCHEMA_CHANGE)

  Mensaje: Changing schema from '%s' to '%s' is not allowed.

- Error: 1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2)

  Mensaje: Cannot delete or update a parent row: a foreign key constraint fails (%s)

- Error: 1452 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW_2)

  Mensaje: Cannot add or update a child row: a foreign key constraint fails (%s)

- Error: 1453 SQLSTATE: 42000 (ER_SP_BAD_VAR_SHADOW)

  Mensaje: Variable '%s' must be quoted with `...`, or renamed

- Error: 1454 SQLSTATE: HY000 (ER_TRG_NO_DEFINER)

  Mensaje: 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.

- Error: 1455 SQLSTATE: HY000 (ER_OLD_FILE_FORMAT)

  Mensaje: '%s' has an old format, you should re-create the '%s' object(s)

- Error: 1456 SQLSTATE: HY000 (ER_SP_RECURSION_LIMIT)

  Mensaje: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

- Error: 1457 SQLSTATE: HY000 (ER_SP_PROC_TABLE_CORRUPT)

  Mensaje: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)

- Error: 1458 SQLSTATE: 42000 (ER_SP_WRONG_NAME)

  Mensaje: Incorrect routine name '%s'

- Error: 1459 SQLSTATE: HY000 (ER_TABLE_NEEDS_UPGRADE)

  Mensaje: Table upgrade required. Please do "REPAIR TABLE `%s`" to fix it!

- Error: 1460 SQLSTATE: 42000 (ER_SP_NO_AGGREGATE)

  Mensaje: AGGREGATE is not supported for stored functions

- Error: 1461 SQLSTATE: 42000 (ER_MAX_PREPARED_STMT_COUNT_REACHED)

  Mensaje: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error: 1462 SQLSTATE: HY000 (ER_VIEW_RECURSIVE)

  Mensaje: `%s`.`%s` contains view recursion

- Error: 1463 SQLSTATE: 42000 (ER_NON_GROUPING_FIELD_USED)

  Mensaje: non-grouping field '%s' is used in %s clause

- Error: 1464 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_SPKEYS)

Mensaje: The used table type doesn't support SPATIAL indexes

- Error: 1465 SQLSTATE: HY000 (ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA)

  Mensaje: Triggers can not be created on system tables

- Error: 1466 SQLSTATE: HY000 (ER_REMOVED_SPACES)

  Mensaje: Leading spaces are removed from name '%s'

- Error: 1467 SQLSTATE: HY000 (ER_AUTOINC_READ_FAILED)

  Mensaje: Failed to read auto-increment value from storage engine

- Error: 1468 SQLSTATE: HY000 (ER_USERNAME)

  Mensaje: user name

- Error: 1469 SQLSTATE: HY000 (ER_HOSTNAME)

  Mensaje: host name

- Error: 1470 SQLSTATE: HY000 (ER_WRONG_STRING_LENGTH)

  Mensaje: String '%s' is too long for %s (should be no longer than %d)

- Error: 1471 SQLSTATE: HY000 (ER_NON_INSERTABLE_TABLE)

  Mensaje: The target table %s of the %s is not insertable-into

- Error: 1472 SQLSTATE: HY000 (ER_ADMIN_WRONG_MRG_TABLE)

  Mensaje: Table '%s' is differently defined or of non-MyISAM type or doesn't exist

- Error: 1473 SQLSTATE: HY000 (ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT)

  Mensaje: Too high level of nesting for select

- Error: 1474 SQLSTATE: HY000 (ER_NAME_BECOMES_EMPTY)

  Mensaje: Name '%s' has become ''

- Error: 1475 SQLSTATE: HY000 (ER_AMBIGUOUS_FIELD_TERM)

  Mensaje: First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY

- Error: 1476 SQLSTATE: HY000 (ER_LOAD_DATA_INVALID_COLUMN)

  Mensaje: Invalid column reference (%s) in LOAD DATA

- Error: 1477 SQLSTATE: HY000 (ER_LOG_PURGE_NO_FILE)

  Mensaje: Being purged log %s was not found

- Error: 1478 SQLSTATE: XA106 (ER_XA_RBTIMEOUT)

  Mensaje: XA_RBTIMEOUT: Transaction branch was rolled back: took too long

- Error: 1479 SQLSTATE: XA102 (ER_XA_RBDEADLOCK)

  Mensaje: XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected

- Error: 1480 SQLSTATE: HY000 (ER_TOO_MANY_CONCURRENT_TRXS)

  Mensaje: Too many active concurrent transactions

La información de error de cliente proviene de los siguientes ficheros:

- Los valores de Error y los símbolos en paréntesis se corresponden a las definiciones en el fichero fuente MySQL include/errmsg.h .
- Los valores de Mensaje se corresponden con los mensajes de error que se listan en el fichero libmysql/errmsg.c . %d y %s representan números y cadenas de caracteres, respectivamente, que se substituyen en los mensajes cuando se muestran.

Como las actualizaciones son frecuentes, es posible que estos ficheros contengan información de error adicional que no está listada aquí.

- Error: 2000 (CR_UNKNOWN_ERROR)

  Mensaje: Unknown MySQL error

- Error: 2001 (CR_SOCKET_CREATE_ERROR)

  Mensaje: Can't create UNIX socket (%d)

- Error: 2002 (CR_CONNECTION_ERROR)

  Mensaje: Can't connect to local MySQL server through socket '%s' (%d)

- Error: 2003 (CR_CONN_HOST_ERROR)

  Mensaje: Can't connect to MySQL server on '%s' (%d)

- Error: 2004 (CR_IPSOCK_ERROR)

  Mensaje: Can't create TCP/IP socket (%d)

- Error: 2005 (CR_UNKNOWN_HOST)

  Mensaje: Unknown MySQL server host '%s' (%d)

- Error: 2006 (CR_SERVER_GONE_ERROR)

  Mensaje: MySQL server has gone away

- Error: 2007 (CR_VERSION_ERROR)

  Mensaje: Protocol mismatch; server version = %d, client version = %d

- Error: 2008 (CR_OUT_OF_MEMORY)

  Mensaje: MySQL client ran out of memory

- Error: 2009 (CR_WRONG_HOST_INFO)

  Mensaje: Wrong host info

- Error: `2010` (`CR_LOCALHOST_CONNECTION`)

  Mensaje: Localhost via UNIX socket

- Error: `2011` (`CR_TCP_CONNECTION`)

  Mensaje: %s via TCP/IP

- Error: `2012` (`CR_SERVER_HANDSHAKE_ERR`)

  Mensaje: Error in server handshake

- Error: `2013` (`CR_SERVER_LOST`)

  Mensaje: Lost connection to MySQL server during query

- Error: `2014` (`CR_COMMANDS_OUT_OF_SYNC`)

  Mensaje: Commands out of sync; you can't run this command now

- Error: `2015` (`CR_NAMEDPIPE_CONNECTION`)

  Mensaje: Named pipe: %s

- Error: `2016` (`CR_NAMEDPIPEWAIT_ERROR`)

  Mensaje: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error: `2017` (`CR_NAMEDPIPEOPEN_ERROR`)

  Mensaje: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: `2018` (`CR_NAMEDPIPESETSTATE_ERROR`)

  Mensaje: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error: `2019` (`CR_CANT_READ_CHARSET`)

  Mensaje: Can't initialize character set %s (path: %s)

- Error: `2020` (`CR_NET_PACKET_TOO_LARGE`)

  Mensaje: Got packet bigger than 'max_allowed_packet' bytes

- Error: `2021` (`CR_EMBEDDED_CONNECTION`)

  Mensaje: Embedded server

- Error: `2022` (`CR_PROBE_SLAVE_STATUS`)

  Mensaje: Error on SHOW SLAVE STATUS:

- Error: `2023` (`CR_PROBE_SLAVE_HOSTS`)

  Mensaje: Error on SHOW SLAVE HOSTS:

- Error: `2024` (`CR_PROBE_SLAVE_CONNECT`)

  Mensaje: Error connecting to slave:

- Error: 2025 (CR_PROBE_MASTER_CONNECT)

  Mensaje: Error connecting to master:

- Error: 2026 (CR_SSL_CONNECTION_ERROR)

  Mensaje: SSL connection error

- Error: 2027 (CR_MALFORMED_PACKET)

  Mensaje: Malformed packet

- Error: 2028 (CR_WRONG_LICENSE)

  Mensaje: This client library is licensed only for use with MySQL servers having '%s' license

- Error: 2029 (CR_NULL_POINTER)

  Mensaje: Invalid use of null pointer

- Error: 2030 (CR_NO_PREPARE_STMT)

  Mensaje: Statement not prepared

- Error: 2031 (CR_PARAMS_NOT_BOUND)

  Mensaje: No data supplied for parameters in prepared statement

- Error: 2032 (CR_DATA_TRUNCATED)

  Mensaje: Data truncated

- Error: 2033 (CR_NO_PARAMETERS_EXISTS)

  Mensaje: No parameters exist in the statement

- Error: 2034 (CR_INVALID_PARAMETER_NO)

  Mensaje: Invalid parameter number

- Error: 2035 (CR_INVALID_BUFFER_USE)

  Mensaje: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error: 2036 (CR_UNSUPPORTED_PARAM_TYPE)

  Mensaje: Using unsupported buffer type: %d (parameter: %d)

- Error: 2037 (CR_SHARED_MEMORY_CONNECTION)

  Mensaje: Shared memory: %s

- Error: 2038 (CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR)

  Mensaje: Can't open shared memory; client could not create request event (%lu)

- Error: 2039 (CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR)

  Mensaje: Can't open shared memory; no answer event received from server (%lu)

- Error: `2040` (`CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR`)

  Mensaje: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: `2041` (`CR_SHARED_MEMORY_CONNECT_MAP_ERROR`)

  Mensaje: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: `2042` (`CR_SHARED_MEMORY_FILE_MAP_ERROR`)

  Mensaje: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: `2043` (`CR_SHARED_MEMORY_MAP_ERROR`)

  Mensaje: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: `2044` (`CR_SHARED_MEMORY_EVENT_ERROR`)

  Mensaje: Can't open shared memory; client could not create %s event (%lu)

- Error: `2045` (`CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR`)

  Mensaje: Can't open shared memory; no answer from server (%lu)

- Error: `2046` (`CR_SHARED_MEMORY_CONNECT_SET_ERROR`)

  Mensaje: Can't open shared memory; cannot send request event to server (%lu)

- Error: `2047` (`CR_CONN_UNKNOW_PROTOCOL`)

  Mensaje: Wrong or unknown protocol

- Error: `2048` (`CR_INVALID_CONN_HANDLE`)

  Mensaje: Invalid connection handle

- Error: `2049` (`CR_SECURE_AUTH`)

  Mensaje: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: `2050` (`CR_FETCH_CANCELED`)

  Mensaje: Row retrieval was canceled by mysql_stmt_close() call

- Error: `2051` (`CR_NO_DATA`)

  Mensaje: Attempt to read column without prior row fetch

- Error: `2052` (`CR_NO_STMT_METADATA`)

  Mensaje: Prepared statement contains no metadata

- Error: `2053` (`CR_NO_RESULT_SET`)

  Mensaje: Attempt to read a row while there is no result set associated with the statement

- Error: `2054` (`CR_NOT_IMPLEMENTED`)

  Mensaje: This feature is not implemented yet

- Error: `2055` (`CR_SERVER_LOST_EXTENDED`)

  Mensaje: Lost connection to MySQL server at '%s', system error: %d

Capítulo 27. Extender MySQL

Tabla de contenidos

27.1. El interior de MySQL

Este capítulo describe muchas cosas que necesita saber cuando trabaje con el código de MySQL. Si planea contribuir al desarrollo de MySQL, quiere tener acceso al límite de código de versiones, o sólo quiere estar al día del desarrollo, siga las instrucciones en Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”. Si está interesado en conceptos internos de MySQL, debe suscribirse a la lista de correo `internals` . Esta lista tiene poco tráfico. Para más detalles sobre cómo suscribirse, consulte Sección 1.6.1.1, “Las listas de correo de MySQL”. Todos los desarrolladores en MySQL AB están en la lista `internals` y ayudamos a otra gente trabajando en el código de MySQL. Siéntase libre de usar esta lista para preguntar cosas sobre el código y para enviar parches con los que quiera contribuir al proyecto MySQL!

27.1.1. Los subprocesos (threads) MySQL

MySQL server crea los siguientes flujos:

- El flujo de conexión TCP/IP trata todas las peticiones de conexión y crea un nuevo flujo dedicado para tratar el proceso de autenticación y de consulta SQL para cada conexión.
- En Windows NT hay un flujo para tratar los named pipes que hace el mismo trabajo que el flujo de conexiones TCP/IP en peticiones de conexión por named pipe.
- El flujo de señales trata todas las señales. Este flujo también trata normalmente alarmas y llamadas `process_alarm()` para forzar tiempos máximos en las conexiones que llevan en espera demasiado tiempo.
- Si se compila `mysqld` con `-DUSE_ALARM_THREAD`,se crea un flujo dedicado que trata alarmas. Sólo se usa en algunos sistemas donde hay problemas con `sigwait()` o si quiere usar el código `thr_alarm()` en su aplicación sin un flujo dedicado a tratar señales.
- Si usa la opción `--flush_time=#` , se crea un flujo dedicado para volcar todas las tablas en un intervalo dado.
- Cada conexión tiene su propio flujo.
- Cada tabla diferente en que se usa `INSERT DELAYED` tiene su propio flujo.

- Si usa `--master-host`, se arranca un flujo de replicación esclavo para leer y aplicar actualizaciones del maestro.

`mysqladmin processlist` sólo muestra la conexión, `INSERT DELAYED`, y flujos de replicación.

27.1.2. El paquete de pruebas MySQL Test

El sistema de testeo incluído en distribuciones fuente Unix y distribuciones binarias hace que sea posible para los usuarios y desarrolladores realizar tests de regresión en código MySQL. Estos tests pueden ejecutarse en Unix o en Windows (usando el entorno Cygwin) si el servidor se ha compilado bajo Cygwin. No pueden ejecutarse en un entorno Windows nativo.

El conjunto actual de casos de uso no testea todo en MySQL, pero debería atrapar la mayoría de bugs obvios en el código de proceso SQL, temas de bibliotecas del SO, y es bastante útil para testear replicación. Nuestra meta eventual es que el test cubra el 100% del código. Los contribuidores de nuestra suite de test son bienvenidos. Puede querer contribuir con tests que examinen la funcionalidad crítica del sistema, ya que esto asegura que todas las futuras versiones de MySQL funcionen bien con sus aplicaciones.

27.1.2.1. Ejecutar el paquete de pruebas MySQL Test

El sistema de test consiste en un intérprete de idioma de test (`mysqltest`), un shell script para ejecutar todos los tests(`mysql-test-run`), los casos de usos de test actuales escritos en un lenguaje especial de test, y sus resultados esperados. Para ejecutar la test suite en su sistema tras construirlo escriba `make test` o `mysql-test/mysql-test-run` de la raíz del código fuente. Si ha instalado una distribución binaria, `cd` a la raíz de la instalación (ej. `/usr/local/mysql`), y ejecute `scripts/mysql-test-run`. Todos los tests deberían tener éxito. Si no fuera así, debería intentar encontrar el porqué y reportar el problema si es un bug de MySQL. Consulte Sección 27.1.2.3, “Reportar bugs de la suite de test de MySQL”.

Desde MySQL 4.1 , si tiene una copia de `mysqld` ejecutándose en la máquina donde quiere ejecutar el test suite no tiene que pararlo, mientras no use los puertos `9306` y `9307`. Si uno de esos puertos está ocupado, debe editar `mysql-test-run` y cambiar los valores de los puertos maestro y/o esclavo a uno que esté disponible.

Antes de MySQL 4.1, `mysql-test-run` no intenta ejecutar su propio servidor por defecto pero intenta usar su servidor que esté en ejecución. Para cambiar este comportamiento y hacer que `mysql-test-run` arranque su propio servidor, ejecútelo con la opción `--local` .

Puede ejecutar un caso de test individual con `mysql-test/mysql-test-run test_name`.

Si falla un test, debe probar ejecutar `mysql-test-run` con la opción `--force` para chequear si algún otro test falla.

27.1.2.2. Extender el paquete de pruebas MySQL Test

Puede usar el lenguaje `mysqltest` para crear sus propios casos de test. Desafortunadamente, todavía no hay una documentación completa para el mismo. Sin embargo, puede ojear nuestros casos de uso y usarlos como ejemplo. Los siguientes puntos pueden ayudarle a empezar:

- Los tests se encuentran en `mysql-test/t/*.test`
- Un caso de test consiste en comandos terminados por `;` y es similar a la entrada del cliente de línea de comandos `mysql` . Un comando por defecto es una consulta que se envía a MySQL server, a no ser que se reconozca como comando interno (p.e. `sleep`).

- Todas las consultas que producen resultados--por ejemplo, `SELECT`, `SHOW`, `EXPLAIN`, etc., debe precederse con *`@/path/to/result/file`*. El fichero debe contener los resultados esperados. Una forma fácil de generar el fichero de resultados es ejecutar `mysqltest -r <t/test-case-name.test` del directorio `mysql-test`, y editar los ficheros resultantes, si es necesario, para ajustarlos a la salida esperada. En ese caso, sea muy cuidadoso de no añadir o borrar caracteres invisibles -- asegúrese de cambiar sólo el texto y/o borrar líneas. Si tiene que insertar una línea, asegúrese que los campos estén separados por un tabulador, y que hay un tabulador al final. Si quiere puede usar `od -c` para asegurarse que su editor de texto no ha desbaratado nada durante la edición. Esperamos que nunca tenga que editar la salida de `mysqltest -r` ya que sólo tiene que hacerlo al encontrar un bug.
- Para ser consistente con nuestra inicialización, debe poner sus ficheros resultantes en el directorio `mysql-test/r` y llamarlos `test_name.result`. Si el test produce más de un resultado, debe usar `test_name.a.result`, `test_name.b.result`, etc.
- Si un comando retorna un error, debe especificarlo con `--error error-number` en la línea anterior al comando. El número de error puede ser una lista de números de error posibles separados por ',
'.
- Si está escribiendo un caso de test de replicación, debe poner en la primera línea del fichero de test, `source include/master-slave.inc;`. Para cambiar entre maestro y esclavo, use `connection master;` y `connection slave;`. Si necesita hacer algo en una conexión alternativa, puede hacer `connection master1;` para el maestro, y `connection slave1;` para el esclavo.
- Si necesita hacer algo en un bucle, puede usar algo como esto:

```
let $1=1000;
while ($1)
{
 # do your queries here
 dec $1;
}
```

- Para dormir entre consultas, use el comando `sleep`. Soporta fracciones de segundo, así que puede usar `sleep 1.3;`, por ejemplo, para dormir 1.3 segundos.
- Para ejecutar el esclavo con opciones adicionales para sus casos de test, póngalos en el formato de línea de comandos en `mysql-test/t/test_name-slave.opt`. Para el maestro, póngalas en `mysql-test/t/test_name-master.opt`.
- Si tiene una pregunta sobre la suite de test, o tiene un caso de test al que contribuir, envíe un email a la lista de correo MySQL `internals` . Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”. Como esta lista no acepta adjuntos, debe subir por ftp todos los ficheros relevantes a: ftp://ftp.mysql.com/pub/mysql/upload/

27.1.2.3. Reportar bugs de la suite de test de MySQL

Si su versión de MySQL no pasa el suite de test debe hacer lo siguiente:

- No envíe un reporte de bug antes que haya encontrado todo lo posible sobre qué ha ido mal! Cuando lo haga, por favor use el script `mysqlbug` para que podamos obtener información sobre el sistema y la versión MySQL. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.
- Asegúrese de incluir la salida de `mysql-test-run`, así como los contenidos de todos los ficheros `.reject` en el directorio `mysql-test/r` .
- Si falla un test en la suite, chequee si el test también falla al ejecutarlo en solitario:

```
cd mysql-test
```

```
mysql-test-run --local test-name
```

Si esto falla, debe configurar MySQL con `--with-debug` y ejecutar `mysql-test-run` con la opción `--debug` . Si también falla envíe el fichero de traza `var/tmp/master.trace` a ftp://ftp.mysql.com/pub/mysql/upload/ para que podamos examinarlo. Por favor recuerde de incluir una descripción completa de su sistema, la versión del binario `mysqld` y cómo lo compiló.

- También intente ejecutar `mysql-test-run` con la opción `--force` para ver si hay algún otro test que falla.
- Si ha compilado MySQL usted mismo, chequee nuestro manual para ver cómo compilar MySQL en su plataforma, o preferiblemente, usar uno de los binarios que hemos compilado en http://dev.mysql.com/downloads/. Todos nuestros binarios estándar deben pasar la suite de tests!
- Si obtiene un error como `Result length mismatch` o `Result content mismatch` significa que la salida del test no coincide exactamente con la salida esperada. Este puede ser un bug en MySQL o que su versión de `mysqld` produce resultados ligeramente distintos bajo algunas circunstancias.

  Los resultados de tests fallidos se ponen en un fichero con el mismo nombre base que el fichero de resultados con la extensión `.reject` . Si su caso de test falla, debe hacer un diff entre los dos ficheros. Si no puede ver en qué se distinguen, exemine ambos con `od -c` y compruebe los tamaños.
- Si un test falla completamente, debe chequear los ficheros de logs en el directorio `mysql-test/var/log` para ayudas sobre qué ha fallado.
- Si ha compilado MySQL con depuración puede intentar depurarlo ejecutando `mysql-test-run` con las opciones `--gdb` y/o `--debug` . Consulte Sección D.1.2, “Crear ficheros de traza”.

  Si no ha compilado MySQL para depuración debería hacerlo. Especifique la opción `--with-debug` en `configure`. Consulte Sección 2.8, “Instalación de MySQL usando una distribución de código fuente”.

27.2. Añadir nuevas funciones a MySQL

Hay dos formas de añadir nuevas funciones a MySQL:

- Puede añadir funciones con la interfaz de funciones definidas de usuario (UDF) (N.del T. Acrónimo para User Defined Functions). Las funciones definidas por el usuario se ocmpilan como ficheros objeto y se añaden y borran del servidor dinámicamente usando los comandos `CREATE FUNCTION` y `DROP FUNCTION` . Consulte Sección 27.2.2, “Sintaxis de `CREATE FUNCTION/DROP FUNCTION`”.
- Puede añadir funciones como funciones nativas MySQL. Se compilan en el servidor `mysqld` y están disponibles permanentemente.

Cada método tiene ventajas y desventajas:

- Si escribe funciones definidas por el usuario, debe instalar ficheros objeto además del servidor mismo. Si compilar su función en el servidor, no necesita hacerlo.
- Puede añadir UDFs a distribuciones binarias MySQL . Las funciones nativas requieren modificar una distribución fuente.
- Si actualiza su distribución MySQL , puede continuar usando las UDFs previamente instaladas, a no ser que actualice a una versión en la que la interfaz UDF cambie. (Un cambio incompatible ocurrió en MySQL 4.1.1 para funciones agregadas. Una función llamada `xxx_clear()` debe definirse en lugar de `xxx_reset()`.) Para funciones nativas, debe repetir sus modificaciones cada vez que actualice.

Use el método que use para añadir nuevas fucnciones, pueden invocarse en comandos SQL como funciones nativas tales como `ABS()` o `SOUNDEX()`.

Otra forma de añadir funciones es creando funciones almacenadas. Se escriben con comandos SQL en lugar de compilando código objeto. La sintaxis para escribr funciones almacenadas se describe en `Stored Procedures`.

La siguiente sección describe características de la interfaz UDF, proporciona intrucciones para escribir UDFs, y discute sobre precauciones de seguridad que toma MySQSL para prevenir un mal uso de UDF.

Para código fuente de ejemplo que ilustra cómo escribir UDFs, mire el fichero `sql/udf_example.cc` que se proporciona en las distribuciones fuentes de MySQL.

27.2.1. Características de la interfaz para funciones definidas por el usuario

La interfaz de MySQL para funciones definidas por el usuario proporciona las siguientes funcionalidades y capacidades:

- Las funciones pueden retornar cadenas de caracteres, enteros o valores reales.
- Puede definir funciones simples que operen en con un único registro a la vez, o agregar funciones que operen con grupos de registros.
- Se proporciona información a las funciones que permite chequear el tipo y número de argumentos que se les pasa.
- Le puede decir a MySQL que coercione argumentos de un tipo dado antes de pasarlos a la función.
- Puede indicar que una funcion retorne `NULL` o que ha ocurrido un error.

27.2.2. Sintaxis de `CREATE FUNCTION/DROP FUNCTION`

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL}
       SONAME shared_library_name

DROP FUNCTION function_name
```

Una funciones definidas por el usuario (UDF) es un modo de extender MySQL con una nueva función que funciona como una función nativa de MySQL tal como `ABS()` o `CONCAT()`.

`function_name` es el nombre que debe usarse en comandos SQL para invocar la función. La cláusula `RETURNS` indica el tipo del valor de retorno de la función. *`shared_library_name`* es el nombre base de la fichero del objeto compartido que contiene el código que implementa la función. El fichero debe localizarse en un directorio en el que busque el lincador dinámico del sistema.

Para crear una función , debe tener el privilegio `INSERT` para la base de datos `mysql` . Para borrar una función, debe tener el privilegio `DELETE` para la base de datos `mysql` . Esto es así porque `CREATE FUNCTION` añade un registro a la tabla de sistema `mysql.func` que registra los nombres de función, tipo, y nombre de la biblioteca compartida, y `DROP FUNCTION` borra el registro de la función de dicha tabla. Si no tiene esta tabla, debe ejecutar el script `mysql_fix_privilege_tables` para crearla. Consulte Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.

Una función activa es una que se ha cargado con `CREATE FUNCTION` y no se ha eliminado con `DROP FUNCTION`. Todas las funciones activas se recargan cada vez que el servidor arranca, a no ser que arranque `mysqld` con la opción `--skip-grant-tables` . En este caso, la inicialización de UDF no se hace y no están disponibles.

Para instrucciones sobre escribir funciones definidas por el usuario consulte Sección 27.2.3, “Añadir una nueva función definida por el usuario”. Para que funcione el mecanismo de UDF, las funciones deben

escribirse en C o C++, su sistema operativo debe soportar carga dinámica y debe haber compilado `mysqld` dinámicamente (no estáticamente).

`AGGREGATE` es una nueva opción para MySQL 3.23. Una función `AGGREGATE` funciona exactamente como una función agregada (resumen) de MySQL tal como `SUM` o `COUNT()`. Para que funcione `AGGREGATE` , su tabla `mysql.func` debe contener una columna `type` . Si su tabla `mysql.func` no tiene esta columna, debe ejecutar el script `mysql_fix_privilege_tables` para crearla.

27.2.3. Añadir una nueva función definida por el usuario

Para que funciones el mecanismo UDF, las funciones deben escribirse en C o C++ y su sistema operativo debe soportar carga dinámica. La distribución fuente de MySQL incluye un fichero `sql/udf_example.cc` que define 5 nuevas funciones. Consulte este fichero para ver cómo funcionan las convenciones de llamadas de UDF.

Para poder usar UDFs, necesita lincar `mysqld` dinámicamente. No configure MySQL usando `--with-mysqld-ldflags=-all-static`. Si quiere usar una UDF que necesite acceder a símbolos desde `mysqld` (por ejemplo la función `metaphone` en `sql/udf_example.cc` que usa `default_charset_info`), debe lincar el programa con `-rdynamic` (consulte `man dlopen`). Si planea usar UDFs, la rula es configurar MySQL con `--with-mysqld-ldflags=-rdynamic` a no ser que tenga una muy buena razón para no hacerlo.

Si usa una distribución precompilada de MySQL, use MySQL-Max, que contiene un servidor lincado dinámicamente que soporta carga dinámica.

Para cada función que quiera usar en comandos SQL, debe definir las funciones correspondientes en C (o C++). En la siguiente discusión, el nombre “xxx” se usa como nombre de función de ejemplo. Para distinguir entre el uso de SQL y C/C++ , `XXX()` (mayúsculas) indicata una llamada de función SQL, `xxx()` (minúsculas) indica una llamada de función C/C++ .

Las funciones C/C++ que escribe para implementar la interfaz para `XXX()` son:

- `xxx()` (requerido)

  La función principal. Es donde el resultado de la función se computa. La correspondencia entre los tipos de datos de la función SQL y el tipo de retorno de la función C/C++ se muestra aquí:

SQL Type	C/C++ Type
`STRING`	`char *`
`INTEGER`	`long long`
`REAL`	`double`

- `xxx_init()` (opcional)

  La función de inicialización para `xxx()`. Puede usarse para:

  - Chequea el número de argumentos para `XXX()`.
  - Chequea que los argumentos son de un tipo requerido o, alternativamente, le dice a MySQL que coercione argumentos a los tipos que quiera cuando se llama a la función principal.
  - Reserva cualquier memoria requerida por la función principal.
  - Especifica la longitud máxima del resultado.

  - Especifica (para funciones `REAL`) el máximo número de decimales.
  - Especifica si el resultado puede ser `NULL`.

- `xxx_deinit()` (opcional)

  La función de deinicialización para `xxx()`. Debe liberar cualquier memoria reservada por la función de inicialización.

Cuando un comando SQL invoca `XXX()`, MySQL llama a la función de inicialización `xxx_init()` para que realice cualquier inicialización necesaria, tales como chequeo de argumentos o reserva de memoria. Si `xxx_init()` retorna un error, el comando SQL se aborta con un mensaje de error y no se llama ni a la función principal ni a la de deinicialización. En caso contrario, se llama a la función principal `xxx()` una vez para cada registro. Tras procesar todos los registros, se llama a la función de deinicialización `xxx_deinit()` para que pueda realizar cualquier limpieza requerida.

Para funciones agregadas que funcionan como `SUM()`, debe proporcionar las siguientes funciones:

- `xxx_reset()` (necesaria antes de 4.1.1)

  Resetea el valor agregado actual e inserta el argumento como valor agregado inicial para un nuevo grupo.

- `xxx_clear()` (requerido a partir de 4.1.1)

  Resetea el valor agregado actual pero no inserta el argumento como valor agregado inicial para un nuevo grupo.

- `xxx_add()` (requerido)

  Añade el argumento al valor agregado actual.

MySQL trata UDFs agregados como sigue:

1. Llama a `xxx_init()` para permitir a la función agregada reservar la memoria necesaria para ordenar resultados.
2. Ordena la table según la expresión `GROUP BY`.
3. Llama a `xxx_clear()` para el primer registro en cada grupo.
4. Llama a `xxx_add()` para cada nuevo registro que permita al mismo grupo.
5. Llama a `xxx()` para obtener el resultado del agregado cuando el grupo cambia o cuando el último registro se ha procesado.
6. Repite 3-5 hasta que se procesan todos los registros
7. Llama a `xxx_deinit()` para permitir al UDF liberar la memoria reservada.

Todas las funciones deben ser flujos seguros. Esto incluye no sólo la función principal,también las funciones de inicialización o deinicialización, y las funciones adicionales requeridas por las funciones agregadas. Una consecuencia de esta restricción es que no se le permite reservar ninguna variable global o estática que cambien! Si necesita memoria, debe reservarla en `xxx_init()` y liberarla en `xxx_deinit()`.

27.2.3.1. Secuencias de llamada UDF para funciones simples

Esta sección describe las distintas funciones que necesita definir cuando crea un UDF simple. Sección 27.2.3, “Añadir una nueva función definida por el usuario” describe el orden en que MySQL llama a estas funciones.

La función principal `xxx()` debe declararse como se muestra en esta sección. Tenga en cuenta que el tipo de retorno y los parámetros difieren, dependiendo de si declara la función SQL `XXX()` para retornar `STRING`, `INTEGER`, o `REAL` en el comando `CREATE FUNCTION` :

Para funciones `STRING` :

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

Para funciones `INTEGER` :

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

Para funciones `REAL` :

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

Las funciones de inicialización y deinicialización se declaran así:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

El parámetro `initid` se pasa a las tres funciones. Apunta a la estructura `UDF_INIT` que se usa para comunicar información entre funciones. Los mienbros de la estructura `UDF_INIT` se muestran a continuación. La función de inicialización debe rellenar cualquier miembro que quiera cambiar. (Para usar el valor por defecto de un miembro no lo cambie.)

- `my_bool maybe_null`

  `xxx_init()` debe asignar a `maybe_null` `1` si `xxx()` puede retornar `NULL`. El valor por defecto es `1` si alguno de los argumentos se declaran `maybe_null`.

- `unsigned int decimals`

  El número de decimales. El valor por defecto es el número máximo de decimales en los argumentos pasados a la función principal. (Por ejemplo, si a la función se pasa `1.34`, `1.345`, y `1.3`, el valor por defecto es 3, ya que `1.345` tiene 3 decimales.

- `unsigned int max_length`

  Longitud máxima del resultado. El valor por defecto `max_length` difiere en función del tipo de resultado de la función. Para funciones de cadenas de caracteres, el valor por defecto es el argumento más largo. Para funciones enteras, el valor por defecto es de 21 dígitos. Para funciones reales, el valor por defecto es 13 mas el número de decimales indicados por `initid->decimals`. (Para funciones numéricas, la longitud incluye cualquier signo o carácter de punto decimal.)

  Si quiere retornar un valor blob, puede asignar a `max_length` de 65KB a 16MB. Esta memoria no se reserva, pero el valor se usa para decidir qué tipo de columna usar si hay una necesidad de almacenar los datos temporalmente.

- `char *ptr`

  Puntero que la función puede usar para su propio propósito. Por ejemplo, las funciones pueden usar `initid->ptr` para comunicar memoria reservada entre ellos. `xxx_init()` debe reservar la memoria y asignarla al puntero:

```
initid->ptr = allocated_memory;
```

  En `xxx()` y `xxx_deinit()`, refiérase a `initid->ptr` para usar o liberar la memoria.

27.2.3.2. Secuencias de llamada UDF para funciones agregadas

Esta sección describe las distintas funciones que necesita definir cuando crea un UDF agregado. Sección 27.2.3, “Añadir una nueva función definida por el usuario” describe el orden en que MySQL llama a estas funciones.

- `xxx_reset()`

  Esta función se llama cuando MySQL encuentra el primer registro en un nuevo grupo. Debe resetear cualquier variable resumen interna y usar el argumento dado `UDF_ARGS` como primer valor en su resumen interno del grupo. Declare `xxx_reset()` como se muestra:

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                char *is_null, char *error);
```

  `xxx_reset()` se necesita sólo antes de MySQL 4.1.1. *NO* se necesita usar desde MySQL 4.1.1, cuando la interfaz UDF cambió para usar `xxx_clear()` en su lugar. Sin embargo, puede definir `xxx_reset()` y `xxx_clear()` si quiere que su UDF funcione antes y después del cambio de interfaz. (Si no incluye ambas funciones, la función `xxx_reset()` en muchos casos puede implementarse internamente llamando `xxx_clear()` para resetear todas las variables, y luego llamar `xxx_add()` para añadir el argumento `UDF_ARGS` como primer valor del grupo.)

- `xxx_clear()`

  Esta función se llama cuando MySQL necesita resetear los resultados resumen. Se llama al principio para cada nuevo grupo pero sólo puede llamarse para resetear los valores para una consulta donde no hubieran registros que coincidan con la búsqueda. Declare `xxx_clear()` como sigue:

```
char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

  `is_null` se asigna para que apunte a `CHAR(0)` antes de llamar a `xxx_clear()`.

  Si algo falla, puede almacenar un valor en la variable a la que apunta el argumento `error`. `error` apunta a una variable de un byte, no a un búfer de cadenas de caracteres.

  `xxx_clear()` se requiere sólo a partir de MySQL 4.1.1. Antes de MySQL 4.1.1, use `xxx_reset()` en su lugar.

- `xxx_add()`

  Esta función se llama para todos los registros que pertenezcan al mismo grupo, excepto para el primer registor. Debe ussarlo para añadir el valor en el argumento `UDF_ARGS` a su variable de resumen interna.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
```

```
              char *is_null, char *error);
```

La función `xxx()` para un UDF agregado debe declararse de la misma forma que UDF no agregados. Consulte Sección 27.2.3.1, “Secuencias de llamada UDF para funciones simples”.

Para un UDF agregado, MySQL llama a la función `xxx()` una vez que todos los registros en el grupo han sido procesados. Normalmente no debe acceder el argumento `UDF_ARGS` aquí sino devolver un valor basado en sus variables de resumen internas.

El tratamiento de valores retornados en `xxx()` debe hacerse del mismo modo que para UDF no agregados. Consulte Sección 27.2.3.4, “Valores de retorno y tratamiento de errores”.

Las funciones `xxx_reset()` y `xxx_add()` tratan sus argumentos `UDF_ARGS` del mismo modo que las funciones para UDFs no agregados. Consulte Sección 27.2.3.3, “Proceso de argumentos”.

Los argumentos punteros de `is_null` y `error` son los mismos para todas las llamadas a `xxx_reset()`, `xxx_clear()`, `xxx_add()` y `xxx()`. Puede usar esto para recordar que obtuvo un error o si la función `xxx()` debería retornar `NULL`. No debe almacenar una cadena de caracteres en `*error`! `error` apunta a una variable de un byte, no a un búfer de cadenas de caracteres.

`*is_null` se resetea para cada grupo (antes de llamar `xxx_clear()`). `*error` nunca se resetea.

Si `*is_null` o `*error` se asignan cuando `xxx()` retorna, MySQL retorna `NULL` como resultado para la función de grupo.

27.2.3.3. Proceso de argumentos

El parámetro `args` apunta a una estructura `UDF_ARGS` que tiene los miembros listados a continuación:

- `unsigned int arg_count`

  Número de argumentos. Chequee este valor en la función de inicialización si necesita que su función sea llamada con un número particular de argumentos. Por ejemplo:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

- `enum Item_result *arg_type`

  Puntero a una matriz conteniendo los tipos para cada argumento. Los tipos posibles son `STRING_RESULT`, `INT_RESULT`, y `REAL_RESULT`.

  Par asegurar que los argumentos sean de un tipo dado y retorne un error si no lo son, chequee la matriz `arg_type` en la función de inicialización. Por ejemplo:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

  Como alternativa a requerir que los argumentos de la función sean de un tipo particular, puede usar la función de inicialización para asignar los elementos `arg_type` con los tipos que quiera. Esto hace que MySQL fuerce a los argumentos a los tipos para cada llamada de `xxx()`. Por ejemplo, para especificar

que los primeros dos argumentos se fuercen a una cadena de caracteres de enteros, respectivamente, haga lo siguiente en `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

- `char **args`

  `args->args` comunica información a la función de inicialización acerca de la naturaleza general de los argumentos pasados a la función. Para un argumento constante `i`, `args->args[i]` apunta al valor del argumento. (Consulte a continuación instrucciones sobre cómo acceder al valor apropiadamente.) Para argumentos no constantes, `args->args[i]` es `0`. Un argumento constante es una expresión que usa sólo constantes, como `3` o `4*7-2` o `SIN(3.14)`. Un argumento no constante es una expresión que se refiere a valores que pueden cambiar de registro a registro, tales como nombres de columna o funciones que se llaman con argumentos no constantes.

  Para cada invocación de la función principal, `args->args` contiene los argumentos que se pasan para el registro que está procesando.

  Las funciones pueden referirse a un argumento `i` como se muestra:

  - Un argumento de tipo `STRING_RESULT` se da como puntero a una cadena de caracteres más una longitud, para permitir tratar datos binarios o datos arbitráriamente largos. Los contenidos de la cadena de caracteres están disponibles como `args->args[i]` y la longitud de la cadena es `args->lengths[i]`. No debe asumir que las cadenas de caracteres están terminadas por null.

  - Para un argumento de tipo `INT_RESULT`, debe convertir `args->args[i]` a un valor `long long` :

    ```
    long long int_val;
    int_val = *((long long*) args->args[i]);
    ```

  - Para un argumento de tipo `REAL_RESULT`, debe convertir `args->args[i]` a un valor `double` :

    ```
    double    real_val;
    real_val = *((double*) args->args[i]);
    ```

- `unsigned long *lengths`

  Para la función de inicialización, la matriz `lengths` indica la longitud máxima de cadena de caracteres para cada argumento. No debe cambiar este valor. Para cada invocación de la función principal `lengths` contiene las longitudes reales de cualquier argumento de cadenas de caracteres que se pasa al registro en proceso. Para argumentos de tipos `INT_RESULT` o `REAL_RESULT`, `lengths` contiene la longitud máxima del argumento (como para la función de inicialización).

27.2.3.4. Valores de retorno y tratamiento de errores

La función de inicialización debe retornar `0` si no hay errores y `1` en cualquier otro caso. Si ocurre un error, `xxx_init()` debe almacenar un mensaje de error terminado en null en el parámetro `message` . El mensaje se retorna al cliente. El búffer de mensajes es de longitud `MYSQL_ERRMSG_SIZE`, pero debe tratar que el mensaje ea inferior a 80 caracteres para que coincida con la anchura de una pantalla de terminal estándar.

El valor retornado por una función principal `xxx()` es el valor de la función, para funciones `long long` y `double` . Una función de cadenas de caracteres debe retornar un puntero al resultado y asignar `*result` y `*length` con los contenidos y longitud del valor de retorno. Por ejemplo:

```
memcpy(result, "result string", 13);
*length = 13;
```

El búffer `result` que se pasa a la función `xxx()` tiene longitud de 255 bytes. Si su resultado coincide con esto, no tiene que preocuparse acerca de reservar memoria para los resultados.

Si su función de cadenas de caracteres necesita retornar una cadena de caracteres mayor a 255 bytes, debe reservar el espacio para ello con `malloc()` en su función `xxx_init()` o su función `xxx()` y liberarla en su función `xxx_deinit()` . Puede almacenar la memoria reservada en la entrada `ptr` en la estructura `UDF_INIT` para reusar para llamadas futuras `xxx()` . Consulte Sección 27.2.3.1, “Secuencias de llamada UDF para funciones simples”.

Para indicar un valor de retorno de `NULL` en la función principal, asigne a `*is_null` `1`:

```
*is_null = 1;
```

Para indicar un retorno de error en la función principal, inicialice `*error` con `1`:

```
*error = 1;
```

Si `xxx()` asigna a `*error` `1` para cualquier registro, el valor de la función es `NULL` para el registro actual y para cualquier subsecuente registro procesado por el comando en que se invoca `XXX()` . (`xxx()` no se llama ni por registros subsecuentes.) **Nota**: Antes de MySQL 3.22.10, debe asignar tanto `*error` como `*is_null`:

```
*error = 1;
*is_null = 1;
```

27.2.3.5. Compilar e instalar funciones definidas por el usuario

Los ficheros implementando UDFs deben compilarse e instalarse en el equipo donde corre el servidor. Este proceso se describe a continuación para el fichero UDF de ejemplo `sql/udf_example.cc` que se incluye en la distribución fuente de MySQL.

Las instrucciones siguientes son para Unix. Las instructiones para Windows se dan posteriormente en esta sección.

El fichero `udf_example.cc` contiene las siguientes funciones:

- `metaphon()` retorna una cadena de caracteres metaphon del argumento de cadena de caracteres. Esto es algo como una cadena de caracteres soundex, pero más ajustado al inglés.
- `myfunc_double()` retorna la suma de los valores ASCII de los caracteres en los argumentos, divididos por la suma de la longitud de los argumentos.
- `myfunc_int()` retorna la suma de la longitud de los argumentos.
- `sequence([const int])` retorna una secuencia empezando por el número dado o 1 si no se da ningún número.
- `lookup()` retorna la IP para un nombre de equipo.
- `reverse_lookup()` retorna el nombre de equipo para una IP. La función puede llamarse con un único argumento de cadena de caracteres de la forma `'xxx.xxx.xxx.xxx'` o con cuatro números.

Un fichero cargable dinámicamente debe compilarse como fichero objeto compartible, usando un comando como:

```
shell> gcc -shared -o udf_example.so udf_example.cc
```

Si usa `gcc`, debe ser capaz de crear `udf_example.so` con un comando más simple:

```
shell> make udf_example.so
```

Puede determinar fácilmente las opciones de compilación adecuadas para su sistema ejecutando este comando en el directorio `sql` del árbol fuente de MySQL:

```
shell> make udf_example.o
```

Debe ejecutar un comando de compilación similar al que muestra `make` , excepto que debe quitar la opción `-c` cerca del final de la línea y añadir `-o udf_example.so` al final de la línea. (En algunos sistemas, puede necesitar dejar `-c` en el comando.)

Tras compilar un objeto compartido conteniendo UDFs, debe instalarlo y comunicarlo a MySQL. Compilar un objeto compartido de `udf_example.cc` produce un fichero llamado algo como `udf_example.so` (el nombre exacto puede variar de plataforma a plataforma). Copie este fichero en un directorio como `/usr/lib` en el que busque el lincador dinámico del sistema (en tiempo de ejecución), o añada el directorio en el que está el objeto compartido en el fichero de configuración del lincador (por ejemplo, `/etc/ld.so.conf`).

El nombre del lincador depende del sistema (por ejemplo, `ld-elf.so.1` en FreeBSD, `ld.so` en Linux, o `dyld` en Mac OS X). Consulte su documentación del sistema para información acerca del nombre del lincador y cómo configurarlo.

En muchos sistemas, puede cambiar las variables de entorno `LD_LIBRARY` o `LD_LIBRARY_PATH` para que apunten al directorio donde tiene los ficheros de su UDF. La página de manual `dlopen` explica qué variables debe usar en su sistema. Debe inicializarla en los scripts de arranque `mysql.server` o `mysqld_safe` y reiniciar `mysqld`.

En algunos sistemas, el programa `ldconfig` que configura el lincador dinámico no reconoce un objeto compartido a no ser que su nombre comience con `lib`. En ese caso debe renombar el fichero como `udf_example.so` a `libudf_example.so`.

En Windows, puede compilar funciones definidas por el usuario usando el siguiente procedimiento:

1. Necesita obtener el repositorio fuente BitKeeper para MySQL 4.0 o superior. Consulte Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”.
2. En el repositorio fuente, busque el directorio `VC++Files/examples/udf_example` . Hay ficheros llamados `udf_example.def`, `udf_example.dsp`, y `udf_example.dsw` .
3. En el repositorio fuente, busque en el directorio `sql` . Copie `udf_example.cc` de este directorio al directorio `VC++Files/examples/udf_example` y renombre el fichero a `udf_example.cpp`.
4. Abra el fichero `udf_example.dsw` con Visual Studio VC++ y úselo para compilar los UDFs como un proyecto normal.

Cuando el objeto compartido se ha instalado, notifique a `mysqld` con las nuevas funciones con estos comandos:

```
```

```
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 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';
```

Las funciones pueden borrarse con `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

Los comandos `CREATE FUNCTION` y `DROP FUNCTION` actualizan la tabla de sistema `func` en la base de datos `mysql` . El nombre de función, tipo y nombre de biblioteca compartida se salvan en la tabla. Debe tener los privilegios `INSERT` y `DELETE` para la base de datos `mysql` para crear y borrar funciones.

No debe usar `CREATE FUNCTION` para añadir una función que se ha creado préviamente. Si necesita reinstalar una función, debe borrarla con `DROP FUNCTION` y reinstalarla con `CREATE FUNCTION`. Puede necesitar hacer esto, por ejemplo si recompila una nueva versión de su función, de forma que `mysqld` tenga la nueva versión. De otro modo, el servidor continua usando la versión antigua.

Una función activa es una que se ha cargado con `CREATE FUNCTION` y no se ha borrado con `DROP FUNCTION`. Todas las funciones activas se recargan cada vez que el servidor arranca, a no ser que arranque `mysqld` con la opción `--skip-grant-tables` . En ese caso, la inicialización de UDFs no se hace y no están disponibles.

27.2.3.6. Precauciones de seguridad en funciones definidas por usuarios

MySQL toma las siguientes medidas para evitar uso inadecuado de funciones definidas por el usuario.

Debe tener el privilegio `INSERT` para poder usar `CREATE FUNCTION` y el privilegio `DELETE` para poder usar `DROP FUNCTION`. Esto es necesario ya que estos comandos añaden y borran registros de la tabla `mysql.func` .

UDFs debe tener como mínimo un símbolo definido además del símbolo `xxx` que corresponde a la función principal `xxx()` . Estos símbolos auxiliares se corresponden con las funciones `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, y `xxx_add()` . Desde MySQL 4.0.24, 4.1.10a, y 5.0.3, `mysqld` soporta una opción `--allow-suspicious-udfs` que controla si UDFs que tienen un solo símbolo `xxx` pueden cargarse. Por defecto, la opción está desactivada, para evitar intentos de cargar funciones de ficheros de objetos compartidos otros aparte de los que contienen UDFs legítimos. Si tiene UDFs antiguos que contienen sólo el símbolo `xxx` y no pueden recompilarse para incluir un símbolo auxiliar, puede ser necesario especificar la opción `--allow-suspicious-udfs` . De otro modo, debe evitar activar esta capacidad.

Los ficheros objeto UDF no pueden guardarse en un directorio arbitrario. Deben estar localizado en un directorio de sistema en el que busque el lincador dinámico. Para forzar esta restricción y evitar intentos de especificar rutas fuera de los directorios buscados por el lincador dinámico, MySQL chequea el fichero de objeto compartido especificado en comandos `CREATE FUNCTION` para delimitadores de rutas. Desde MySQL 4.0.24, 4.1.10a, y 5.0.3, MySQL también chequea los delimitadores de rutas en nombres de ficheros almacenados en la tabla `mysql.func` cuando carga funciones. Esto evita intentos de especificar rutas ilegítimas manipuladas en la tabla `mysql.func` . Para información acerca de UDFs y lincador en tiempo de ejecución, consulte Sección 27.2.3.5, “Compilar e instalar funciones definidas por el usuario”.

27.2.4. Añadir una nueva función nativa

El procedimiento para añadir una nueva función nativa se describe aquí. Tenga en cuenta que no puede añadir funciones nativas a una distribución binaria ya que el procedimiento implica modificar código fuente MySQL. Debe compilar MySQL de una distribución fuente. También tenga en cuenta que si migra a otra versión de MySQL (por ejemplo, cuando una nueva versión aparece), necesita repetir el procedimiento con la nueva versión.

Para añadir una nueva función MySQL nativa, siga estos pasos:

1. Añada una línea en `lex.h` que defina el nombre de función en la matriz `sql_functions[]`.

2. Si el prototipo de función es simple (sólo tiene cero, uno, dos o tres argumentos), debe especificar en `lex.h` `SYM(FUNC_ARGN)` (donde *`N`* es el número de argumentos) como el segundo argumento en la matriz `sql_functions[]` y añadir una nueva función que cree un objeto función en `item_create.cc`. Consulte `"ABS"` y `create_funcs_abs()` para un ejemplo.

   Si la función prototipo es complicada (por ejemplo, tiene un número variable de argumentos), debe añadir dos líneas en `sql_yacc.yy`. Una indica el símbolo de preprocesador que `yacc` debe definir (debe añadirse al principio del fichero). Luego defina los parámetros de función y añada un “item” con estos parámetros a la regla de parseo `simple_expr` . Para un ejemplo, consulte todas las ocurrencias de `ATAN` en `sql_yacc.yy` para ver cómo se hace.

3. En `item_func.h`, declare una clase heredando de `Item_num_func` o `Item_str_func`, en función de si su función retorna un número o una cadena de caracteres.

4. En `item_func.cc`, añada una de las siguientes declaraciones, dependiendo de si está definiendo una función numérica o de cadena de caracteres:

   ```
   double   Item_func_newname::val()
   longlong Item_func_newname::val_int()
   String  *Item_func_newname::Str(String *str)
   ```

   Si hereda su objeto de cualquiera de los objetos estándar (como `Item_num_func`), probablemente sólo tiene que definier una de estas funciones y dejar que el objeto padre se ocupe de las otras funciones. Por ejemplo, la clase `Item_str_func` define una función `val()` que ejecuta `atof()` en el valor retornado por `::str()`.

5. Debería probablemente definir la siguiente función objeto:

   ```
   void Item_func_newname::fix_length_and_dec()
   ```

   Esta función debe calcular al menos `max_length` basándose en los argumentos dados. `max_length` es el máximo número de caracteres que la función puede retornar. Esta función debería también asignar `maybe_null = 0` si la función principal no puede retornar un valor `NULL` . La función puede chequear si algunos de los argumentos de la función puede retornar `NULL` chequeando la variable `maybe_null` de los argumentos. Puede consultar `Item_func_mod::fix_length_and_dec` para un ejemplo típico de cómo hacer esto.

Todas las funciones deben ser flujos seguros. En otras palabras, no usar ninguna variable global o estática en las funciones sin protegerlas con semáforos

Si quiere retornar `NULL`, desde `::val()`, `::val_int()` or `::str()` debe asignar a `null_value` 1 y retornar 0.

Para funciones objeto `::str()` , hay algunas consideraciones adicionales a tener en cuenta:

- El argumento `String *str` proporciona un búffer de cadenas de caracteres que puede usarse para guardar el resultado. (Para más información acerca del tipo `String` , consulte el fichero `sql_string.h` .)
- La función `::str()` debe retornar la cadena de caracteres que tiene el resultado o `(char*) 0` si el resultado es `NULL`.
- Todas las funciones de cadenas de caracteres tratan de evitar cualquier reserva de memoria a no ser que sea absolutamente necesario!

27.3. Añadir nuevos procedimientos a MySQL

En MySQL, puede definir un procedimiento en C++ que pueda acceder y modificar los datos en una consulta antes de enviarla al cliente. La modificación puede hacerse registro a registro o a nivel `GROUP BY` .

Hemos creado un procedimiento de ejemplo en MySQL 3.23 para mostrar cómo se hace.

Adicionalmente, recomendamos que consulte `mylua`. Con esto puede usar el lenguaje LUA para cargar procedimientos en tiempo de ejecución en `mysqld`.

27.3.1. Procedimiento Analyse

`analyse([`*`max_elements`*`,[`*`max_memory`*`]])`

Este procedimiento se define en `sql/sql_analyse.cc`. Esto examina el resultado de su consulta y retorna un análisis del resultado:

- *`max_elements`* (por defecto 256) es el máximo número de valores distintos que `analyse` puede tratar por columna. Esto lo usa `analyse` para chequear si el tipo óptimo de columna debe ser de tipo `ENUM`.
- *`max_memory`* (por defecto 8192) es la cantidad máxima de memoria que `analyse` puede reservar por columna mientras trata de encontrar todos los valores distintos.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

27.3.2. Escribir un procedimiento

De momento, la única documentación para esto son las fuentes.

Puede encontrar toda la información acerca de procedimientos examinando los siguientes ficheros:

- `sql/sql_analyse.cc`
- `sql/procedure.h`
- `sql/procedure.cc`
- `sql/sql_select.cc`

Apéndice A. Problemas y errores comunes

Tabla de contenidos

Este apéndice enumera algunos problemas comunes y mensajes de error que usted podría encontrarse. Explica como determinar los motivos de los problemas y qué hacer para resolverlos.

A.1. Cómo determinar a qué es debido un problema

Cuando se encuentre un problema, la primera cosa que debe hacer es determinar qué programa o pieza de hardware lo está causando:

- Si tiene uno de los siguientes síntomas, entonces es probable que sea un problema de hardware (como memória, placa madre, CPU, o disco duro), o un problema del núcleo del sistema operativo:
  - El teclado no funciona. Esta anomalía puede comprobarse normalmente pulsando la tecla Bloq Mayus (Caps Lock). Si la luz de bloqueo de mayúsculas no se enciende, debería cambiar su teclado. (Antes de hacer esto, debería intentar reiniciar la máquina y comprobar todo el cableado del teclado).
  - El puntero del ratón no se mueve.
  - La máquina no responde a pings de máquinas remotas.
  - Otros programas que no están relacionados con MySQL no se comportan correctamente.
  - El sistema se reinició inesperadamente. (Un programa de nivel de usuario defectuoso nunca debería ser capaz de hacer caer el sistema.)

  En este caso, debería comenzar por comprobar todos los cables y ejecutar alguna herramienta de diagnóstico para comprobar el hardware. Debería también comprobar si hay algún parche, actualización, o paquetes de servicio para su sistema operativo que podría resolver su problema. Compruebe también que todas las librerías (tales como `glibc`) están actualizadas.

  Siempre es bueno utilizar una máquina con memoria ECC para descubrir los problemas de memoria lo antes posible.
- Si el teclado está bloqueado, debería ser capaz de recuperarlo conectándose a su máquina desde otra y ejecutando `kbd_mode -a`.
- Por favor, examine su archivo de registro del sistema (`/var/log/messages` o similar) para encontrar motivos de su problema. Si piensa que el problema está en MySQL, también debería revisar los archivos de registro de MySQL. Consulte Sección 5.10, “Los ficheros de registro (log) de MySQL”.
- Si no cree que tenga problemas de hardware, debería intentar encontrar el programa que le está casuando problemas. Intente utilizar los programas `top`, `ps`, el Administrador de Tareas, o algún programa similar, para comprobar cual de los procesos que se están ejecutando está monopolizando la CPU o bloqueando la máquina.
- Utilice `top`, `df`, o un programa similar para comprobar si se está quedando sin memoria, espacio en disco, descriptores de archivo, o algún otro recurso crítico.
- Si el problema es algún proceso desbocado, siempre puede intentar matarlo. Si no quiere morir, probablemente exista algún error en el sistema operativo.

Si tras haber examinado el resto de posibilidades y llega a la conclusión de que el servidor o el cliente MySQL puedan estar causando el problema, es el momento de crear un informe de fallos para nuestra lista de correos o equipo de soporte. En el informe, intente dar una descripción muy detallada de como el

sistema se está comportando y qué es lo que usted cree que está sucediendo. También debería explicar por qué cree que MySQL está causando el problema. Tenga en cuenta todas los puntos de este capítulo. Explique cualquier problema de la maenra exacta como aparecen cuando usted examina su sistema. Utilice el método de “copiar y pegar” para enviar cualquier salida o mensaje de error de los programas o archivos de registro.

Intente explicar con detalle qué programa no está funcionando y los síntomas que usted ve. En el pasado, hemos recibido muchos informes de error que únicamente decían “el sistema no funciona.” Esto no nos da mucha información sobre cual pueda ser el problema.

Si un programa falla, siempre es útil saber lo siguiente:

- ¿Ha hecho el programa en cuestión un fallo de segmentación?
- ¿El programa está ocupando todo el tiempo disponible de CPU? Compruébelo con `top`. Deje el programa ejecutarse durante unos instantes, podría ser simplemente que está haciendo algunos cálculos intensivos.
- Si el servidor `mysqld` está causando problemas, ¿puede usted obtener algún tipo de respuesta de él con `mysqladmin -u root ping` o `mysqladmin -u root processlist`?
- ¿Qué dicen los programas cliente cuando intenta conectarse al servidor MySQL? (Inténtelo con `mysql`, por ejemplo.) ¿Funciona el cliente? ¿Consigue algún tipo de respuesta desde el cliente?

Al enviar un informe de fallo, usted debe seguir el borrador descrito en Sección 1.6.1.2, “Hacer preguntas y reportar bugs”.

A.2. Errores comunes al usar programas MySQL

Esta sección enumera algunos errores que los usuarios encuentran de manera frecuente cuando ejecutan programas MySQL. Aunque los problemas se muestran cuando intenta ejecutar programas cliente, las soluciones a muchos de los problems pasan por cambios en la configuración del servidor MySQL.

A.2.1. `Access denied`

Un error de `Acceso denegado` puede tener muchas causas. Frecuentemente el problema está relacionado con las cuentas MySQL a las que el servidor deja que se conecten los programas cliente. Consulte Sección 5.6.8, “Causas de errores `Access denied`”. Consulte Sección 5.6.2, “Cómo funciona el sistema de privilegios”.

A.2.2. `Can't connect to [local] MySQL server`

Un cliente MySQL en Unix puede conectarse al servidor `mysqld` de dos maneras diferentes: Utilizando un archivo socket de Unix para conectarse a través de un archivo en el sistema de ficheros (por defecto `/tmp/mysql.sock`), o utilizando TCP/IP, que se conecta a través de un número de puerto. Una conexión a través de archivo socket de Unix es más rápida que a través de TCP/IP, pero solo puede ser utilizada cuando se conecta a un servidor en la misma máquina. Se utiliza un archivo de socket Unix siempre que no se especifique un nombre de servidor o si se especifica el nombre de servidor especial `localhost`.

Si el servidor MySQL está ejecutándose en Windows 9x o Me, puede conectarse únicamente a través de TCP/IP. Si el servidor se está ejecutando sobre Windows NT, 2000, XP, o 2003 y ha sido iniciado con la opción `--enable-named-pipe`, puede también conectarse a través de named pipes si el cliente se está ejecutando en la misma máquina que el servidor. El nombre de la named pipe es por defecto `MySQL`. Si no se especifica un nombre de servidor al conectar a `mysqld`, un cliente MySQL intenta primero conectarse

a la named pipe. Si esto no funciona, se conecta al puerto TCP/IP. Usted puede forzar la utilizaciónde named pipes en windows utilizando `.` como el nombre de servidor. hostname.

El error (2002) `Can't connect to ...` normalmente significa que no hay un servidor MySQL ejecutándose en el sistema o que usted está especificando un archivo de socket Unix o número de puerto TCP/IP al intentar conectarse al servidor.

Comience por comprobar si hay un proceso llamado `mysqld` ejecutándose en el servidor. (Utilice `ps xa | grep mysqld` en Unix o el Administrador de tareas en Windows). Si no existe ese proceso, debería iniciar el servidor. Consulte Sección 2.9.2.3, “Arrancar y resolver problemas del servidor MySQL”.

Si hay un proceso `mysqld` ejecutándose, puede comprobarlo ejecutando los siguientes comandos. El número de puerto o nombre del archivo socket de Unix pueden ser diferentes en su configuración. `host_ip` representa el número de IP de la máquina donde se está ejecutando el servidor.represents the IP number 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
```

Tenga en cuenta la utilización de acentos abiertos en vez de comillas en el comando `hostname`; esto provoca que la salida de `hostname` (es decir, el nombre de máquina actual) sea sustituida en el comando `mysqladmin`. Si no tiene ningún comando `hostname` o está ejecutando sobre Windows, puede escribir manualmente el nombre de su máquina (sin acentos abiertos) tra la opción `-h`. También puede intentarlo con `-h 127.0.0.1` para conectarse con TCP/IP a la máquina local.

Aquí hay algunas razones por las que el error `Can't connect to local MySQL server` podría ocurrir:

- `mysqld` no se está ejecutando. Compruebe la lista de procesos de sus sistema operativo para asegurarse de que el proceso `mysqld` está presente.
- Usted está ejecutando un sistema que utiliza hilos tipo MIT-pthreads. Si está ejecutando un sistema que no tiene hilos antivos, `mysqld` utiliza el paquete de MIT-pthreads package. Consulte Sección 2.1.1, “Sistemas operativos que MySQL soporta”. Aún así, no todas las versiones de MIT-pthreads soportan los archivos socket de Unix. En un sistema sin soporte para archivos socket, siempre debe especificar el nombre de máquina explícitamente cuando se conecte al servidor. Intente utilizar este comando para comprobar la conexión con el servidor:

  ```
  shell> mysqladmin -h `hostname` version
  ```

- Alguien ha borrado el archivo socket de Unix que `mysqld` utiliza (`/tmp/mysql.sock` por defecto). Por ejemplo, usted podría tener un trabajo de `cron` que elimine los archivos antiguos del directorio `/tmp`. Siempre puede ejecutar `mysqladmin version` para comprobar si el archivo socket de Unix que to check whether the Unix socket file that `mysqladmin` está intentando utilizar existe realmente. La solución en este caso es cambiar el trabajo de `cron` para que no elimine `mysql.sock` o colocar el archivo socket en algún otro lugar. Consulte Sección A.4.5, “Cómo proteger o cambiar el fichero socket de MySQL `/tmp/mysql.sock`”.
- Usted ha iniciado el servidor `mysqld` con la opción `--socket=/path/to/socket`, pero ha olvidado decirle al programa cliente el nuevo nombre del archivo socket. Si cambia la ruta del socket en el servidor, también tiene que notificárselo a los programas cliente. Puede hacer esto proporcionándole al cliente la misma opción `--socket` al ejecutarlo. También debe asegurarse de que los programas

cliente tienen permiso para acceder al archivo `mysql.sock`. Para averiguar donde está almacenado el archivo, puede hacer:

```
shell> netstat -ln | grep mysql
```

Consulte Sección A.4.5, “Cómo proteger o cambiar el fichero socket de MySQL `/tmp/mysql.sock`”.

- Usted está ejecutando Linux y un hilo del servidor ha muerto (volcado de memoria). En este caso, usted debe matar el resto de hilos de `mysqld` (por ejemplo, con `kill` o con el script `mysql_zap`) antes de que pueda reiniciar el servidor MySQL. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”.
- El servidor o el programa cliente podrían no tener los privilegios de acceso adecuados para el directorio que almacena el archivo socket de Unix, o para el archivo mismo. En este caso, usted debe cambiar los privilegios del directorio o los del archivo mismo para que el servidor y los clientes puedan acceder a ellos, o reiniciar `mysqld` con una opción `--socket` que especifique un nombre de archivo de socket en un directorio donde el servidor pueda crearlo y los programas cliente puedan acceder a él.

Si usted obtiene el mensaje de error `Can't connect to MySQL server on some_host`, puede intentar los siguientes procedimientos para averiguar cual es el problema:

- Compruebe si el servidor se está ejecutando en esa máquina mediante la ejecución de `telnet some_host 3306` y presionando la tecla Enter unas cuantas veces. (3306 es el puerto por defecto de MySQL. Cambie el valor si su servidor está escuchando en un puerto diferente.) Si hay un servidor MySQL ejecutándose y escuchando al puerto, debería obtener una respuesta que incluyera el número de versión del servidor. Si obtiene un error como `telnet: Unable to connect to remote host: Connection refused`, entonces no hay ningún servidor ejecutándose en el puerto dado.
- Si el servidor está ejecutándose en la máquina local, intente utilizar `mysqladmin -h localhost variables` para conectar utilizando el archivo socket de Unix. Colmpruebe el número de puerto TCP/IP al que el servidor está configurado para escuchar (es el valor de la variable `port`.)
- Asegúrese de que su servidor `mysqld` no fue iniciado utilizando la opción `--skip-networking`. Si lo fue no puede conectarse a él utilizando TCP/IP.
- Compruebe que no hay un cortafuegos bloqueando el acceso a MySQL. Aplicaciones como ZoneAlarm o el cortafuegos personal de Windows XP podría necesitar ser configurados para permitir el acceso externo a un servidor MySQL.

A.2.3. `Client does not support authentication protocol`

Las versiones de MySQL número 4.1 y superiores utilizan un protocolo de autentificación basado en un algoritmo de hash de la clave que es incompatible con el que se utiliza en los clientes anteriores. Si actualiza su servidor a 4.1, los intentos de conectarse a él desde un cliente más viejo pueden fallar con el siguiente mensaje:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

Para resolver este problema, debería utilizar alguno de los siguientes métodos:

- Actualizar todos los programas clientes para que utilicen la librería de cliente 4.1.1 o posterior.
- Cuando se conecte al servidor con un programa cliente anterior al 4.1, utilice una cuenta que todavía mantenga una clave al estilo pre-4.1.

- Reestablezca la clave al estilo pre-4.1 para cada usuario que necesite utilizar un programa cliente anterior a la versión 4.1. Esto puede hacerse utilizando la sentencia `SET PASSWORD` y la función `OLD_PASSWORD()`:

  ```
  mysql> SET PASSWORD FOR
      -> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
  ```

  Una alternativa es utilizar `UPDATE` y `FLUSH PRIVILEGES`:

  ```
  mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
      -> WHERE Host = 'some_host' AND User = 'some_user';
  mysql> FLUSH PRIVILEGES;
  ```

  Sustituya la clave que quiera utilizar por "*newpwd*" en los ejemplos precedentes. MySQL no puede retornar la clave original, así que es necesario introducir una clave nueva.

- Indique al servidor que utilice el algoritmo de hashing de claves antiguo:

  1. Inicie `mysqld` con la opción `--old-passwords`.

  2. Asigne una clave con formato antiguo a cada cuenta que tenga su clave actualizada al formato más largo de la versión 4.1. Puede identificar estas cuentas con la siguiente consulta:

     ```
     mysql> SELECT Host, User, Password FROM mysql.user
         -> WHERE LENGTH(Password) > 16;
     ```

     Para cada registro de cuentas que se muestre en la consulta, utilice los valores de `Host` y `User` y asigne una clave utilizando la función `OLD_PASSWORD()` y `SET PASSWORD` o `UPDATE`, tal como se ha explicado previamente.

Nota: En PHP, la extensión `mysql` no soporta el nuevo protocolo de autentificación en MySQL 4.1.1 y superior. Esto es así independientemente de la versión de PHP utilizada. Si desea poder utilizar la extensión `mysql` con MySQL 4.1 seguir alguna de las indicaciones explicadas arriba para configurar MySQL con clientes antiguos. La extensión `mysqli` (que significa "MySQL mejorado" - "MySQL Improved"; nueva en PHP 5) **es** compatible con el nuevo algoritmo de hashing mejorado empleado en MySQL 4.1 y superiores, y sin ninguna configuración especial necesaria que deba hacerse para utilizar esta nueva librería cliente de MySQL para PHP. Para más información sobre la extensión `mysqli` consulte http://php.net/mysqli.

For additional background on password hashing and authentication, Consulte Sección 5.6.9, "Hashing de contraseñas en MySQL 4.1".

A.2.4. La contraseña falla cuando se introduce interactivamente

Los programas cliente de MySQL piden una contraseña cuando son invocados con la opción `--password` o `-p` sin especificar ningún valor para la contraseña:

```
shell> mysql -u user_name -p
Enter password:
```

En algunos sistemas, puede ocurrir que su contraseña funcione cuando es especificada en un archivo de opciones o en la línea de comandos, pero no cuando sea introducida interactivamente en la línea de comandos. Esto ocurre cuando la librería proveida por el sistema para leer contraseñas limite los valores de éstas a un número pequeño de caracteres (normalmente ocho). Eso es un problema con la librería del

sistema, no con MySQL. Para poder solucionarlo, cambie su contraseña de MySQL a un valor que sea de ocho o menos caracteres de longitud, o ponga su constraseña en un archivo de opciones.

A.2.5. La máquina '*host_name*' está bloqueada

Si obtiene el siguiente error, significa que `mysqld` ha recibido demasiados intentos de conexión desde la máquina `'host_name'` que han sido interrumpidos:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

El número de intentos de conexión interrumpidos se puede determinar con el valor de la variable de sistema `max_connect_errors`. Tras `max_connect_errors` intentos fallidos, `mysqld` asume que hay algo que va mal (por ejemplo, que alguien está intentando romper la seguridad del sistema), y bloquea la máquina para que no pueda intentar volver a conectarse hasta que usted ejecute el comando `mysqladmin flush-hosts` o introduzca la sentencia `FLUSH HOSTS`. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

Por defecto, `mysqld` bloquea una máquina tras 10 errores de conexión. Puede ajustar el valor iniciando el servidor así:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

Si usted obtiene este mensaje de error para una máquina concreta, debería primero averiguar que no hay ningún problema con las conexiones TCP/IP desde esa máquina. Si está teniendo problemas de red, no hace ningún bien el incrementar el valor de la variable `max_connect_errors`.

A.2.6. Demasiadas conexiones

Si obtiene un error `Too many connections` cuando intenta conectarse al servidor `mysqld`, significa que todas las conexiones disponibles están siendo utilizadas por otros clientes.

El número de conexiones permitidas está controlado por la variable de sistema `max_connections`. Su valor por defecto es 100. Si necesita soportar más conexiones, debería reiniciar `mysqld` con un valor más grande de esta variable.

`mysqld` realmente permite conectarse a `max_connections+1` clientes. La conexión extra esta reservada para ser utilizada por cuentas que tienen el privilegio `SUPER`. Otorgando el privilegio `SUPER` a los administradores y no a usuarios normales (que no deberían necesitarlo), un administrador puede conectarse al servidor y utilizar `SHOW PROCESSLIST` para diagnosticar problemas aún cuando el máximo número de clientes sin privilegios estén conectados. Consulte Sección 13.5.4.16, “Sintaxis de `SHOW PROCESSLIST`”.

El número máximo de conexiones que MySQL puede soportar depende de la calidad de la librería de hilos de una plataforma dada. Linux o Solaris deberían ser capaces de soportar 500-1000 conexiones simultáneas, dependiendo de cuanta RAM tenga y que estén haciendo los clientes. Los binarios estáticos de Linux proveidos por MySQL AB pueden soportar hasta 4000 conexiones.

A.2.7. Out of memory

Si usted ejecuta una consulta utilizando el programa cliente `mysql` y recibe un error como el siguiente, significa que `mysql` no tiene suficiente memoria para almacenar el resultado completo de la consulta:

```
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
```

Para remediarlo, primero compruebe si su consulta es correcta. ¿Es razonable que devuelva tantas filas? Si no, corríjala y inténtelo de nuevo. Si no es así, puede invocar `mysql` con la opción `--quick`. Esto provoca que utilice la función `mysql_use_result()` de la API C para obtener el resultado, lo que hace que haya menos carga en el cliente (pero más en el servidor).

A.2.8. MySQL se ha apagado

Esta sección también explica el error relacionado `Lost connection to server during query`.

La razón más común para el error `MySQL server has gone away` es que el servidor ha agotado el tiempo de espera y ha cerrado la conexión. En este caso, normalmente obtendrá uno de los siguientes códigos de error (dependiendo del sistema operativo):

Código de error	Descripción
`CR_SERVER_GONE_ERROR`	El cliente no pudo enviar una consulta al servidor.
`CR_SERVER_LOST`	El cliente no obtuvo ningún error al escribir al servidor pero tampoco obtuvo una respuesta completa (o ninguna respuesta) a la pregunta.

Por defecto, el servidor cierra la conexión tras ocho horas si no pasa nada. Puede cambiar el límite de tiempo estableciendo la variable `wait_timeout` cuadno inicie `mysqld` Consulte Sección 5.3.3, “Variables de sistema del servidor”.

Si usted tiene un script, tiene que ejecutar la consulta de nuevo para que el cliente haga una reconexión automática. Esto da por hecho que tiene la reconexión automática activada en el cliente (que es la opción por defecto en el cliente de línea de comandos `mysql`).

Otras razones comunes por las que puede aparecer el error `MySQL server has gone away` son:

- Usted (o el administrador de la base de datos) ha matado el hilo que se estaba ejecutando con una sentencia `KILL` o el comando `mysqladmin kill`.
- Usted ha intentado ejecutar una sentencia tras cerrar la conexión con el servidor. Esto es síntoma de un error lógico en la aplicación que debería ser corregido.
- Se ha agotado el tiempo de espera de una conexión TCP/IP desde el lado cliente. Esto puede ocurrir si usted ha estado utilizando los comandos: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` o `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. En este caso, aumentar el tiempo de espera puede ayudar a resolver el problema.
- Se ha agotado el tiempo de espera en el lado del servidor, y el cliente no tiene activada la opción de reconexión automática (la opción `reconnect` en la estructura `MYSQL` es igual a 0).
- Usted está utilizando un cliente windows y el servidor ha cortado la conexión (probablemente porque `wait_timeout` ha expirado) antes de que el comando fuese ejecutado.

  El problema en windows es que en algunos casos MySQL no obtiene un error desde el SO cuando escribe a la conexión TCP/IP desde el servidor, sino que obtiene el error cuando intenta leer la respuesta desde la conexión.

  En este caso, aunque el flag `reconnect` en la estructura `MYSQL` sea igual a 1, MySQL no reconecta y vuelve a ejecutar la sentencia, ya que no sabe si el servidor recibió la sentencia original o no.

  La solución a esto es o hacer un `mysql_ping` en la conexión si ha pasado mucho tiempo desde la última sentencia (esto es lo que `MyODBC` hace) o establecer un `wait_timeout` en el servidor `mysqld` tan alto que en la práctica, nunca llegue a sobrepasarse.

- También puede obtener estos errores si envía una consulta al servidor que sea incorrecta o demasiado grande. Si `mysqld` recibe un paquete que es demasiado grande o fuera de lugar, asume que ha habido algún error con el cliente y cierra la conexión. Si necesita realizar grances consultas (por ejemplo, si está trabajando con columnas `BLOB` muy grandes), debería incrementar el límite de las consultas estableciendo la variable de servidor `max_allowed_packet`, que tiene un valor por defecto de 1MB. También podría necesitar incrementar el tamaño máximo de paquete en el lado cliente. Puede encontrar más información para establecer el tamaño de paquete en Sección A.2.9, “`Packet too large`”.
- También puede perder la conexión si envía un paquete de más de 16MB y su cliente es anterior a la versión 4.0.8 y su servidor posterior a 4.0.8, o viceversa.
- También puede ver el error `MySQL server has gone away` si MySQL se inicia con la opción `--skip-networking`.
- Ha encontrado un error por el que el servidor cayó mientas ejecutaba una sentencia.

Puede comprobar si el servidor MySQL cayó y se reinició ejecutando `mysqladmin version` y examinando el tiempo de ejecución del servidor (uptime). Si la conexión del cliente se cortó debido a que `mysqld` falló y se reninicó, debería intentar encontrar la razón del fallo. Comience por comprobar si ejecutando la misma sentencia el servidor cae de nuevo. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”.

Puede obtener más información sobre las conexiones perdidas iniciando mysqld con la opción `--log-warnings=2`. Esto registra algunos de los errores de desconexión en el archivo `hostname.err`. Consulte Sección 5.10.1, “El registro de errroes (Error Log)”.

Si quiere crear un informe de error en relación a este problema, asegúrese de incluir la siguiente información:

- Indique si el servidor MySQL murió. Puede enecontrar esta información en el registro de errores del servidor. Consulte Sección A.4.2, “Qué hacer si MySQL sigue fallando (crashing)”.
- Si una consulta específica mata a `mysqld` y las tablas implicadas habían sido comprobadas con `CHECK TABLE` antes de ejecutar la consulta, ¿puede proporcionar una prueba que permita reproducir el caso? Consulte Sección D.1.6, “Crear un caso de prueba tras haber encontrado una tabla corrupta”.
- ¿Cual es el valor de la variable de sistema `wait_timeout` en el servidor MySQL? (`mysqladmin variables` le da el valor de esta variable.)
- Ha intentado ejecutar `mysqld` con la opción `--log` para determinar si la consulta problemática aparece en el registro?

Consulte también Sección A.2.10, “Errores de comunicación y conexiones abortadas”.

Consulte Sección 1.6.1.2, “Hacer preguntas y reportar bugs”.

A.2.9. `Packet too large`

Un paquete de comunicación es una única sentencia SQL enviada al servidor MySQL o una única fila que es enviada al cliente.

En MySQL 3.23, el páquete más grande posible es de 16MB, debido a los límites del protocolo cliente/servidor. En MySQL 4.0.1 y superiores, el límite es de 1GB.

Cuando un cliente MySQL o el servidor `mysqld` recibe un paquete más grande de `max_allowed_packet` bytes, este devuelve un error `Packet too large` y cierra la conexión. Con algunos clientes, también podría obtener un error `Lost connection to MySQL server during query` si el paquete de comunicación es demasiado grande.

Tanto el cliente como el servidor tienen su propia variable `max_allowed_packet` así que si quiere gestionar paquetes grandes, debe aumentar esta variable tanto en el cliente como en el servidor.

Si está utilizando el programa cliente `mysql`, el valor por defecto de la variable `max_allowed_packet` es de 16MB. Este es también el máximo valor permitido anteriormente a la versión MySQL 4.0. Para establecer un valor mayor desde 4.0 en adelante, inicie `mysql` de la siguiente manera:

```
mysql> mysql --max_allowed_packet=32M
```

Esto establece el tamaño del paquete en 32MB.

El valor por defecto de la variable `max_allowed_packet` en el servidor es de 1MB. Pue incrementar esto si el servidor necesita gestionar consultas grandes (por ejemplo, si está trabajando con columnas `BLOB` grandes). Por ejemplo, para establecer la variable a 16MB, inicie el servidor así:

```
mysql> mysqld --max_allowed_packet=16M
```

Para versiones previas a MySQL 4.0, utilice esta sintaxis:

```
mysql> mysqld --set-variable=max_allowed_packet=16M
```

También puede utilizar un archivo de opciones para establecer `max_allowed_packet`. Por ejemplo, para establecer el valor para el servidor en 16MB, añada las siguientes líneas en su archivo de opciones:

```
[mysqld]
max_allowed_packet=16M
```

Para versiones previas a MySQL 4.0, utilice esta sintaxis:

```
[mysqld]
set-variable = max_allowed_packet=16M
```

Es seguro incrementar el valor de esta variable porque la memoria extra tan solo es utilizada cuando se necesite. Por ejemplo, `mysqld` solo ocupa más memoria cuando usted ejecuta una consulta grande o cuando `mysqld` debe retornar una fila de resultados grande. El valor por defecto pequeño en la variable, es una precaución para atrapar paquetes incorrectos entre el cliente y el servidor, y también para asegurarse de que usted no se queda sin memoria al utilizar paquetes grandes de manera accidental.

También puede sufrir problemas extraños con paquetes grandes si está utilizando valores `BLOB` grandes pero no ha dado a `mysqld` acceso a suficiente memoria para gestionar la consulta. Si sospecha que este es el caso, intente añadir `ulimit -d 256000` al principio del script `mysqld_safe` y reinicie `mysqld`.

A.2.10. Errores de comunicación y conexiones abortadas

El registro de errores del servidor puede ser una fuente de información útil sobre problemas de conexión. Consulte Sección 5.10.1, “El registro de errroes (Error Log)”. A partir de MySQl 3.23.40, si quiere iniciar el servidor con la opción `--warnings` (o `--log-warnings` desde MySQl 4.0.3 en adelante), usted podría encontrar mensajes como este en su registro de errores:

```
010301 14:38:23  Aborted connection 854 to db: 'users' user: 'josh'
```

Si un mensaje `Aborted connections` aparece en el registro de errores, la causa puede ser alguna de las siguientes:

- El programa cliente no llamó a `mysql_close()` antes de salir.

- El cliente ha estado inactivo más de `wait_timeout` o `interactive_timeout` segundos, sin enviar ninguna petición al servidor. Consulte Sección 5.3.3, “Variables de sistema del servidor”.
- El programa cliente abortó de manera abrupta en mitad de una transferencia de datos.

Cuando alguna de estas cosas pasa, el servidor incrementa la variable de estado `Aborted_clients`.

El servidor incrementa la variable de estado `Aborted_connects` cuando una de las siguientes cosas ocurren:

- Un cliente no tiene privilegios para conectar a una base de datos.
- Un cliente utiliza una contraseña incorrecta.
- Un paquete de conexión no contiene la información correcta.
- Se tarda más de `connect_timeout` en obtener un paqueta de conexión. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

Si este tipo de cosas pasan, ¡podría ser indicativo de que alguien está intentando entrar en su servidor!

Otros motivos para los problemas con clientes que abortan o conexiones interrumpidas:

- Utilización del protocolo Ethernet con Linux, tanto en half como en full duplex. Muchos drivers de Ethernet en Linux tienen este error. Debería comprobar si su driver contiene este error transfiriendo un archivo enorme via FTP entre el cliente y el servidor. Si la transferencia entra en un modo de ejecución-pausa-ejecución-pausa, usted está experimentando el síndrome duplex de Linux. La única solución es cambiar el modo duplex tanto de su tarjeta de red como de su concentrador o switch tanto a full como a half duplex, y comprobar los resultados para determinar la mejor configuración.
- Algunos problemas con la librería de hilos de ejecución que causa interrupciones en las lecturas.
- TCP/IP mal configurado.
- Redes, concentradores, switches o cables defectuosos. Esto solo puede ser diagnosticado mediante el reemplazo de hardware.
- El valor de la variable `max_allowed_packet` es demasiado pequeño o las consultas requieren más memoria de la que tiene disponible para `mysqld`. Consulte Sección A.2.9, “`Packet too large`”.

Consulte también Sección A.2.8, “`MySQL se ha apagado`”.

A.2.11. `The table is full`

Hay varias maneras en que puede producirse un error de tabla llena:

- Cuando utiliza un servidor MySQL anterior a la versión 3.23 y una tabla temporal en memoria se hace más grande de `tmp_table_size` bytes. Para evitar este problema puede utilizar la opción `-O tmp_table_size=#` para hacer que `mysqld` incremente el valor temporal del tamaño de tablas, o utilizar la opción `SQL_BIG_TABLES` antes de ejecutar la consulta problemática. Consulte Sección 13.5.3, “Sintaxis de `SET`”.

  También puede iniciar `mysqld` con la opción `--big-tables`. Esto es exactamente lo mismo que utilizar la opción `SQL_BIG_TABLES` para todas las consultas.

  A partir de MySQL 3.23, este problema no debería ocurrir. Si una tabla en memoria se hace más grande que `tmp_table_size`, el servidor automáticamente la convierte en una tabla de disco `MyISAM`.

- Está utilizando tablas `InnoDB` y se quedó sin espacio en el espacio de tablas `InnoDB`. En este caso, la solución es aumentar el espacio de tablas `InnoDB`. Consulte Sección 15.7, “Añadir y suprimir registros y ficheros de datos `InnoDB`”.
- Está utilizando tablas `ISAM` o `MyISAM` en un sistema operativo que tan solo soporta hasta 2GB de tamaño de archivo, habiendo superado ya este límite para el archivo de datos o de índices.
- Está utilizando una tabla `MyISAM`, y el espacio requerido por la tabla excede a lo que permite el tamaño de puntero interno. Si no especifica la opción de tabla `MAX_ROWS` cuando cree una tabla, MySQL utiliza la variable de sistema `myisam_data_pointer_size`. Desde MySQL 5.0.6 en adelante, el valor por defecto es de 6 bytes, que es suficiente para permitir 65536 TB de datos. Antes de MySQL 5.0.6, el valor por defecto es de 4 bytes, que solo es suficiente para permitir 4 GB de datos. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

  Puede comprobar los tamaños máximos de datos e índices mediante esta consulta:

  ```
  SHOW TABLE STATUS FROM database LIKE 'nombre_tabla';
  ```

  También puede utilizar `myisamchk -dv /ruta/a/archivo-indice-de-tabla`.

  Si el tamaño del puntero es demasiado pequeño, puede corregir el problema utilizando `ALTER TABLE`:

  ```
  ALTER TABLE nombre_tabla MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
  ```

  Tiene que especificar `AVG_ROW_LENGTH` solo para tablas con columnas `BLOB` o `TEXT`; en este caso, MySQL no puede optimizar el espacio requerido basándose únicamente en el número de filas.

A.2.12. Can't create/write to file

Si obtiene un error del siguiente tipo en algunas consultas, significa que MySQL no puede crear un archivo temporal para el conjunto de resultados en el directorio temporal:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

El error anterior es un mensaje típico para Windows; el mensaje de Unix es similar.

Una solución es iniciar `mysqld` con la opción `--tmpdir` o añadir la opción a la sección `[mysqld]` de su archivo de opciones. Por ejemplo, para especificar el directorio `C:\temp`, utilice estas líneas:

```
[mysqld]
tmpdir=C:/temp
```

El directorio `C:\temp` debe existir y tener suficiente espacio para que el servidor MySQL escriba en él. Consulte Sección 4.3.2, “Usar ficheros de opciones”.

Otra causa de este error pueden ser temas de permisos. Asegúrese de que el servidor MySQL puede escribir en el directorio `tmpdir`.

Compruebe también el código de error que obtiene con `perror`. Una razón por la que el servidor puede no escribir en una tabla es que el sistema de archivos esté lleno:

```
shell> perror 28
Error code  28:  No space left on device
```

A.2.13. Commands out of sync

Si botiene un error `Commands out of sync; you can't run this command now` en su código de cliente, está llamando a las funciones cliente en orden erróneo.

Esto puede pasar, por ejemplo, si está utilizando `mysql_use_result()` e intenta ejecutar una nueva consulta antes de llamar a `mysql_free_result()`. También puede pasar si intenta ejecutar dos consultas que retornan datos sin llamar a `mysql_use_result()` o `mysql_store_result()` entre ellas.

A.2.14. Ignoring user

Si obtiene el siguiente error, significa que cuando se inició `mysqld` o cuando recargó las tablas de privilegios, encontró una cuenta en la tabla `user` que tenía una contraseña no válida.

`Found wrong password for user 'un_usuario'@'un_host'; ignoring user`

Como resultado, la cuenta es sencillamente ignorada por el sistema de privilegios.

La siguiente lista indica posibles causas de este problema, y algunas soluciones:

- Quizá está ejecutando una versión nueva de `mysqld` con una tabla `user` vieja. Puede comprobar esto ejecutando el comando `mysqlshow mysql user` para ver si la columna `Password` es más corta de 16 caracteres. Si es así, puede corregir esta condición ejecutando el script `scripts/add_long_password`.
- La cuenta tiene una contraseña antigua (de ocho caracteres de longitud) y no inició el servidor `mysqld` con la opción `--old-protocol`. Actualice la cuenta en la tabla `user` para que tenga una nueva contraseña, o reinicie `mysqld` con la opción `--old-protocol`.
- Ha especificado una contraseña en la tabla `user` sin utilizar la función `PASSWORD()`. Utilice `mysql` para actualizar la cuenta en la tabla `user` con una nueva contraseña asegurándose de utilizar la función `PASSWORD()`:

```
mysql> UPDATE user SET Password=PASSWORD('nuevopwd')
    -> WHERE User='usuario' AND Host='maquina';
```

A.2.15. Table 'nombre_de_tabla' doesn't exist

If you get either of the following errors, it usually means that no table exists in the current database with the given name:

```
Table 'nombre_de_tabla' doesn't exist
Can't find file: 'nombre_de_tabla' (errno: 2)
```

En algunos casos, puede ser que la tabla no exista porque usted se esté refiriendo a ella incorrectamente:

- Debido a que MySQL utiliza directorios y archivos para almacenar bases de datos y tablas, los nombres de bases de datos y tablas son sensibles a las letras mayúsculas si se encuentran en un sistema de archivos que lo sea.
- Aún en los sistemas de archivo que no son sensibles a la diferencia entre minúscula y mayúscula, como en Windows, todas las referencias a una tabla dada deben utilizar el mismo tipo de letra.

Puede comprobar las tablas que hay en la base de datos actual con `SHOW TABLES`. Consulte Sección 13.5.4, “Sintaxis de SHOW”.

A.2.16. Can't initialize character set

Si tiene problemas con los juegos de caracteres, puede obtener un error como este:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

Este error puede tener una de las siguientes causas:

- El juego de caracteres es un juego multi-byte, y no tiene soporte para ese juego de caracteres en el cliente. En este caso, necesita recompilar el cliente ejecutando `configure` con la opción `--with-charset=charset_name` o `--with-extra-charsets=charset_name`. Consulte Sección 2.8.2, "Opciones típicas de `configure`".

  Todos los binarios estándar MySQL se compilan con `--with-extra-character-sets=complex`, que activa el soporte para los juegos de carácter multi-byte. Consulte Sección 5.9.1, "El conjunto de caracteres utilizado para datos y ordenación".

- El juego de caracteres es un juego simple que no está compilado en `mysqld`, y el archivo de definición de juegos de caracteres no está en el lugar que el cliente espera encontrarlo.

  En este caso, necesita utilizar uno de los siguientes métodos para resolver el problema:

  - Recompile el cliente con soporte para el juego de caracteres. Consulte Sección 2.8.2, "Opciones típicas de `configure`".
  - Especifique al cliente el directorio donde la definición del juego de caraceres tiene sus archivos. En muchos clientes puede hacerlo con la opción `--character-sets-dir`.
  - Copie los archivos de definición de caracteres a la ruta donde el cliente espera que estén.

A.2.17. No se encontró el fichero

Si obtiene `ERROR '...' not found (errno: 23)`, `Can't open file: ... (errno: 24)`, o cualquier otro error con `errno 23` o `errno 24` de MySQL, significa que no tiene reservados suficientes descriptores de archivo para el servidor MySQL. Puede utilizar la utilidad `perror` para obtener una descripción de lo que el número de error significa:

```
shell> perror 23
Error code  23:  File table overflow
shell> perror 24
Error code  24:  Too many open files
shell> perror 11
Error code  11:  Resource temporarily unavailable
```

El problema aquí es que `mysqld` está intentando mantener abiertos demasiados archivos de manera simultánea. Puede decirle a `mysqld` que no abra tantos archivos a la vez, o incrementar el número de descriptores disponibles para `mysqld`.

Para decirle a `mysqld` que mantenga abiertos menos archivos de manera simultánea, puede hacer la cache de la tabla mas pequeña reduciendo el valor de la variable de sistema `table_cache` (el valor por defecto es 64). Reducir el valor de `max_connections` también reduce el número de archivos abiertos (el valor por defecto es 100).

Para cambiar el número de descriptores de archivo disponibles para `mysqld`, puede utilizar la opción `--open-files-limit` para `mysqld_safe` o (desde MySQL 3.23.30) establecer la variable de sistema `open_files_limit`. Consulte Sección 5.3.3, "Variables de sistema del servidor". La manera más fácil de establecer estos valores es añadir una opción a su archivo de opciones. Consulte Sección 4.3.2, "Usar ficheros de opciones". Si usted dispone de una versión antigua de `mysqld` que no soporta el

establecimiento del límite de archivos abiertos, puede editar el script `mysqld_safe`. Hay una línea comentada en el script, `ulimit -n 256`. Puede eliminar el carácter '`#`' para descomentar esta línea, y cambiar el número `256` para establecer el número de descriptores de archivos que serán puestos a disposición de `mysqld`.

`--open-files-limit` y `ulimit` pueden incrementar el número de descriptores de archivo, pero sólo hasta el límite impuesto por el sistema operativo. Hay también un límite impuesto que puede ser sobreseido sólo si inicia `mysqld_safe` o `mysqld` como `root` (simplemente recuerde que necesita iniciar el servidor con la opción `--user` en este caso para que no continue ejecutándose como `root` tras el inicio). Si usted necesita incrementar el límite del sistema operativo sobre el número de descriptores de archivo disponibles para cada proceso, consulte la documentación de su sistema operativo.

Nota: ¡Si usted ejecuta el shell `tcsh`, `ulimit` no funciona! `tcsh` también informa de valores incorrectos cuando le interroga por los límites actuales. En este caso, debería iniciar `mysqld_safe` utilizando `sh`.

A.3. Problemas relacionados con la instalación

A.3.1. Problemas al enlazar a la biblioteca de clientes MySQL

Cuando usted enlaza un programa para utilizar la librería cliente de MySQL, usted podría obtener errores de referencias no definidas para los símbolos que comiencen con `mysql_`, tal como los que se muestran aquí:

```
/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+0x57): 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'
```

Debería ser capaz de resolver este problema añadiento `-Ldir_path -lmysqlclient` al final de su comando de enlace, donde `dir_path` representa la ruta de el directorio donde la librería cliente está situada. Para determinar el directorio correcto, utilice este comando:

```
shell> mysql_config --libs
```

La salida de `mysql_config` podría indicar otras librerías que deberían ser especificadas en el comando de enlace también.

Si obtiene errores de `undefined reference` en las funciones `uncompress` o `compress`, intente añadir `-lz` al final de su comando de enlace e intente de nuevo.

Si usted obtiene errores `undefined reference` para una función que no deberían ocurrir en su sistema, tal como `connect`, compruebe la página de la función en el manual y determine qué librerías debería añadir para el comando de enlace.

Podría obtener errores `undefined reference` como el siguiente por las funciones que no existen en su sistema:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

Esto normalmente significa que su librería cliente MySQL fue compilada en un sistema que no es compatible al 100% con el suyo. En este caso, debería descargar la última distribución de código fuente de MySQL y compilar usted mismo MySQL. Consulte Sección 2.8, “Instalación de MySQL usando una distribución de código fuente”.

Podría obtener errores de referencia en tiempo de ejecución cuando intenta ejecutar un programa MySQL. Si estos errores especifican símbolos que comienzan con `mysql_` o indican que la librería cliente `mysqlclient` no puede encontrarse, significa que su sistema no puede encontrar la librería compartida `libmysqlclient.so`. La corrección para esto es decirle a su sistema que busque las librerías compartidas donde estén situadas. Utilice aquel de los siguientes métodos que sea apropiedo para su sistema:

- Añada la ruta al direcotorio donde se encuentre `libmysqlclient.so` a la variable de entorno `LD_LIBRARY_PATH`.
- Añada la ruta al directorio donde `libmysqlclient.so` se encuentra a la variable de entorno `LD_LIBRARY`.
- Copie el archivo `libmysqlclient.so` a algún directorio de los que su sistema busca, como por ejemplo `/lib`, y actualice la información de la librería compartido ejecutando `ldconfig`.

Otra manera de resolver este problema es enlazando su programa de manera estática con la opción `-static`, o eleminando las librería dinámicas MySQL antes de enlazar su código. Antes de intentar el segundo método, usted debería asegurarse de que ningún otro programa está utilizando las librerías dinámicas.

A.3.2. Cómo ejecutar MySQL como usuario normal

En Windows, usted puede ejecutar el servidor como un servicio de Windows utilizando cuentas de usuario normales a partir de MySQL 4.0.17 and 4.1.2. (Versiones más antiguas de MySQL requieren que usted tenga derechos de administrador. Eso fue un error introducido en MySQL 3.23.54.)

En Unix, el servidor MySQL `mysqld` puede iniciarse y ser ejecutado por cualquier usuario. Aún así, usted debería evitar ejecutar el servidor como el usuario Unix `root`, por razones de seguridad. Para ejecutar `mysqld` como un usuario normal Unix sin privilegios *`user_name`*, debe hacer lo siguiente:

1. Pare el servidor si se está ejecutando (utilice el comando `mysqladmin shutdown`).
2. Cambie los directorios de la base de datos y archivos, de manera que el usuario *`user_name`* tenga privilegios para leer y escribir archivos en ellos (podría necesitar hacerlo como usuario `root` en Unix):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   Si no hace esto, el servidor no es capaz de acceder a bases de datos o tablas cuando se ejecuta como *`user_name`*.

   Si los directorios o archivos en el directorio de datos de MySQL son enlaces simbólicos, necesitará también seguir estos enlaces y cambiar los directorios y archivos a los que apuntan. Podría ser que `chown -R` no siguiera los enlaces por usted.
3. Inicie el servidor como el usuario *`user_name`*. Si usted está utilizando MySQL 3.22 o posterior, otra alternativa es iniciar `mysqld` como usuario `root` de Unix y utilizar la opción `--user=`*`user_name`*. `mysqld` se inicia, y entonces cambia la ejecución al usuario Unix *`user_name`* antes de aceptar ninguna conexión.
4. Para iniciar al servidor como el usuario dado automáticamente al inicio del sistema, especifique el nombre de usuario añadiendo una opción `user` a el grupo `[mysqld]` del archivo de opciones `/etc/my.cnf` o el archivo de opciones `my.cnf` en el directorio de datos del servidor. Por ejemplo:

   ```
   [mysqld]
   user=user_name
   ```

Si su máquina Unix no es segura, debería asignar contraseñas a las cuentas `root` de MySQL en las tablas de privilegios. De otra manera cualquier usuario con una cuenta de entrada a su máquina podría ejecutar el cliente `mysql` con una opción `--user=root` y realizar cualquier operación. (Es una buena idea asignar contraseñas a las cuentas MySQL en cualquier caso, pero especialmente si existen otras cuentas de entrada en la máquina del servidor.) Consulte Sección 2.9, “Puesta en marcha y comprobación después de la instalación”.

A.3.3. Problemas con permisos de archivos

Si usted tiene problemas con los permisos de archivos, la variable de entorno `UMASK` puede estar especificada de manera incorrecta al inicio de `mysqld`. Por ejemplo, MySQL puede devolver el siguiente mensaje de error al crear una tabla:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

El valor por defecto de `UMASK` es `0660`. Ustee puede cambiar este comportamiento iniciando `mysqld_safe` de la siguiente manera:

```
shell> UMASK=384  # = 600 en octal
shell> export UMASK
shell> mysqld_safe &
```

Por defecto, MySQL crea las bases de datos y directorios `RAID` con un valor de permiso de acceso de `0700`. Usted puede modificar este comportamiento estableciendo la variable `UMASK_DIR`. Si usted establece su valor, los nuevos directorios serán creados con los valores combinados de `UMASK` y `UMASK_DIR`. Por ejemplo, si usted quiere dar acceso de grupo a todos los directorios nuevos, puede hacer eso:

```
shell> UMASK_DIR=504  # = 770 en octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

En MySQL 3.23.25 y superiores, MySQL asume que el valor de `UMASK` y `UMASK_DIR` está en octal si comienza por cero.

Consulte Apéndice E, *Variables de entorno*.

A.4. Cuestiones relacionadas con la administración

A.4.1. Cómo reiniciar la contraseña de root

Si nunca ha establecido una contraseña para el usuario `root` de MySQL, el servidor no requiere ninguna contraseña para conectar como `root`. De todas formas, se recomienda establecer una contraseña para cada cuenta. Consulte Sección 5.5.1, “Guía de seguridad general”.

Si usted había establecido previamente una contraseña para el usuario `root`, pero ha olvidado cual era, puede establecer una nueva contraseña. El siguiente procedimiento es para sistemas Windows. El procedimiento para sistemas Unix está descrito más adelante en esta sección.

El proceso bajo Windows: The procedure under Windows:

1. Entre en su sistema como Administrador.
2. Pare el servidor MySQL si se está ejecutando. Para servidores que se estén ejecutando como servicio de Windows, vaya al Gestor de Servicios:

```
Menú Inicio -> Panel de Control -> Herramientas administrativas -> Servicios
```

Después encuentre en la lista el servicio MySQL, y parelo.

Si su servidor no está ejecutándose como servicio, podría necesitar utilizar el Gestor de tareas para forzarlo a parar.

3. Cree un archivo de texto e introduzca el siguiente comando en él, en una única línea:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MiNuevaContraseña');
```

Guarde el archivo con cualquier nombre. Para este ejemplo, el nombre del archivo será `C:\mysql-init.txt`.

4. Abra una ventana de comandos para obtener una consola de comandos DOS:

```
Menú Inicio -> Ejecutar -> cmd
```

5. Asumiremos que usted tiene instalado MySQL en `C:\mysql`. Si lo intaló en algún otro lugar, ajuste los siguientes comandos de manera adecuada.

En la línea de comandos DOS, ejecute esta orden:

```
C:\> C:\mysql\bin\mysqld-nt --init-file=C:\mysql-init.txt
```

Los contenidos del archivo nombrado por la opción `--init-file` son ejecutados en el inicio del servidor, cambiando la contraseña de `root`. Cuando el servidor se haya iniciado correctamente, debería borrar el archivo `C:\mysql-init.txt`.

Los usuarios de MySQL 4.1 y superiores que instalen MySQL utilizando el instalador de mySQL, pueden necesitar especificar una opción `--defaults-file`:

```
C:\> C:\Archivos de Programa\MySQL\MySQL Server 5.0\bin\mysqld-nt.exe
         --defaults-file="C:\Archivos de Programa\MySQL\MySQL Server 5.0\my.ini"
         --init-file=C:\mysql-init.txt
```

La configuración apropiada de `--defaults-file` puede encontrarse utilizando el Gestor de Servicios:

```
Menú Inicio -> Panel de Control -> Herramientas Administrativas -> Servicios
```

Encuentre el servicio MySQL en la lista, pulse con el botón derecho del ratón, y escoja la opción `Propiedades`. El campo `Ruta al Ejecutable` contiene la configuración de `--defaults-file`.

6. Pare el servidor MySQL, y reinícielo en modo normal de nuevo. Si ejecuta el servidor como servicio, inícielo desde la ventana de servicios de Windows. Si ejecuta el servidor manualmente, utilice el comando que normalmente use.

7. Debería poder conectar utilizando la nueva contraseña.

En un entorno Unix, el procedimiento para restablecer la contraseña `root` es el siguiente:

1. Entre en sus sitema como usuario Unix `root` o bien como el mismo usuario que ejecuta el servidor `mysqld`.

2. Localice el archivo `.pid` que contiene el ID de proceso del servidor. La localización exacta y el nombre de este archivo depende de su distribución, nombre de máquina, y configuración. Lugares comunes son `/var/lib/mysql/`, `/var/run/mysqld/`, y `/usr/local/mysql/data/`. Generalmente, el archivo tiene una extensión `.pid` y comienza con `mysqld` o el nombre de su máquina.

   Puede parar el servidor MySQL enviando un comando `kill` (no `kill -9`) a el proceso `mysqld` utilizando la ruta del archivo `.pid` en el siguiente comando:

   ```
   shell> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Nótese el uso de acentos abiertos en vez de comillas simples con el comando `cat`; estos causan que la salida de `cat` sea sustituida en el comando `kill`.

3. Cree un archivo de texto e introduzca el siguiente comando en una única línea:

   ```
   SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MiNuevaContraseña');
   ```

   Guarde el archivo con cualquier nombre. Para este ejemplo, el archivo tendrá el nombre `~/mysql-init`.

4. Reinicie el servidor MySQL con la opción especial `--init-file=~/mysql-init`:

   ```
   shell> mysqld_safe --init-file=~/mysql-init &
   ```

   Los contenidos del archivo son ejecutados al inicio del servidor, cambiando la contraseña de root. Después de que el servidor se haya iniciado con éxito, debería borrar `~/mysql-init`.

5. Debería poder conectar utilizando la nueva contraseña.

Una alternativa, en cualquier plataforma, es establecer la nueva contraseña desde el cliente `mysql` (pero esta manera es menos segura):

1. Pare `mysqld` y reinícielo con la opción `--skip-grant-tables --user=root` (Los usuarios de Windows deben omitir la parte de `--user=root`).

2. Conecte al servidor `mysqld` con este comando:

   ```
   shell> mysql -u root
   ```

3. Ejecute las siguientes sentencias en el cliente `mysql`:

   ```
   mysql> UPDATE mysql.user SET Password=PASSWORD('nuevacontraseña')
       ->                   WHERE User='root';
   mysql> FLUSH PRIVILEGES;
   ```

   Reemplace “*nuevacontraseña*” con la contraseña de `root` real que quiere utilizar.

4. Debería poder conectar utilizando la nueva contraseña.

A.4.2. Qué hacer si MySQL sigue fallando (crashing)

Cada versión de MySQL es probada en muchas plataformas antes de ser lanzada al público. Esto no significa que no haya ningún fallo en MySQL, pero si hay alguno, deberían ser muy pocos y difíciles de encontrar. Si tiene un problema, siempre ayuda si usted intenta encontrar qué es lo que hace fallar su sistema exactamente, porque tendrá una probabilidad mucho mayor de que su problema sea solucionado rápidamente.

Primero, debería intentar averiguar si el problema es que el servidor `mysqld` muere o el problema tiene que ver con su cliente. Puede comprobar cuando tiempo se ha estado ejecutando ininterrumpidamente su servidor ejecutando `mysqladmin version`. Sí `mysqld` ha caido y se ha reiniciado, podría encontrar la razón mirando en el registro de errores del servidor. Consulte Sección 5.10.1, “El registro de errroes (Error Log)”.

En algunos sistemas, puede encontrar en volcado de pila en el registro de errores del momento en que el servidor `mysqld` murió, y que puede analizar con el programa `resolve_stack_dump`. Consulte Sección D.1.4, “Usar stack trace”. Nótese que los valores de las variables escritos en el registro de errores pueden no ser siempre correctos al 100%.

Muchas caidas del servidor son causadas por archivos de datos o índices corruptos. MySQL actualiza los archivos en el disco con la llamada de sistema `write()` después de cada sentencia SQL y antes de que el cliente sea notificado del resultado. (Esto no es así si está ejecutando con `--delay-key-write`, en cuyo caso los archivos de datos son escritos pero no los de índices.) Esto significa que los contenidos de los archivos de datos están seguros aunque `mysqld` caiga, porque el sistema operativo se asegura de que los datos no volcados sean escritos al disco. Puede forzar a que MySQL escriba todo a disco tras cada sentencia SQL iniciando `mysqld` con la opción `--flush`.

Esto significa que normalmente usted no debería obtener tablas corruptas a menos que una de las siguientes cosas ocurra:

- El servidor MySQL o la máquina han sido cerrados en medio de una actualización.
- Usted ha encontrado un fallo en `mysqld` que ha causado que caiga en el medio de una actualización.
- Algún programa externo está manipulando los archivos de datos o índices a la vez que `mysqld` sin bloquear la tabla apropiadamente.
- Está ejecutando varios servidores `mysqld` utilizando el mismo directorio de datos en un sistema que no soporta un buen mecanismo de bloqueo de sistema de archivos (normalmente gestionado por el gestor de bloqueos `lockd`), o está ejecutando múltiples servidores con la opción `--skip-external-locking`.
- Usted tiene un archivo de datos o índices erróneo que contiene datos muy corruptos que han confundido a `mysqld`.
- Ha encontrado un fallo en el código de almacenaje de datos. Esto no es muy probable, pero es posible. En este caso, puede intentar cambiar el tipo de la tabla a otro motor de almacenamiento utilizando `ALTER TABLE` sobre una copia reparada de la tabla.

Debido a que es muy difícil saber por qué algo está fallando, primero intente comprobar las cosas que funcionan frente a las que fallan en su caso. Por favor, intente las siguientes comprobaciones:

- Pare el servidor `mysqld` con `mysqladmin shutdown`, ejecute `myisamchk --silent --force */*.MYI` desde el directorio de datos para comprobar todas las tablas `MyISAM`, y reinicie `mysqld`. Esto le hará estar seguro de que está ejecutando el servidor desde un estado inicial limpio. Consulte Capítulo 5, *Administración de bases de datos*.
- Inicie `mysqld` con la opción `--log` e intente determinar a través de la información escrita en el registro si hay alguna consulta particular que mate al servidor. Sobre el 95% de todos los fallos son relacionados con una consulta en particular. Normalmente, esta es una de las últimas consultas en el archivo de registro antes de que el servidor se reinicie. Consulte Sección 5.10.2, “El registro general de consultas”. Si usted puede matar repetidamente MySQL con una consulta específica, aún cuando ha comprobado todas las tablas antes de ejecutarla, entonces ha sido capaz de localizar el fallo, y debería enviar un informe de fallos. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

- Intente hacer un juego de pruebas que podamos utilizar para repetir el problema. Consulte Sección D.1.6, “Crear un caso de prueba tras haber encontrado una tabla corrupta”.
- Intente ejecutar las pruebas en el direcotrio `mysql-test` y los bancos de pruebas MySQL. Consulte Sección 27.1.2, “El paquete de pruebas MySQL Test”. Estos deberían comprobar bastante bien MySQL. También puede añadir código a los bancos de pruebas que simulen su aplicación. Los bancos de pruebas pueden ser localizados en el directorio `sql-bench` en una distribución de código fuente, o, en una distribución binaria, en el directorio `sql-bench` bajo su directorio de instalación de MySQL.
- Intente el script `fork_big.pl`. (Está ubicado en el directorio `tests` de una distribución de código fuente.)
- Si usted configura MySQL para depuración, es mucho más fácil recoger información sbore los posibles errores cuando algo va mal. Configurar MySQL para depuración causa que un gestor de memoria seguro se incluya para encontrar algunos errores. También produce muchas más salidas informativas sobre lo que está pasando. Reconfigure MySQL con la opción `--with-debug` o `--with-debug=full` para ejecutar después `configure` y seguidamente recompilar. Consulte Sección D.1, “Depurar un servidor MySQL”.
- Asegúrese de que usted ha aplicado los últimos parches para su sistema operativo.
- Utilice la opción `--skip-external-locking` para `mysqld`. En algunos sitemas, el gestor de bloqueos `lockd` no funciona apropiadamente; la opción `--skip-external-locking` le dice a `mysqld` que no utilice bloqueos externos. (Esto significa que no puede ejecutar dos servidores `mysqld` en el mismo directorio de datos, y que debe ser muy cuidadoso cuando utilice `myisamchk`). Aún así, puede ser muy instructivo intentar la opción como prueba.)
- ¿Ha intentado el comando `mysqladmin -u root processlist` cuando `mysqld` parece estar ejecutándose pero no responde? A veces `mysqld` no está comatoso, aunque usted pueda pensar lo contrario. El problema puede ser que todas las conexiones estén en uso, o que haya algún problema de bloqueo interno. `mysqladmin -u root processlist` es normalmente capaz de hacer una conexión aún en esots casos, y puede proveerle de información útil sobre el número actual de conexiones y su estado.
- Ejecute del comando `mysqladmin -i 5 status` o `mysqladmin -i 5 -r status` en una ventana separada para producir estadísticas mientras ejecuta sus otras consultas.
- Intente lo siguiente:

  1. Inicie `mysqld` desde `gdb` (u otro depurador). Consulte Sección D.1.3, “Depurar `mysqld` con `gdb`”.
  2. Ejecute sus scripts de comprobación.
  3. Imprima el trazado y las variables locales en los tres niveles más bajos. En `gdb`, puede hacer esto con los siuguientes comandos cuando `mysqld` ha caido dentro de `gdb`:

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     Con `gdb`, usted puede también examinar que hilos de ejecución existen con `info threads` y cambiar a un hilo específico con `thread #`, donde `#` es el ID del hilo.

- Trate de simular su aplicación con un script Perl para forzar a que MySQL falle o se comporte indebidamente.

- Envíe un informe de fallos normal. Consulte Sección 1.6.1.3, “Cómo informar de bugs y problemas”. Sea aún más detallado de lo normal. Debido a que MySQL funciona para mucha gente, podría ser que los fallos fueran resultado de algo que exista tan sólo en su máquina (por ejemplo, un error relacionado con sus librerías de sistema particulares).

- Si usted tiene un problema con tablas que contengan filas de longitud dinámica y está utilizando únicamente columnas `VARCHAR` (no columnas `BLOB` ni `TEXT`), puede intentar cambiar todas las columnas `VARCHAR` a `CHAR` con `ALTER TABLE`. Esto fuerza a MySQL a utilizar filas de tamaño fijo. Las filas de tamaño fijo ocupan un poco más de espacio, pero son mucho más tolerantes a la corrupción.

  El código de filas dinámicas actual ha sido utilizado en MySQL AB durante muchos años con muy pocos problemas, pero las filas de longitud dinámica son, por naturaleza, más propensas a errores, así que podría ser una buena idea intentar esta estrategia y ver si ayuda.

- No deje fuera el hardware de su servidor cuando esté diagnosticando problemas. El hardware defectuoso puede ser causa de corrupción de datos. Debe poner especial atención en la RAM y los discos duros cuando esté buscando problemas de hardware.

A.4.3. Cómo se comporta MySQL ante un disco lleno

Esta sección explica como MySQL se comporta frente a errores de disco lleno (como “no space left on device”), y, a partir de MySQL 4.0.22, a errores de límite de cuota (como “write failed” o “user block limit reached”).

Esta sección es relevante para escrituras a tablas `MyISAM`. A paritr de MySQL 4.1.9, también se aplica a las escrituras a archivos de registro binario y también a sus índices, excepto por las referencias a “row” y “record” que deberían entenderse como “event.”

Cuando se llega a la situación de disco lleno, MySQL hace lo siguiente:

- Comprueba una vez por minuto si hay suficiente espacio para escribir la fila actual. Si hay espacio, continua como si nada hubiese pasado.

- Cada 10 minutos, escribe una entrada en el archivo de registro, avisando sobre la situación de disco lleno.

Para aliviar el problema, puede hacer lo siguiente:

- Para continuar, lo único que tiene que hacer es liberar suficiente espacio en disco para insertar todos los registros.

- Para abortar el hilo, debe utilizar el comando `mysqladmin kill`. El hilo se aborta la próxima vez que comprueba el disco (en un minuto).

- Otros hilos podrían estar esperando a la tabla que causó la situación de disco lleno. Si tiene varios hilos bloqueados matar al que está esperando por la situación de disco lleno permite que los demás hilos puedan continuar.

Hay excepciones al comportamiento descrito previamente, como cuando utiliza `REPAIR TABLE` o `OPTIMIZE TABLE`, o cuando los índices son creados en batch tras la sentencia `LOAD DATA INFILE` o la sentencia `ALTER TABLE`. Todas estas sentencias podrían crear grandes archivos temporales que, si se dejan a su suerte, podrían causar enormes problemas para el resto del sistema. Si el disco se llena mientras MySQL está haciendo alguna de estas operaciones, elimina el archivo temporal grande y marca la tabla como corrupta. La excepción es que con `ALTER TABLE`, la tabla antigua se deja tal como estaba, sin cambios.

A.4.4. Dónde almacena MySQL los archivos temporales

MySQL utiliza el valor de la variable de entorno `TMPDIR` como la ruta al directorio en el que almacenar archivos temporales. Si no tiene configurada `TMPDIR`, MySQL utiliza el valor por defecto del sistema, que normalmente es `/tmp`, `/var/tmp`, o `/usr/tmp`. Si el sistema de archivos que contiene su directorio de archivos temporales es demasiado pequeño, puede utilizar la opción `--tmpdir` de `mysqld` para especificar un directorio en un sistema de archivos en el que tenga suficiente espacio.

Desde MySQL 4.1, la opción `--tmpdir` puede establecerse como una lista de varias rutas que son utilizadas de manera round-robin. Las rutas deben estar separadas por caracteres de dos puntos (`':'`) en Unix, y de punto y coma (`';'`) en Windows, NetWare y OS/2. **Nota**: Para repartir la carga de manea efectiva, estas rutas tienen que estar colocadas en diferentes discos físicos, no en diferentes particiones del mismo disco.

Si el servidor MySQL actua como un esclavo de replicación, no debería hacer que `--tmpdir` apunte a un drectorio en un sistema de archivos basada en memoria, o un directorio que se limpia cuando la máquina se reinicia. Un esclavo de replicación necesita algunos de sus archivos temporales para sobrevivir a un reinicio de la máquina de manera que pueda replicar tablas temporales u operaciones `LOAD DATA INFILE`. Si se pierden archivos del directorio temporal cuando el servidor reinicia, la replicación falla.

MySQl crea todos los archivos temporales como archivos ocultos. Esto asegura que los archivos temporales se eliminan si `mysqld` termina. La desventaja de utilizar archivos ocultos es que usted no podrá ver el gran archivo temporal que llena su sistema de archivos.

Cuando ordena (`ORDER BY` o `GROUP BY`), MySQL normalmente utiliza uno o dos archivos temporales. El espacio máximo de disco requerido se determina con la siguiente expresión:

```
(longitud de lo que está siendo ordenado + tamañode(puntero de fila))
* número de filas concordantes
* 2
```

El tamaño del puntero de fila es normalmente de 4 bytes, pero puede crecer en el futuro para tablas realmente grandes.

Para algunas consultas `SELECT`, MySQL también crea tablas SQL temporales. Estas no están ocultas y tienen nombres del tipo `SQL_*`.

`ALTER TABLE` crea una tabla temporal en el mismo directorio que la tabla original.

A.4.5. Cómo proteger o cambiar el fichero socket de MySQL `/tmp/mysql.sock`

El lugar por defecto del archivo de socket Unix que el servidor utiliza para comunicarse con los clientes locales es `/tmp/mysql.sock`. Esto podría causar problemas, porque en agunas versiones de Unix, cualquiera puede borrar archivos del direcotorio `/tmp`.

En la mayoría de versiones de Unix, usted puede proteger su directorio `/tmp` de manera que los archivos solo puedan ser borrados por sus propietarios o el superusuario (`root`). Para hacer esto, establezca el bit `sticky` en el directorio `/tmp` entrando en el sitema como `root` y utilizando el siguiente comando:

```
shell> chmod +t /tmp
```

Puede comprobar si el bit `sticky` esta activado ejecutando `ls -ld /tmp`. Si el último carácter de privilegios es `t`, el bit está activado.

Otra solución es cambiar el lugar donde el servidor crea el archivo socket de Unix. Si hace esto, debería también hacérselo saber a los programas clientes. Puede especificar el lugar del archivo de diferentes maneras:

- Especifique la ruta en un archivo de opciones global o local. Por ejemplo, ponga las siguientes líneas en `/etc/my.cnf`:

  ```
  [mysqld]
  socket=/ruta/a/socket

  [client]
  socket=/ruta/a/socket
  ```

  Consulte Sección 4.3.2, “Usar ficheros de opciones”.

- Especifique una opción `--socket` en la línea de comandos a `mysqld_safe` y cuando ejecute programas cliente.

- Establezca la variable de entorno `MYSQL_UNIX_PORT` apuntando a la ruta del archivo de socket Unix.

- Recompile MySQL desde el código fuente para utilizar una ruta por defect diferente para el archivo socket. Defina la ruta al arhivo con la opción `--with-unix-socket-path` cuando ejecute `configure`. Consulte Sección 2.8.2, “Opciones típicas de `configure`”.

Puede comprobar si el nuevo lugar del socket funciona intentando conectar al servidor con el siguiente comando:

```
shell> mysqladmin --socket=/ruta//socket version
```

A.4.6. Problemas con las franjas horarias

Si tiene un prooblema cn que `SELECT NOW()` retorno los valores en GMT y no en su hora local, tiene que decirle al servidor cual es su zona actual. Lo mismo es aplicable para `UNIX_TIMESTAMP()` su retorna un valor incorrecto. Esto debe hacerse para el entorno en que el servidor se ejecuta; por ejemplo en `mysqld_safe` o `mysql.server`. Consulte Apéndice E, *Variables de entorno*.

Puede establecer la zona horaria del servidor con la opción `--timezone=timezone_name` para `mysqld_safe`. También puede establecerla configurando la varible de entorno `TZ` antes de iniciar `mysqld`.

Los valores permisibles para `--timezone` o `TZ` dependen de sus sistema. Consulte la documentación de su sistema operativo para ver qué valores son aceptables.

A.5. Problemas relacionados con consultas

A.5.1. Sensibilidad a mayúsculas en búsquedas

Por defecto, las búsquedas de MySQL no tienen sensibilidad a mayúsculas (aunque algunos juegos de caracteres tienen siempre sensibilidad a mayúsculas como el checo). Esto significa que si busca con *`col_name`* `LIKE 'a%'`, obtendrá todas las columnas que comiencen con `A` o `a`. Si quiere hacer que esta consulta sea sensible a las mayúsculas, asegúrese de que alguno de los operandos tiene una colación binaria sensible a las mayúsculas. Por ejemplo, si está comparando una columna y una cadena de caracteres que tienen ambos el juego de caracteres `latin1`, puede utilizar el operador `COLLATE` para causar que cualquiera de los operandos tenga la colación `latin1_general_cs` o `latin1_bin`. Por ejemplo:

```
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
```

Si quiere que una columna siemrpe sea tratada de manera sensible a mayúsculas, declárela de esa manera, con una colación que lo sea. Consulte Sección 13.1.5, “Sintaxis de CREATE TABLE”.

Previamentea MySQL 4.1, COLLATE no está disponible. Utilice el operador BINARY en las expresiones para tratar una cadena de caráceres como binaria: BINARY *col_name* LIKE 'a%' o *col_name* LIKE BINARY 'a%'. En declaracines de columnas, utilice el atributo BINARY.

Las operaciones de comparación simples (>=, >, =, <, <=, ordenación, y agrupación) se basan en el valor de orden del carácter. Caracteres con el mismo valor de orden (como 'E', 'e', y 'é') se tratan como el mismo carácter.

A.5.2. Problemas en el uso de columnas DATE

El format de un valor DATE es 'YYYY-MM-DD'. De acuerdo al estándar SQL ningún otro formato se permite. Debería usar este formato en las sentencias UPDATE y en la cláusula WHERE de las sentencias SELECT. Por ejemplo:

```
mysql> SELECT * FROM nombre_de_tabla WHERE date >= '2003-05-05';
```

Por motivos de conveniencia, MySQL convierte automáticamente una fecha a un número si la fecha se utiliza en un contexto numérico (y viceversa). Es también suficientemente inteligente para permitir diversos formatos de cadena de caracteres cuando se está actualizando y en las cláusulas WHERE que comparan una fecha con una columna TIMESTAMP, DATE, o DATETIME. (Diversos formatos, significa que puede utilizarse cualquier carácter de puntuación como separador de las partes de la fecha. Por ejemplo, '2004-08-15' y '2004#08#15' son equivalentes.) MySQL también puede convertir una cadena de caracteres que no contenga separadores (como '20040815'), siempre que tenga sentido como fecha.

Cuando compara una colmna DATE, TIME, DATETIME, o TIMESTAMP a una cadena de caracteres constante con los operadores <, <=, =, >=, >, o BETWEEN, MySQL normalmente convierte la cadena de caráteres a un entero largo interno para una comparación más rápda (y también para hacer una comparación algo más permisiva). Aún así, esta conversión está sujeta a las siguientes restricciones:

- Cuando compara dos columnas
- Cuando usted compare columnas DATE, TIME, DATETIME, o TIMESTAMP con una expresión.
- Cuando usted utilice cualquier otro método de comparación que no sean aquellos recientemente citados, como IN o STRCMP().

Para estos casos excepcionales, la comparación se hace convirtiendo los objetos a cadenas de caracteres y realizando una comparación de cadenas de caracteres.

Para mantener todo a salvo, asuma que las fechas son comparadas como cadenas de caracteresy utilice las funciones de cadena de caracteres apropiadas para comparar un valor temporal con una cadena de caracteres.

La fecha especial '0000-00-00' puede almacenarse y recogerse como '0000-00-00'. Cuando se utiliza una fecha '0000-00-00' a través de MyODBC, es convierte automáticamente en NULL en MyODBC 2.50.12 y superiores, porque ODBC no puede gestionar este tipo de fechas.

Como MySQL realiza las conversiones anteriores, las siguientes sentencias funcionan:

```
mysql> INSERT INTO nombre_de_tabla (idate) VALUES (19970505);
mysql> INSERT INTO nombre_de_tabla (idate) VALUES ('19970505');
mysql> INSERT INTO nombre_de_tabla (idate) VALUES ('97-05-05');
mysql> INSERT INTO nombre_de_tabla (idate) VALUES ('1997.05.05');
mysql> INSERT INTO nombre_de_tabla (idate) VALUES ('1997 05 05');
mysql> INSERT INTO nombre_de_tabla (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM nombre_de_tabla WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM nombre_de_tabla WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM nombre_de_tabla WHERE idate >= 19970505;
mysql> SELECT idate FROM nombre_de_tabla WHERE idate >= '19970505';
```

Pero lo siguente no funciona:

```
mysql> SELECT idate FROM nombre_de_tabla WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` es una función de comparación de cadenas, así que convierte `idate` a una cadena de caracteres en formato `'YYYY-MM-DD'` y realiza una comparación de cadenas. Es decir, NO convierte `'20030505'` en la fecha `'2003-05-05'` para realizar una comparación de fechas.

Si está utilizando el modo SQL `ALLOW_INVALID_DATES (permitir fechas no válidas)`, MySQL le permite almacenar las fechas que le proporciona con una comprobación mínima: MySQL se asegura únicamente que el día esté en el rango entre 1 y 31, y el mes en el rango entre 1 y 12.

Esto hace que MySQL sea muy convenienta para las aplicaciones web en que usted obtiene el año, mes y día en tres campos diferentes y quiere almacenar exactamente lo que el usuario insertó (sin validación de fechas).

Si no está utilizando el modo SQL `NO_ZERO_IN_DATE`, la parte del día o el mes pueden ser cero. Esto es conveniente si quiere almacenar una fecha de nacimiento en una columna `DATE` y solo conoce parte de esa fecha.

Si no está utlizando el modo SQL `NO_ZERO_DATE`, MySQL también le permite almacenar `'0000-00-00'` como fecha comodín. En algunos momentos esto puede ser más adecuado que utilizar valores `NULL`.

Si la fecha no puede ser convertida a ningún valor razonable, se almacena un `0` en la columna `DATE`, que se extrae como `'0000-00-00'`. Esto es una cuestión de velocidad y conveniencia. Creemos que la responsabilidad del servidor de bases de datos es recoger la misma fecha que usted almacenó (aún cuando la fecha no fuese lógicamente correcta en todos los casos). Creemos que es responsabilidad de la aplicación y no del servidor comprobar las fechas.

Si quiere que MySQL compruebe todas las fechas y acepte solo las que sean correctas (al menos que esto sea sobreseido por IGNORE), debería establecer el modo SQL a `"NO_ZERO_IN_DATE,NO_ZERO_DATE"`.

La gestión de fechas en MySQL 5.0.1 y anteriores funciona como en MySQL 5.0.2 con el modo SQL `ALLOW_INVALID_DATES` activado.

A.5.3. Problemas con valores `NULL`

El concepto del valor `NULL` es una fuente común de confusión para los recién llegados a SQL, que frecuentemente piensan que `NULL` es lo mismo que una cadena de caracteres vacía `''`. Esto no es así. Por ejemplo, las siguientes sentencias son completamente diferentes:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Ambas sentencias insertan un valor en la columna phone, pero la primera inserta un valor NULL y la segunda una cadena vacía. El significado de la primera se puede traducir por “el número de teléfono no es conocido”, y el significado de la segunda es “se sabe que la persona no tiene teléfono, y por tanto, no hay número de teléfono.”

Para ayudarle con la gestión de NULL, tiene disponibles los operadores IS NULL y IS NOT NULL y la función IFNULL().

En SQL, el valor NULL nunca da verdadero al compararlo con otro valor, aún cuando este valor sea también NULL. Una expresión que contiene NULL siempre produce un valor NULL a menos que se indique lo contrario en la documentación de los operadores y funciones implicadas en la expresión. Todas las columnas en el siguiente ejemplo retornan NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

Si quiere buscar valores de columna que son NULL, no puede utilizar una comprobación expr = NULL. La siguiente sentencia no retorna registros, porque expr = NULL nunca es verdadero para cualquier expresión:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

Para buscar valores NULL debe utilizar la comprobaciónIS NULL. Las siguientes sentencias muestran como encontrar el número de teléfono NULL y el vacío:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

Puede añadir una columna índice que tenga valores NULL si utiliza MySQL 3.23.2 o superiores, o si está utilizando los motores de almacenamiento MyISAM, InnoDB, o BDB. A partir de MySQL 4.0.2, el motor de almacenamiento MEMORY también tiene soporte para valores NULL en los índices. Si no es así, todas las columnas de índice deben ser declaradas como NOT NULL y usted no podrá insertar un valor NULL en la columna.

Cuando lee datos con LOAD DATA INFILE, las columnas vacías o inexistentes se rellenan con ''. Si quiere un valor NULL en una columna, usted deberá utilizar \N en el archivo de datos. La palabra exacta “NULL” puede ser también utilizada en algunas circunstancias. Consulte Sección 13.2.5, “Sintaxis de LOAD DATA INFILE”.

Cuando utiliza DISTINCT, GROUP BY, o ORDER BY, todos los valores NULL son tratados como iguales.

Cunado utilice ORDER BY, los valores NULL son mostrados al principio, o al final si se especifica DESC para ordenar de manera descendiente. Excepción: De MySQL 4.0.2 hasta 4.0.10, los valores NULL se muestran primeros independientemente del orden.

Las funciones de agregación (resumen) como COUNT(), MIN(), y SUM() ignoran los valores NULL. La excepción a esto es COUNT(*), que cuenta filas y no valores de columna individuales. Por ejemplo, la siguiente sentencia produce dos conteos. El primero es un recuento del número de registros en la tabla, y el segundo un recuento del número de valores diferentes a NULL en la columna age:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

En algunos tipos de columnas, MySQL gestiona los valores NULL de manera especial. Si inserta un valor NULL en una columna TIMESTAMP, se insertan la fecha y hora actual. Si inserta un valor NULL en una columna entera que tiene el atributo AUTO_INCREMENT, se inserta el siguiente número en la secuencia.

A.5.4. Problemas con alias de columnas

Puede utilizar un alias para referirse a una columna en las clausulas `GROUP BY`, `ORDER BY`, o `HAVING`. Los alias también se puede utilizar para dar mejores nombres a las columnas:

```
SELECT SQRT(a*b) AS root FROM nombre_de_tabla GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM nombre_de_tabla GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM nombre_de_tabla;
```

El SQL estándar no permite referirse a un alias de columna en una clausula `WHERE`. Esto es porque cuando se ejecuta el código de `WHERE`, el valor de la columna podría no estar establecido aún. Por ejemplo, la siguiente consulta es ilegal:

```
SELECT id, COUNT(*) AS cnt FROM nombre_de_tabla WHERE cnt > 0 GROUP BY id;
```

La sentencia `WHERE` se ejecuta para determinar qué registros serán incluidos en la parte de `GROUP BY`, así como `HAVING` se utiliza para decidir qué filas del conjunto de resultados deben utilizarse.

A.5.5. Fallo en la cancelación de una transacción con tablas no transaccionales

Si usted recibe el siguiente mensjae cuando intenta realizar un `ROLLBACK`, significa que una o más de las tablas que utilizó en la transacción no tienen soporte para trasacciones:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

Estas tablas no transaccionales no se ven afectadas por la sentencia `ROLLBACK`.

Si usted no estaba mezclando tablas transaccionales y no transaccionales deliberadamente en la transacción, la razón más probable para que este mensaje aparezca es que una tabla que usted pensaba que era transaccional no lo es realmente. Esto puede pasar si usted intenta crear una tabla utilizando un motor de almacenamiento que no está configurado en su servidor `mysqld` (o que fue desactivado con una opción inicial). Si `mysqld` no tiene soporte para ese motor de almacenamiento, crea la tabla como una tabla `MyISAM`, que es no transaccional.

Puede comprobar el tipo de una tabla utilizando cualquiera de las siguientes sentencias:

```
SHOW TABLE STATUS LIKE 'nombre_de_tabla';
SHOW CREATE TABLE nombre_de_tabla;
```

Consulte Sección 13.5.4.18, “Sintaxis de SHOW TABLE STATUS” y Sección 13.5.4.5, “Sintaxis de SHOW CREATE TABLE”.

Puede consultar qué motores de almacenamiento soporta su servidor `mysqld` mediante la sentencia:

```
SHOW ENGINES;
```

Anteriormente a MySQL 4.1.2, la sentencia `SHOW ENGINES` no está disponible. Utilice la siguiente sentencia en su lugar y compruebe el valor de la variable que está asociada al motor de almacenamiento en el que está interesado:

```
SHOW VARIABLES LIKE 'have_%';
```

Por ejemplo, para determinar si el motor `InnoDB` está disponible, compruebe el valor de la variable `have_innodb`.

Consulte Sección 13.5.4.8, “Sintaxis de `SHOW ENGINES`” y Sección 13.5.4.21, “Sintaxis de `SHOW VARIABLES`”.

A.5.6. Borrar registros de tablas relacionadas

MySQL no soporta subconsultas en versiones previas a la 4.1, o el uso de más de una tabla en la sentencia `DELETE` anteriormente a la versión 4.0. Si su versión de MySQL no soporta subconsultas, o sentencias `DELETE` multi-tabla, puede utilizar las siguientes estrategias para borrar registros de dos tablas relacionadas:

1. Seleccione (`SELECT`) los registros basados en una condición `WHERE` en la tabla principal.
2. Borre (`DELETE`) los registros de la tabla principal basándose en la misma condición.
3. `DELETE FROM tabla_relacionada WHERE columna_relacionada IN (registros_seleccionados)`.

Si la longitud total de la sentencia `DELETE` para `tabla_relacionada` es más de 1MB (el valor por defecto de la variable de sistema `max_allowed_packet`), debería partirla en partes más pequeñas y ejecutar múltiples sentencias `DELETE`. Problablemente obtenga el borrado más rápido especificando entre 100 y 1000 valores de `columna_relacionada` por sentencia si `columna_relacionada` está indexada. Si `columna_relacionada` no está indexada, la velocidad es independiente del número de argumentos en la clausula `IN`.

A.5.7. Resolver problemas con registros que no salen

Si tiene una consulta complicada que utiliza muchas tablas pero que no retorna ningún registro, debería utilizar el siguiente procedimiento para encontrar qué está fallando:

1. Compruebe la consulta con `EXPLAIN` para ver si encuentra algo que sea erróneo de una manera obvia. Consulte Sección 7.2.1, “Sintaxis de `EXPLAIN` (Obtener información acerca de un `SELECT`)”.
2. Seleccione únicamente aquellas columnas que están en la clausula `WHERE`.
3. Vaya quitando las tablas una a una de la consulta hasta que devuelva algunos registros. Son las tablas son grandes, es una buena idea utilizar `LIMIT 10` con la consulta.
4. Ejecute un `SELECT` sobre la columna que debería haber extraido un registro de la tabla que fue eliminada en último lugar de la consulta.
5. Si está comparando columnas `FLOAT` o `DOUBLE` con números que tengan decimales, no puede utilizar comparaciones de igualdad (`=`). Este problema es común en la mayor parte de los lenguajes informáticos debido a que no todos los valores de coma flotante se pueden almacenar con precisión exacta. En algunos casos, cambiar de `FLOAT` a `DOUBLE` soluciona esto. Consulte Sección A.5.8, “Problemas con comparaciones en coma flotante”.
6. Si aún así no puede imaginar que es lo que funciona incorrectamente cree un pequeña prueba que pueda ser ejecutada con `mysql test < query.sql` y que muestre sus problemas. Muede crear un archivo de prueba volcando las tablas con `mysqldump --quick db_name nombre_de_tabla_1 ... nombre_de_tabla_n > query.sql`. Abra el archivo en un editor, elimine algunas líneas de inserción (si hay más de las necesarias para demostrar el problema), y añada su sentencia `SELECT` al final del archivo.

   Verifique que el archivo de prueba demuestra el problema ejecutando los siguientes comandos:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Envíe el archivo de pruebas utilizando `mysqlbug` a la lista de correo general de MySQL. Consulte Sección 1.6.1.1, “Las listas de correo de MySQL”.

A.5.8. Problemas con comparaciones en coma flotante

Tenga en cuenta que la siguiente sección es relevante principalmente para versiones de MySQL anteriores a 5.0.3. A partir de la versión 5.0.3, MyQL realiza las operaciones `DECIMAL` con una precisión de 64 dígitos decimales, lo que debería resolver los problemas de imprecisión más comunes en lo que se refiere a columnas `DECIMAL`. Para las columnas `DOUBLE` y `FLOAT` los problemas siguen porque la inexactitud es la naturaleza básica de los números en coma flotante.

Los números en coma flotante a veces causan confusión porque no son almacenados como valores exactos en la arquitectura de la computadora. Lo que puede ver en la pantalla no es el valor exacto del número. Los tipos de columna `FLOAT`, `DOUBLE`, y `DECIMAL` son de este tipo. Las columnas `DECIMAL` almacenan valores con precisión exacta porque se representan como cadenas de caracteres, pero los cálculos sobre valores `DECIMAL`, en versiones previas a 5.0.3, eran realizadas utilizando operaciones de coma flotante.

El siguiente ejemplo (para versiones anteriores a 5.0.3 de MySQL) demuestra el problema. Muestra que incluso para los tipos de columna `DECIMAL`, los cálculos se realizan utilizando operaciones de coma flotante que están sujetas al error. (En todas las versiones de MySQL, tendría problemas similares si reemplazara las columnas `DECIMAL` con `FLOAT`).

```
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 |
+------+--------+-------+
```

El resultado es correcto. Aunque los cinco primeros registros aparentan no poder pasar la comparación (los valores `a` y `b` no parecen ser diferentes), lo son porque la diferencia entre los dos números se encuentra a partir del decimal número diez aproximadamente, dependiendo de la arquitectura de la computadora.

A partir de MySQL 5.0.3, usted obtendría únicamente el último registro en el resultado anterior.

El problema no puede ser resulto utilizando funciones como `ROUND()` o similares, porque el resultado sigue siendo un número en coma flotante:

```
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 |
+------+--------+-------+
```

Esta es la apariencia que los números en la columna a tienen al mostrarse más cifras decimales:

```
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 |
+------+----------------------+-------+
```

Dependiendo de la arquitectura de su computadora, usted puede que vea o no resultados similares. Diferentes procesadores pueden evaluar los números en coma flotante de manera diferente. Por ejemplo, en algunas máquinas usted podría obtener los resultados “correctos” multiplicando ambos argumentos por 1, como en el siguiente ejemplo.

Aviso: Nunca utilice este método en sus aplicaciones. No es un ejemplo de procedimiento fiable.

```
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 |
+------+--------+------+
```

La razón por la que el ejemplo precedente parece funcionar es que en una máquina particular donde la prueba fue realizada, la aritmética de coma flotante del procesador redondea los números al mismo valor. No obstante, no hay ninguna regla que diga que un procesador deba hacerlo, así que este método no merece confianza.

La manera adecuada de hacer comparaciones en coma flotante es primero decidir una tolerancia aceptable para las diferencias entre los números y realizar entonces las comparaciones sobre el valor de tolerancia. Por ejemplo, si estamos de acuerdo en que los números en coma flotante deben ser considerados el mismo si están dentro de una precisión de uno sobre diez mil (0.0001), la comparación debería escribirse para encontrar diferencias mayores que el valor de tolerancia:

```
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)
```

De manera análoga, para obtener los registros en que los números son iguales, la comparación debería encontrar diferencias dentro del valor de tolerancia:

```
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 |
+------+-------+-------+
```

A.6. Cuestiones relacionadas con el optimizador

MySQL utiliza un optimizador basado en costos para determinar la mejor manera de resolver una consulta. En muchos casos, MySQL puede calcular el mejor plan de consulta posible, pero a veces MySQL no tiene suficiente información sobre los datos a mano y tiene que hacer suposiciones “educadas” sobre los datos.

Em ñps casps em qie MySQL no hace lo "adecuado", las herramientas que usted tiene disponible para ayudar a MySQL son:

- Utilice la sentencia `EXPLAIN` para obtener información sobre como MySQL procesa una consulta. Para utilizarlo, simplemente añada la palabra clave `EXPLAIN` al inicio de su sentencia `SELECT`:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` está explicado con más detalle en Sección 7.2.1, “Sintaxis de `EXPLAIN` (Obtener información acerca de un `SELECT`)”.

- Utilice `ANALYZE TABLE` *`nombre_de_tabla`* para actualizar las definiciones de claves de la tabla analizada. Consulte Sección 13.5.2.1, “Sintaxis de `ANALYZE TABLE`”.

- Utilice `FORCE INDEX` sobre la tabla analizada para decirle a MySQL que los escaneos de tabla son muy costosos comparado con utilizar el índice dado. Consulte Sección 13.2.7, “Sintaxis de `SELECT`”.

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

  `USE INDEX` y `IGNORE INDEX` podrían también ser útiles.

- `STRAIGHT_JOIN` a nivel global y de tabla. Consulte Sección 13.2.7, “Sintaxis de `SELECT`”.

- Puede adaptar variables globales o específicas de hilos de ejecución. Por ejemplo, inicie `mysqld` con la opción `--max-seeks-for-key=1000` o utilice `SET max_seeks_for_key=1000` para decirle al optimizador que asuma que ninguna rastreo de claves genera más de 1000 búsquedas. Consulte Sección 5.3.3, “Variables de sistema del servidor”.

A.7. Cuestiones relacionadas con definiciones de tabla

A.7.1. Problemas con `ALTER TABLE`

`ALTER TABLE` cambia una tabla a el juego de caracteres actual. Si obtiene un error de clave duplicada durante una operación `ALTER TABLE`, la causa puede ser que el nuevo juego de caracteres haga equivaler dos claves, o que la tabla está corrompida. En este último caso, debería ejecutar `REPAIR TABLE` sobre dicha tabla.

Si `ALTER TABLE` termina abruptamente con el siguiente error, el problema podría ser que MySQL falló durante una operación `ALTER TABLE` anterior, y que hay una antigua llamada`A-xxx` o `B-xxx` por el sistema:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

En este caso, vaya al directorio de datos de MySQL y borre todos los archivos que tengan nombres que comiencen con `A-` o `B-`. (Quizá debería moverlos a algún otro lugar en vez de borrarlos directamente.)

`ALTER TABLE` trabaja de la siguiente manera:

- Crea una tabla llamada `A-xxx` con los cambios estructurales demandados.
- Copia todas las filas de la tabla original en `A-xxx`.
- Renombra la tabla original a `B-xxx`.
- Renombra `A-xxx` a el nombre de tabla original.
- Borra `B-xxx`.

Si algo falla durante la operación de renombrado, MySQL intenta deshacer los cambios. Si algo falla seriamente (aunque esto no debería pasar), MySQL podría dejar la vieja tabla como `B-xxx`. Un renombrado simple de los archivos de tabla al nivel de sistema debería devolverle todos sus datos.

Si usted utiliza `ALTER TABLE` en una tabla transaccional o si está utilizando Windows o OS/2, `ALTER TABLE` desbloquea la tabla si usted ha hecho un `LOCK TABLE` previo sobre ella. Esto es debido a que `InnoDB` y estos sistemas operativos no puede eliminar una tabla que está en uso.

A.7.2. Cómo cambiar el orden de las columnas en una tabla

Primero, cosidere si realmente necesita cambiar el orden de columnas en una tabla. El objetivo primordial de SQL es abstraer la aplicación del formato de almacenamiento de datos. Debería siempre especificar el orden en el que quiere recoger sus datos. La primera de las siguiente sentencias retorna las columnas en el orden First, consider whether you really need to change the column order in a table. The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in which you wish to retrieve your data. The first of the following statements returns columns in the order *`col_name1`*, *`col_name2`*, *`col_name3`*, mientras que la segunda las retorna en orden *`col_name1`*, *`col_name3`*, *`col_name2`*:

```
mysql> SELECT col_name1, col_name2, col_name3 FROM nombre_de_tabla;
mysql> SELECT col_name1, col_name3, col_name2 FROM nombre_de_tabla;
```

Si usted decide cambiar el orden de las columnas de la tabla aún así, debe hacerlo de la siguiente manera:

1. Cree una tabla con las columnas en el nuevo orden.
2. Ejecute esta sentencia:

```
mysql> INSERT INTO new_table
    -> SELECT columns-in-new-order FROM old_table;
```

3. Elimine o renombre `old_table`.

4. Renombre la nueva tabla al nombre original:

```
mysql> ALTER TABLE new_table RENAME old_table;
```

`SELECT *` es muy útil para consultas de prueba. No obstante, en una aplicación, *nunca* debería utilizar `SELECT *` y después recoger las columnas basándose en su posición. El orden y posición en que las columnas son devueltas no es el mismo si usted añade, mueve, o elimina columnas. Un cambio simple a la estructura de su tabla podría causar que su aplicación falle.

A.7.3. Problemas con TEMPORARY TABLE

La siguiente lista indica las limitaciones en el uso de tablas temporales:

- Una tabla `TEMPORARY` solo puede ser de tipo `HEAP`, `ISAM`, `MyISAM`, `MERGE`, o `InnoDB`.
- No puede referirse a una tabla `TEMPORARY` más de una vez en la misma consulta. Por ejemplo, lo siguiente no funciona:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- La sentencia `SHOW TABLES` no muestra tablas `TEMPORARY`.
- No puede utilizar `RENAME` para renombrar una tabla `TEMPORARY`. No obstante, puede utilicer `ALTER TABLE` como alternativa:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

A.8. Problemas conocidos en MySQL

Esta sección es una lista de problemas conocidos en versiones recientes de MySQL.

Para información sobre temas específicos a una plataforma consulte las instrucciones de instalación y porte en Sección 2.12, “Notas específicas sobre sistemas operativos” y Apéndice D, *Portar a otros sistemas*.

A.8.1. Problemas de la versión 3.23 resueltos en una versión posterior de MySQL

Los siguientes problemas conocidos no han sido arreglados en MySQL 3.23 por diversos motivos, y no han sido clasificados como críticos.

- Arreglado en MySQL 4.0: Evite utilizar espacios al final de los nombres de columna porque esto puede causar comportamientos inesperados. (Bug #4196)
- Arreglado en MySQL 4.0.12: Puede obtener un abrazo murtal (deadlock) si utiliza `LOCK TABLE` para bloquear múltiples tablas y luego en la misma conexión utiliza `DROP TABLE` para eliminar una de ellas mientras otro hilo está intentando bloquearla. (Para romper el abrazo mortal, puede utilizar `KILL` para matar cualquiera de los hilos implicados.)

- Arreglado en MySQL 4.0.11: `SELECT MAX(key_column) FROM t1,t2,t3...` donde una de las tablas está vacía no retorna `NULL` sino que retorna el valor máximo de la columna.
- `DELETE FROM heap_table` sin una clausula `WHERE` no funciona en uan tabla `HEAP` bloqueada.

A.8.2. Problemas de la versión 4.0 resueltos en una versión posterior de MySQL

Los siguientes problemas conocidos no han sido arreglado sen MySQL 4.0 por diversos motivos, y no han sido clasificados como críticos.

- Arreglado en MySQL 4.1.10: Utilizanzo `HAVING`, usted podría obtener un fallo o un resultado erróneo si utiliza un alias para una función `RAND()`. Esto no será corregido en la versión 4.0 porque la corrección podría romper la compatibilidad con algunas aplicaciones.
- Arreglado en MySQL 4.1.1: En una `UNION`, el primer `SELECT` determina las propiedades de tipo, `max_length`, y `NULL` para las columnas del resultado.
- Arreglado en MySQL 4.1: En `DELETE`s con muchas tablas, usted no puede referirse a tablas que serán borradas mediante un alias.
- Arreglado en MySQL 4.1.2: No se puede mezclar `UNION ALL` y `UNION DISTINCT` en la misma consulta. Si utiliza `ALL` para un `UNION`, se utiliza para todos ellos.
- `FLUSH TABLES WITH READ LOCK` no bloquea a los `CREATE TABLE`, lo que puede causar un problema con la posición del registro binario cuando se hace una copia de seguridad completa de las tablas y el registro binario.
- Arreglado en MySQL 4.1.8: `mysqldump --single-transaction --master-data` se comportaban como `mysqldump --master-data`, así que el volqueado era bloqueante.
- Al utilizar la función `RPAD()` (o cualquier función que añada espacios a la derecha) en una consulta que tenía que ser resulta utilizando una tabla temporal, todos los resultados se obtenían sin esos espacios (es decir `RPAD()` no funcionaba).

A.8.3. Problemas de la versión 4.1 resueltos en una versión posterior de MySQL

Los siguientes problemas conocidos no han sido arreglados en MySQL 4.1 por diversos motivos, y no han sido clasificados como críticos.

- Arreglado en 5.0.3: Fixed in 5.0.3: `VARCHAR` y `VARBINARY` no almacenaban los espacios finales.

A.8.4. Cuestiones abiertas en MySQL

Los siguientes problemas son conocidos y arreglarlos es de alta prioridad:

- Si compara un valor `NULL` con una subconsulta utilizando `ALL/ANY/SOME` y la subconsulta retorna un conjunto vacío, la comparación podría devolver el resultado no estándar de `NULL` en vez de `TRUE` o `FALSE`. Esto será arreglado en MySQL 5.1.
- La optimización de las subconsultas para `IN` no es tan efectiva como para `=`.
- Aún cuando utilice `lower_case_table_names=2` (que permite a MySQL recordar la diferencia entre mayúsculas y minúsculas utilizada en los nombres de las bases de datos y las tablas), MySQL no la recuerda en los nombres de bases de datos al utilizar la función `DATABASE()` o en los diversos registro (en sistemas no sensibles a mayúsculas/minúsculas).

- Eliminar una restricción de `FOREIGN KEY` no funciona en una replicación porque la restricción puede tener otro nombre en el esclavo.
- `REPLACE` (y `LOAD DATA` con la opción `REPLACE`) no dispara a `ON DELETE CASCADE`.
- `DISTINCT` con `ORDER BY` no funciona dentro de `GROUP_CONCAT()` si no utiliza todas y únicamente aquellas columnas que están en la lista de `DISTINCT`.
- Si un usuario tiene una transacción ejecutándose durante mucho tiempo y otro usuario elimina una tabla que es actualizada durante la transacción, hay una pequeña posibilidad de que el registro binario contenga el comando `DROP TABLE` antes de que dicha tabla sea utilizada en la transacción. Planeamos arreglar esto haciendo que el comando `DROP TABLE` espera a que la tabla no sea utilizada en ninguna transacción.
- Cuando se inserta un valor de entero grande (entre 2^{63} y 2^{64}–1) en una columna decimal o de cadena de caracteres, se inserta con un valor negativo porque el número es evaluado en un contexto de entero con signo.
- `FLUSH TABLES WITH READ LOCK` no bloquea los `COMMIT` si el servidor se está ejecutando sin registro binario, lo que puede causar un problema (de consistencia entre tablas) cuando se hace una copia de seguridad completa.
- `ANALYZE TABLE` en una tabla `BDB` puede hacer que en algunos casos la tabla quede inutilizable hasta que reinicie `mysqld`. Si esto ocurre, busque los errores similares a este en el archivo de error de MySQL:

  ```
  001207 22:07:56  bdb:  log_flush: LSN past current end-of-log
  ```

- No ejecute `ALTER TABLE` en una tabla `BDB` en la que usted esté ejecutando transacciones de varias sentencias hasta que todas esas transacciones hayan sido completadas. (La transacción podría ser ignorada.)
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, y `REPAIR TABLE` podría causar problemas en tablas para las que usted esté utilizando `INSERT DELAYED`.
- Realizar `LOCK TABLE ...` y `FLUSH TABLES ...` no garantiza que no haya una transacción a medio ejecutar en progreso en la tabla.
- Las tablas `BDB` son relativamente lentas de abrir. Si usted tiene muchas tablas `BDB` en una base de datos, un cliente `mysql` podría tardar mucho en estar listo si no está utilizando la opción `-A` o sí está utilizando `rehash`. Esto es especialmente notable cuando tiene una cache de tablas grande.
- La replicación utiliza registro a nivel de consulta: El maestro escribe las sentencias ejecutadas al registro binario. Este es un método de registro muy rápido, compacto, y eficiente que trabaja de manera perfecta en la mayor parte de los casos.

  Es posible que los datos en el maestro y el esclavo difieran si una consulta se diseña de manera que la modificación de los datos sea no-determinista (generalmente no es una práctica recomendada, incluso sin hablar de replicación).

  Por ejemplo:

  - Las sentencias `CREATE ... SELECT` o `INSERT ... SELECT` que inserten valores cero o `NULL` en una columna `AUTO_INCREMENT`.
  - `DELETE`, si usted está borrando registros de una tabla que tiene claves foráneas con propiedad `ON DELETE CASCADE`.

- `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` si tiene valores de clave duplicados en los datos insertados.

Si y únicamente si las sentencias precedentes no tienen una clausula `ORDER BY` garantizado un orden determinista.

Por ejemplo, en `INSERT ... SELECT` sin un `ORDER BY`, el `SELECT` podría retornar registros en orden diferente (lo que resulta en una fila que termina teniendo diferente rango, es decir, obteniendo un número diferente en la columna `AUTO_INCREMENT`), dependiendo de las elecciones hechas por los optimizadores en el maestro y el esclavo.

Una consulta está optimizada de manera diferente en el maestro y el esclavo sólo sí:

- Los archivos utilizados en las dos consultas no son exactamente los mismos; por ejemplo se ejecutó `OPTIMIZE TABLE` en las tablas del maestro, pero no en las de los esclavos. (Para arreglar esto, `OPTIMIZE TABLE`, `ANALYZE TABLE`, y `REPAIR TABLE` se escriben en el registro binario a partir de MySQL 4.1.1).
- La tabla está almacenada utilizando un motor de almacenamiento diferente en el maestro que en el esclavo. (Es posible utilizar diferentes motores de almacenamiento en el maestro y el esclavo. Por ejemplo, puede utilizar `InnoDB` en el maestro, pero `MyISAM` en el esclavo, si el esclavo tiene menos espacio de disco disponible.)
- El tamaño de buffer de MySQL (`key_buffer_size`, y demás) es diferente en el maestro y el esclavo.
- El maestro y el esclavo ejecutan versiones de MySQL diferentes, y el código del optimizador difiere entra esas dos versiones.

Este problema podría también afectar a la restauración de bases de datos utilizando `mysqlbinlog| mysql`.

La manera más fácil de evitar este problema es añadir una clausula `ORDER BY` a las mencionadas consultas no-deterministas para asegurarse de que los registros son siempre almacenados o modificados en el mismo orden.

En versiones futuras de MySQL, añadiremos automáticamente una clausula `ORDER BY` allá donde sea necesaria.

Los siguientes problemas son conocidos y serán arreglados a su debido tiempo:

- Los nombres de archivo de los registros se basan en el nombre del servidor (si no especifica un nombre en la opción de inicio). Tiene que utilizar opciones como `--log-bin=`*`old_host_name`*`-bin` si usted cambia el nombre de su servidor. Otra opción es renombrar los antiguos archivos para reflejar el cambio al nuevo nombre (si estos son registros binarios, debería editar el archivo de índice del registro binario y arreglar los nombres también). Consulte Sección 5.3.1, “Opciones del comando `mysqld`”.
- `mysqlbinlog` no borra los archivos temporales tras un comando `LOAD DATA INFILE`. Consulte Sección 8.5, “La utilidad `mysqlbinlog` para registros binarios”.
- `RENAME` no funciona con tablas `TEMPORARY` o tablas utilizadas en una tabla `MERGE`.
- Debido a la manera en que los archivos de definición de talba son almacenados, usted no puede utilizar el carácter 255 (`CHAR(255)`) en los nombres de tabla, columna, o enumeraciones. Esto será arreglado previsiblemente en la versión 5.1, cuando implementemos un nuevo formato de archivo de definición de tablas.

- Cuando utiliza `SET CHARACTER SET`, usted no puede utilizar caracteres traducidos en los nombres de base de datos, tabla y columna.
- No puede utilizar '`_`' o '`%`' con `ESCAPE` en `LIKE ... ESCAPE`.
- Si usted tiene una columna `DECIMAL` en la que el mismo número se almacena en diferentes formatos (por ejemplo, `+01.00`, `1.00`, `01.00`), `GROUP BY` puede tratar cada valor como uno diferente.
- No puede instalar el servidor en otro directorio cuando utiliza MIT-pthreads. Debido a que este problema requiere cambios en los hilos MIT-pthreads, no es probable que lo arreglemos. Consulte Sección 2.8.5, “Notas sobre MIT-pthreads”.
- Los valores `BLOB` y `TEXT` no puede ser utilizados “de manera fiablemente” en `GROUP BY`, `ORDER BY` o `DISTINCT`. Solo los primeros `max_sort_length` se utilizan para comparar valores `BLOB` en estos casos. El valor por defecto de `max_sort_length` es 1024 y puede cambiarse en el momento de iniciar el servidor. A partir de MySQL 4.0.3, también puede cambiarse en tiempo de ejecución. En versiones más antiguas, una solución alternativa es utilizar una subcadena de caracteres. Por ejemplo:

```
SELECT DISTINCT LEFT(blob_col,2048) FROM nombre_de_tabla;
```

- Los cálculos numéricos son hechos con `BIGINT` o `DOUBLE` (ambos son normalmente de 64 bits de longitud). La precisión que usted obtenga depende de la función. La regla general es que las funciones de bit son realizadas con precisión de `BIGINT`, `IF` y `ELT()` con precisión `BIGINT` o `DOUBLE`, y el resto con precisión `DOUBLE`. Debería evitar utilizar valores sin signo largos si son más grandes de 63 bits (9223372036854775807) para cualquier cosa que no sean campos de tipo bit. La versión 4.0 de MySQL tiene mejor gestión de `BIGINT` que 3.23.
- Puede tener hasta 255 columnas `ENUM` y `SET` en una tabla.
- En las funciones `MIN()`, `MAX()`, y otras funciones de agregación, MySQL actualmente compara las columnas `ENUM` y `SET` por su valor de cadena decaracteres en vez de la posición relativa de la cadena en el conjunto.
- `mysqld_safe` redirige todos los mensajes desde `mysqld` al registro `mysqld`. Un problema con esto es que si usted ejecuta `mysqladmin refresh` para cerrar y reabrir el registro, `stdout` y `stderr` son redirigidos todavía al registro antiguo. Si usted utiliza `--log` a menudo, debería editar `mysqld_safe` para que almacene sus registros en *`host_name`*`.err` en vez de *`host_name`*`.log` de manera que pueda fácilmente recuperar el espacio borrando el registro antiguo y ejecutando `mysqladmin refresh`.
- En una sentencia `UPDATE`, las columnas son actualizadas de izquierda a drecha. Si se refiere a una columna actualizada, usted obtiene el valor actualizado en vez del original. Por ejemplo, la siguiente sentencia incrementa `KEY` en `2`, **no** `1`:

```
mysql> UPDATE nombre_de_tabla SET KEY=KEY+1,KEY=KEY+1;
```

- Puede referirse a múltiples tablas temporales en la misma sentencia, pero no puede referirse a ninguna de ellas más de una vez. Por ejemplo, lo siguiente no funciona:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- El optimizador podría gestionar `DISTINCT` de manera diferente cuando esté utilizando columnas ocultas en una join, y cuando no lo haga. En una join, las columnas ocultas se contan como parte del resultado (aunque no sean mostradas), mientras que en las consultas normales, las columnas ocultas

no participan en la comparación `DISTINCT`. Probalemente cambiemos esto en el futuro para que nunca se comparen las columnas ocultas al ejecutar `DISTINCT`.

Un ejemplo de esto es:

```
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;
```

En el segundo caso, al utilizar el servidor MySQL 3.23.x, podría obtener dos registros idénticos en el conjunto de resultados (porque los valores en las columnas `id` ocultas podrían diferir).

Tenga en cuenta que esto pasa solo en las consultas que no tienen columnas pertenecientes al `ORDER BY` en el resultado.

- Si ejecuta un `PROCEDURE` en una consulta que retorna un conjunto vacío, en algunos casos el `PROCEDURE` no transforma las columnas.
- La creación de una tabla de tipo `MERGE` no comprueba si los tablas subyacentes son de tipos compatibles.
- Si utiliza `ALTER TABLE` para añadir un índice `UNIQUE` a una tabla utilizada en una tabla `MERGE` y después añade un índice normal en la tabla `MERGE`, el orden de las claves es diferente para las tablas si había una clave no-`UNIQUE` antigua en la tabla. Esto es porque `ALTER TABLE` pone índices `UNIQUE` antes de los índices normales para poder detectar claves duplicadas tan pronto como sea posible.

Apéndice B. Credits

Tabla de contenidos

Este apéndice lista los desarrolladores, contribuidores, y gente que ha soportado y ayudado que MySQL sea lo que es hoy en día.

B.1. Desarrolladores de MySQL AB

Hay desarrolladores que están o han sido empleados por MySQL AB para trabajar en el software de base de datos de `MySQL` para que empiecen a trabajar con nosotros. Después de cada desarrollador hay una pequeña lista de trabajos de los que es responsable, o los logros conseguidos. Todos los desarrolladores están involucrados en el soporte.

- Michael (Monty) Widenius
  - Desarrollador jefe y autor principal del servidor MySQL (`mysqld`).
  - Nuevas funciones para la biblioteca de cadenas.
  - La mayoría de la biblioteca `mysys` .
  - Las bibliotecas `ISAM` y `MyISAM` (tratamiento de ficheros índice B-tree con diferente compresión y distintos formatos de registro).
  - Biblioteca `HEAP` . Un sistema de tablas en memoria con nuestro hashing dinámico completo. En uso desde 1981 y publicado desde 1984.
  - El programa `replace` (échele una ojeada, es **GENIAL**!).
  - Conector/ODBC (MyODBC), el driver ODBC para Windows.
  - Arreglando errores en MIT-pthreads para que funcione con MySQL Server. Y también Unireg, una aplicación basada en curses con varias utilidades.
  - Portar herramientas `mSQL` como `msqlperl`, `DBD`/`DBI`, y `DB2mysql`.
  - La mayoría de `crash-me` y las bases para los programas de rendimiento de MySQL.
- David Axmark
  - Escritor inicial del **Manual de referencia**, incluyendo mejoras para `texi2html`.
  - Actualización automática del sitio web desde el manual.
  - Soporte para Autoconf, Automake, y Libtool .

  - Licenciamiento.
  - Partes de todos los ficheros de texto. (Hoy en día sólo queda el `README` . El resto acabó en el manual.)
  - Testear nuevas características.
  - Nuestro experto en Software Libre .
  - Mantener la lista de mail (aunque nunca tiene tiempo para hacerlo bien...).
  - Nuestro código original de portabilidad (ahora con más de 10 años de antigüedad). Hoy en día sólo quedan algunas partes de `mysys`.
  - Alguien a quien Monty pueda llamar a media noche cuando tiene nuevas funcionalidades listas.
  - Jefe "Open Sourcerer" (relaciones con la comunidad MySQL).
- Jani Tolonen
  - `mysqlimport`
  - Muchas de las extensiones al cliente de línea de comandos.
  - `PROCEDURE ANALYSE()`
- Sinisa Milivojevic (ahora en soporte)
  - Compresión (con `zlib`) en el protocolo cliente/servidor.
  - Hashing perfecto para la fase de análisis léxico.
  - `INSERT` de múltiples registros
  - Opción `mysqldump` -e
  - `LOAD DATA LOCAL INFILE`
  - `SQL_CALC_FOUND_ROWS` `SELECT` opción
  - `--max-user-connections=...` opción
  - `net_read` y `net_write_timeout`
  - `GRANT`/`REVOKE` y `SHOW GRANTS FOR`
  - Nuevo protocolo cliente/servidor para 4.0
  - `UNION` en 4.0
  - `DELETE`/`UPDATE` de múltiples tablas.
  - Tablas derivadas en 4.1
  - Administración de recursos de usuario
  - Desarrollo inicial de la API `MySQL++` C++ y del cliente `MySQLGUI` .

- Tonu Samuel (antiguo desarrollador)
  - Interfaz VIO (la base para el protocolo de cifrado entre cliente/servidor).
  - MySQL Filesystem (forma de usar bases de datos MySQL como ficheros y directorios).
  - La expresión `CASE` .
  - Las funciones `MD5()` y `COALESCE()` .
  - Soporte `RAID` para tablas `MyISAM` .
- Sasha Pachev (antiguo desarrollador)
  - Implementación inicial de la replicación (hasta la versión 4.0).
  - `SHOW CREATE TABLE`.
  - `mysql-bench`
- Matt Wagner
  - MySQL test suite.
  - Webmaster (hasta 2002).
- Miguel Solorzano (ahora en soporte)
  - Desarrollo en Win32 y construcción de las versiones.
  - Código del servidor Windows NT.
  - WinMySQLAdmin
- Timothy Smith (ahora en soporte)
  - Soporte de conjuntos de caracteres dinámico.
  - configure, RPMs y otras partes del sistema de compilación.
  - Desarrollo inicial de `libmysqld`, el servidor empotrado.
- Sergei Golubchik
  - Búsqueda full-text .
  - Claves añadidas a la biblioteca `MERGE` .
  - Matemáticas de precisión.
- Jeremy Cole (antiguo desarrollador)
  - Comprobación y edición de este manual.
  - `ALTER TABLE ... ORDER BY ...`.
  - `UPDATE ... ORDER BY ...`.
  - `DELETE ... ORDER BY ...`.

- Indrek Siitan
  - Desarrollo/programación de nuestra interfaz Web.
  - Autor del sistema de administración del sistema de newsletters.
- Jorge del Conde (ahora en soporte)
  - `MySQLCC` (`MySQL Control Center`)
  - Desarrollo Win32
  - Implementación inicial de los portales del sitio Web .
- Venu Anuganti (antiguo desarrollador)
  - MyODBC 3.51
  - Nuevo protocolo cliente/servidor para 4.1 (para comandos preparados).
- Arjen Lentz (ahora tratando la comunidad)
  - Mantener el manual de referencia de MySQL
  - Preparación de la edición impresa del manual de O'Reilly .
- Alexander (Bar) Barkov, Alexey (Holyfoot) Botchkov, y Ramil Kalimullin
  - Data espacial (GIS) e implementación de R-Trees para 4.1
  - Códigos de caracteres y Unicode para 4.1; documentación para lo mismo
- Oleksandr (Sanja) Byelkin
  - Caché de consultas en 4.0
  - Implementación de subconsultas (4.1).
  - Implementación de vistas y tablas derivadas (5.0).
- Aleksey (Walrus) Kishkin y Alexey (Ranger) Stroganov
  - Diseño de pruebas de rendimiento y análisis.
  - Mantenimiento de la MySQL test suite.
- Zak Greant (antiguo empleado)
  - Abogado Open Source , relaciones comunidad MySQL .
- Carsten Pedersen
  - El programa de certificación MySQL .
- Lenz Grimmer
  - Ingeniería de producción (construcción y publicación).
- Peter Zaitsev

  - Funciones `SHA1()`, `AES_ENCRYPT()` y `AES_DECRYPT()` .
  - Depuración de varias características.
- Alexander (Salle) Keremidarski
  - Soporte.
  - Depuración.
- Per-Erik Martin
  - Desarrollador jefe para procedimientos almacenados (5.0).
- Jim Winstead
  - Antiguo desarrollador jefe Web.
  - Mejorar el servidor, arreglar errores.
- Mark Matthews
  - Connector/J (Java).
- Peter Gulutzan
  - Cumplimiento de estándares SQL .
  - Documentación de código/algoritmos MySQL existentes.
  - Documentación de conjuntos de caracteres.
- Guilhem Bichot
  - Replicación, desde `MySQL` versión 4.0.
  - Arregla tratamiento de exponentes para `DECIMAL`.
  - Autor de `mysql_tableinfo`.
  - Backup (en 5.1).
- Antony T. Curtis
  - Portar el software de base de datos a OS/2.
- Mikael Ronstrom
  - Mucho del trabajo inicial de NDB Cluster hasta 2000. Casi la mitad del código base en aquél entonces. Protocolo de transacciones, recuperación de nodos, reinicio del sistema y código de reinicio y partes de la funcionalidades de la API.
  - Arquitecto lider, desarrollador, depurador de NDB Cluster 1994-2004
  - Varias optimizaciones
- Jonas Oreland
  - On-line Backup

  - Entorno de test automático de MySQL Cluster
  - Biblioteca de portabilidad para NDB Cluster
  - Muchas otras cosas
- Pekka Nouisiainen
  - Implementación de índices ordenados de MySQL Cluster
  - Soporte de BLOB en MySQL Cluster
  - Soporte de conjuntos de caracteres para MySQL Cluster
- Martin Skold
  - Implementación de índice único en MySQL Cluster
  - Integración de NDB Cluster en MySQL
- Magnus Svensson
  - Marco de test para MySQL Cluster
  - Integración de NDB Cluster en MySQL
- Tomas Ulin
  - Trabajo en cambios de configuración ara instalaciones simples y uso de MySQL Cluster
- Konstantin Osipov
  - Comandos preparados.
  - Cursores.
- Dmitri Lenev
  - Soporte zona horaria.
  - Disparadores (en 5.0).

B.2. Han contribuido a crear MySQL

Aunque MySQL AB posee todos los copyrights en el `MySQL server` y `MySQL manual`, queremos reconocer a aquéllos que han hecho contribuciones de cualquier tipo a la distribución `MySQL distribution`. Los contribuidores se listan aquí, en orden algo aleatorio:

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

  Port inicial a Win32/NT.

- Per Eric Olsson

  Críticas más o menos constructivas y testeo real del formato de registro dinámico.

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

Port a Win32 con compilador Borland . `mysqlshutdown.exe` y `mysqlwatch.exe`

- David J. Hughes

  Por el esfuerzo de crear una base de datos SQL shareware. Con TcX, el predecesor de MySQL AB, comenzamos con `mSQL`, pero encontramos que no podía satisfacer nuestros propósitos, así que en su lugar escribimos una interfaz SQL para nuestra aplicación Unireg. Loa clientes `mysqladmin` y `mysql` son programas fuertemente influenciados por sus correspondientes de `mSQL` . Hemos puesto mucho esfuerzo en hacer la sintaxis MySQL un superconjunto de `mSQL`. Muchas de las ideas de la API las tomamos prestadas de `mSQL` para hacer más fácil portar aplicaciones libres `mSQL` a la MySQL API. El software MySQL no contiene ningún código de `mSQL`. Dos ficheros en la distribución (`client/insert_test.c` y `client/select_test.c`) se basan en los ficheros correspondientes (sin copyright) en la distribución `mSQL`, pero se han modificado como ejemplos mostrando los cambios necesarios para convertir código de `mSQL` a MySQL Server. (`mSQL` tiene copyrigh de David J. Hughes.)

- Patrick Lynch

  Por ayudarnos a adquirir http://www.mysql.com/.

- Fred Lindberg

  Por preparar qmail para tratar las listas de distribución de MySQL y por la ayuda obtenida en la administración de las mismas.

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

  `mysqldump` (préviamente `msqldump`, pero portado y mejorado por Monty).

- Yuri Dario

  Por mantener y extender el port MySQL OS/2 .

- Tim Bunce

  Autor de `mysqlhotcopy`.

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

  Ordenación para idioma esloveno.

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

  Las macros del conjunto de caracteres `_MB` y los conjuntos de caracteres ujis y sjis .

- Joshua Chamas `<joshua@chamas.com>`

  Base para inserciones concurrentes, sintaxis de fecha extendida, depuración para NT, y responder las listas de correo de MySQL.

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

  `mysqlaccess`, programa para mostrar los permisos de acceso para el usuario.

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

  Por uno de los primeros JDBC drivers.

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

Desarrollo de uno de los primeros JDBC drivers y otras herramientas Java relacionadas con MySQL.

- James Cooper <pixel@organic.com>

  Preparación de un histórico de lista de correo donde se puede buscar.

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

  Para xmysql, cliente X gráfico para MySQL Server.

- Doug Sisk <sisk@wix.com>

  Por proporcionar paquetes RPM de MySQL para Red Hat Linux.

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

  Por proporcionar paquetes RPM de MySQL para Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

  Por proporcionar versiones RMP de muchos clientes MySQL para Intel y SPARC.

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

  Por proporcionar versiones RPM para MySQL 3.21.

- David Sacerdote <davids@secnet.com>

  Ideas para chequeo seguro de nombres de equipo DNS.

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

  Soporte para caracteres chinos(BIG5).

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

  Mucha funcionalidad para el conjunto de caracteres chino(GBK).

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

  Ordenación checa.

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

  FROM_UNIXTIME() formato temporal, funciones ENCRYPT() y consejos de bison . Miembro activo de la lista de correo.

- Luuk de Boer <luuk@wxs.nl>

  Portó (y extendió) la suite de rendimiento a DBI/DBD. Ha sido de gran ayuda con crash-me y ejecutando pruebas de rendimiento. Algunas nuevas funciones de fecha. El script mysql_setpermission .

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

  Funciones definidas por el usuario (UDFs); CREATE FUNCTION y DROP FUNCTION.

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

La extensión `AGGREGATE` para funciones definidas por el usuario.

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

  Ayuda para preparar InstallShield para MySQL-Win32.

- Jethro Wright III `<jetman@li.net>`

  La biblioteca `libmysql.dll` .

- James Pereria `<jpereira@iafrica.com>`

  Mysqlmanager, una herramienta Win32 GUI para administrar MySQL Servers.

- Curt Sampson `<cjs@portal.ca>`

  Portar MIT-pthreads a NetBSD/Alpha y NetBSD 1.3/i386.

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

  Ejemplos en el MySQL Tutorial.

- Steve Harvey

  Por hacer `mysqlaccess` más seguro.

- Konark IA-64 Centre of Persistent Systems Private Limited

  http://www.pspl.co.in/konark/. Ayua con el port a Win64 de MySQL server.

- Albert Chin-A-Young.

  Configuración de actualizaciones para Tru64, soporte para grandes ficheros y mejor soporte TCP para los wrappers .

- John Birrell

  Emulación de `pthread_mutex()` para OS/2.

- Benjamin Pflugmann

  Tablas `MERGE` extendidas para tratar `INSERTS`. Miembro activo de la lista MySQL.

- Jocelyn Fournier

  Búsqueda y reporte de innumerables fallos (especialmente en el código de MySQL 4.1 para subconsultas).

- Marc Liyanage

  Mantenimiento de paquetes Mac OS X packages y proporciona innumerable opinión sobre cómo crear Mac OS X PKGs.

- Robert Rutherford

  Proporciona información y opinión sobre el port the QNX.

- Desarrolladores anteriores de NDB Cluster

Mucha gente se ha involucrado de muchas formas: estudiantes de verano, estudiantes de tesis, empleados.... En total más de 100 personas, demasiados para mencionar aquí. Un nombre notable es el de Ataullah Dabaghi quién hasta 1999 contribuyó acerda de un tercio del código base. También gracias especiales a los desarrolladores del sistema AXE que proporciona muchas de las bases arquitectónicas para el NDB Cluster con bloques, señales y traceo de errores. También debe darse crédito a los que creyeron en nuestras ideas lo suficiente para dar parte de su presupuesto para este desarrollo desde 1992 hasta hoy.

Otros contribuidores, buscadores de fallos y testeadores: 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.

Y muchos buscadores/arregladores de fallos de las listas de correo.

Un gran tributo para los que nos han ayudado a responder preguntas en las listas de MySQL :

- Daniel Koch <dkoch@amcity.com>

  Irix setup.

- Luuk de Boer <luuk@wxs.nl>

  Preguntas de rendimiento.

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

  DBD::mysql preguntas.

- Boyd Lynn Gerber <gerberb@zenez.com>

  Preguntas relacionadas con SCO.

- Richard Mehalick <RM186061@shellus.com>

  Preguntas relacionadas con xmysql y preguntas básicas de instalación.

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

  Preguntas acerca del módulo de configuración de Apache (log & auth), preguntas sobre PHP, sobre sintaxis de SQL y otras preguntas generales.

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

  Preguntas generales.

- Jonathan J Smith <jsmith@wtp.net>

  Preguntas específicas de SO Linux, sintaxis SQL y otras cosas que pueden necesitar algún trabajo.

- David Sklar <sklar@student.net>

  Uso de MySQL desde PHP y Perl.

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

  No especificado todavía, pero es flexible y puede tratar Linux y tal vez HP-UX. trata de que algún usuario lo use mysqlbug.

- John Lyon <jlyon@imag.net>

  Preguntas sobre instalar MySQL en Linux systems, usando ficheros .rpm o compilando desde las fuentes.

- Lorvid Ltd. <lorvid@WOLFENET.com>

  Temas de facturación/licencias/soporte/copyright .

- Patrick Sherrill <patrick@coconet.com>

  Preguntas sobre interfaz ODBC y VisualC++.

- Randy Harmon <rjharmon@uptimecomputers.com>

  Preguntas sobre DBD, Linux, y sintaxis SQL .

B.3. Documentadores y traductores

Las siguientes personas han ayudado escribiendo la documentación de MySQL y traduciendo la documentación o mensajes de error en MySQL.

- Paul DuBois

  Ayuda corrigiendo este manual. Esto incluye reescribir los intentos de Monty y David de inglés en el inglés que entiende el resto del mundo.

- Kim Aldale

  Ayuda de reescribir los primeros intentos de Monty y David de escritura en inglés.

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

  Por el primer manual MySQL. Y muchos errores de idioma arreglados en las FAQ (que se convirtieron en el manual MySQL hace mucho tiempo).

- Yan Cailin

  Primer traductor del manual de referencia MySQL en chino simplificado a inicios de 2000, en el que se basan los códigos Big5 y HK (http://mysql.hitstar.com/) . Página personal en linuxdb.yeah.net.

- Jay Flaherty <fty@mediapulse.com>

  Mucha parte de la sección Perl DBI/DBD en el manual.

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

  Pruebas de lectura del manual de referencia.

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

  Mensajes de error en francés.

- Petr Snajdr, <snajdr@pvt.net>

  Mensajes de error en checo.

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

Mensajes de error en polaco.

- Miguel Angel Fernandez Roiz

  Mensajes de error en español.

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

  Mensajes de error en noruego y testeo de MySQL 3.21.xx.

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

  Mensajes de error en ruso.

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

  Mensajes de error en italiano.

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

  Mensajes de error en en alemán.

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

  Mensajes de error en eslovaco.

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

  Mensajes de error en rumano

- Peter Feher

  Mensajes de error en húngaro.

- Roberto M. Serqueira

  Mensajes de error en portugés.

- Carsten H. Pedersen

  Mensajes de error en danés.

- Arjen G. Lentz

  Mensajes de error en alemán, complentando una traducción parcial (también trabajó en consistencia y deletreo).

B.4. Bibliotecas incluidas en MySQL y que MySQL utiliza

La siguiente es la lista de los creadores de las bibliotecas que hemos incluido en el código fuente del servidor MySQL para hacer más fácil compilar e instalar MySQL. Estamos muy agradecidos a todos los individuos que las han creado y han hecho nuestr vida mucho más fácil.

- Fred Fish

  Por su excelente biblioteca en C para depurar y tracear. Monty ha hecho un número de pequeñas mejoras a la biblioteca (velocidad y opciones adicionales).

- Richard A. O'Keefe

  Por su biblioteca de dominio público de cadenas.

- Henry Spencer

  Por su biblioteca regex, usada en `WHERE column REGEXP regexp`.

- Chris Provenzano

  Pthreads protables a nivel de usuario. Desde el copyright: este producto incluye software desarrollado por Chris Provenzano, la University of California, Berkeley, y contribuidores. Estamos usando actualmente la versión 1_60_beta6 parcheada por Monty (consulte `mit-pthreads/Changes-mysql`).

- Jean-loup Gailly and Mark Adler

  Por la biblioteca zlib (usada por MySQL en Windows).

- Bjorn Benson

  Por su paquete safe_malloc (comprobador de memoria) seguro que se usa cuando configura MySQL con `--debug`.

- Free Software Foundation

  La biblioteca `readline` (usada por el cliente de línea de comandos `mysql`).

- The NetBSD foundation

  El paquete `libedit` (usado opcionalmente por el cliente de línea de comandos `mysql`).

B.5. Paquetes que soportan MySQL

La siguiente es una lista de creadores/mantenedores de algunas de las API/paquetes/aplicaciones más importantes que usan mucha gente con MySQL.

No podemos listar cada paquete aquí porque la lista sería muy dificil de mantener. Para otros paquetes, consulte el portal de software en http://solutions.mysql.com/software/.

- Tim Bunce, Alligator Descartes

  Por la interfaz `DBD` (Perl) .

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

  Por la interfaz de Perl para MySQL Server.

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

  Por manterner el módulo Perl `DBD::mysql` .

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

  Por portar PHP para MySQL Server.

- Georg Richter

  MySQL 4.1 testeador y buscador de fallos. Nueva extensión PHP 5.0 `mysqli` (API) para usar a partir de MySQL 4.1.

- Giovanni Maruzzelli <maruzz@matrice.it>

  Por portar iODBC (Unix ODBC).

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

  Autor de LinuxThreads (usado por el MySQL Server en Linux).

B.6. Herramientas utilizadas en la creación de MySQL

La siguiente es una lista de algunas de las herramientas que hemos usado para crear MySQL. La usamos para expresar nuestro agradecimiento para los que las han creado sin los que no podríamos haber hecho de MySQL lo que es hoy en día.

- Free Software Foundation

  De los que obtuvimos un excelente compilador (`gcc`), un excelente debugger (`gdb` y la biblioteca `libc` (de la que tomamos prestado `strto.c` para que nuestro código funcionara en Linux).

- Free Software Foundation & Equipo de desarrollo de XEmacs

  Por un gran editor/entorno usado por todo el mundo en MySQL AB.

- Julian Seward

  Autor de `valgrind`, un comprobador de memoria excelente que nos ha ayudado a encontrar muchos errores en MySQL que de otro modo habrían sido muy difíciles de encontrar.

- Dorothea Lütkehaus y Andreas Zeller

  Por `DDD` (El Data Display Debugger) que es un frontend gráfico excelente para `gdb`).

B.7. Han ayudado a MySQL

Aunque MySQL AB tiene todos los copyrights de `MySQL server` y `MySQL manual`, queremos reconocer que las siguientes compañias, que nos han ayudado a financiar el desarrollo de `MySQL server`, así como nos han pagado para desarrollar nuevas características o nos han dado hardware para desarrollo de `MySQL server`.

- VA Linux / Andover.net

  Financiación de la replicación.

- NuSphere

  Edición del manual MySQL.

- Stork Design studio

  El sitio MySQL Web en uso entre 1998-2000.

- Intel

  Contribución al desarrollo en plataformas Linux y Windows.

- Compaq

  Contribución al desarrollo en Linux/Alpha.

- SWSoft

  Desarrollo de la versión de `mysqld` empotrada.

- FutureQuest

  `--skip-show-database`

Apéndice C. Historial de cambios de MySQL

Tabla de contenidos

This appendix lists the changes in the MySQL source code for version 5.1.0 and later releases. For information about changes in previous versions of the MySQL database software, see the *Manual de referencia de MySQL 4.1*, which provides coverage of the 3.22, 3.23, 4.0, and 4.1 series of releases.

We are working actively on MySQL 5.0 and 5.1, and provide only critical bugfixes for MySQL 4.1, 4.0, and MySQL 3.23. We update this section as we add new features, so that everybody can follow the development.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you can't find on our download page (http://dev.mysql.com/downloads/), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last BitKeeper ChangeSet on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

C.1. Cambios en la entrega 5.0.x (Desarrollo)

The following changelog shows what has been done in the 5.0 tree:

- Basic support for read-only server side cursors.
- Basic support for (updatable) views. See, for example, Sección 21.2, “Sintaxis de `CREATE VIEW`”.
- Basic support for stored procedures (SQL:2003 style). See Capítulo 19, *Procedimientos almacenados y funciones*.
- Initial support for rudimentary triggers.
- Added `SELECT INTO` *`list_of_vars`*, which can be of mixed (that is, global and local) types. See Sección 19.2.9.3, “La sentencia `SELECT ... INTO`”.
- Removed the update log. It is fully replaced by the binary log. If the MySQL server is started with `--log-update`, it is translated to `--log-bin` (or ignored if the server is explicitly started with `--log-bin`), and a warning message is written to the error log. Setting `SQL_LOG_UPDATE` silently sets `SQL_LOG_BIN` instead (or do nothing if the server is explicitly started with `--log-bin`).

- Support for the `ISAM` storage engine has been removed. If you have `ISAM` tables, you should convert them before upgrading. See Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0”.
- Support for `RAID` options in `MyISAM` tables has been removed. If you have tables that use these options, you should convert them before upgrading. See Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0”.
- User variable names are now case insensitive: If you do `SET @a=10;` then `SELECT @A;` now returns `10`. Case sensitivity of a variable's value depends on the collation of the value.
- Strict mode, which in essence means that you get an error instead of a warning when inserting an incorrect value into a column. See Sección 5.3.2, “El modo SQL del servidor”.
- `VARCHAR` and `VARBINARY` columns remember end space. A `VARCHAR()` or `VARBINARY` column can contain up to 65,535 characters or bytes, respectively.
- `MEMORY` (`HEAP`) tables can have `VARCHAR()` columns.
- When using a constant string or a function that generate a string result in `CREATE ... SELECT`, MySQL creates the result field based on the max_length of the string/expression:

max_length	Column type
= 0	`CHAR(0)`
< 512	`VARCHAR(max_length)`
>= 512	`TEXT`

For a full list of changes, please refer to the changelog sections for each individual 5.0.x release.

C.1.1. Cambios en la entrega 5.0.11 (todavía no liberada)

Functionality added or changed:

- `mysqldump` now dumps triggers for each dumped table. This can be suppressed with the `--skip-triggers` option. (Bug #10431)
- Added new `ER_STACK_OVERRUN_NEED_MORE` error message to indicate that, while the stack is not completly full, more stack space is required. (Bug #11213)
- `NDB`: Improved handling of the configuration variables `NoOfPagesToDiskDuringRestartACC`, `NoOfPagesToDiskAfterRestartACC`, `NoOfPagesToDiskDuringRestartTUP`, and `NoOfPagesToDiskAfterRestartTUP` should result in noticeably faster startup times for MySQL Cluster. (Bug #12149)
- Added support of where clause for queries with `FROM DUAL`. (Bug #11745)

Bugs fixed:

- Multiple `SELECT SQL_CACHE` queries in a stored procedure causes error and client hang. (Bug #6897)
- Added checks to prevent error when allocating memory when there was insufficient memory available. (Bug #7003)
- Character data truncated when GBK characters `0xA3A0` and `0xA1` are present. (Bug #11987)
- Comparisons like `SELECT "A\\" LIKE "A\\";` fail when using `SET NAMES utf8;`. (Bug #11754)
- Corrected inaccurate error message when inserting out of range data under `TRADITIONAL` SQL mode. (Bug #11546)

- When used in a `SELECT` query against a view, the `GROUP_CONCAT()` function returned only a single row. (Bug #11412)
- Calling the C API function `mysql_stmt_fetch()` after all rows of a result set were exhausted would return an error instead of `MYSQL_NO_DATA`. (Bug #11037)
- Information about a trigger was not displayed in the output of `SELECT ... FROM INFORMATION_SCHEMA.TRIGGERS` when the selected database was `INFORMATION_SCHEMA`, prior to the trigger's first invocation. (Bug #12127)
- Issuing successive `FLUSH TABLES WITH READ LOCK` would cause the `mysql` client to hang. (Bug #11934)
- In stored procedures, a cursor that fetched an empty string into a variable would set the variable to `NULL` instead. (Bug #8692)
- A trigger dependent on a feature of one `SQL_MODE` setting would cause an error when invoked after the `SQL_MODE` was changed. (Bug #5891)
- A delayed insert that would duplicate an existing record crashed the server instead. (Bug #12226)
- `ALTER TABLE` when `SQL_MODE = 'TRADITIONAL'` gave rise to an invalid error message. (Bug #11964)
- On AMD64, attempting to repair a `MyISAM` table with a full-text index would crash the server. (Bug #11684)
- The MySQL Cluster backup log was invalid where the number of Cluster nodes was not equal to a power of 2. (Bug #11675)
- `GROUP_CONCAT()` sometimes returned a result with a different collation that that of its arguments. (Bug #10201)
- The `LPAD()` and `RPAD()` functions returned the wrong length to `mysql_fetch_fields()`. (Bug #11311)
- A `UNIQUE VARCHAR` column would be mis-identified as `MUL` in table descriptions. (Bug #11227)
- Incorrect error message displayed if user attempted to create a table in a non-existing database using `CREATE` *`database_name.table_name`* syntax. (Bug #10407)
- `InnoDB`: Do not flush after each write, not even before setting up the doublewrite buffer. Flushing can be extremely slow on some systems. (Bug #12125)
- `InnoDB`: True `VARCHAR`: Return `NULL` columns in the format expected by MySQL. (Bug #12186)

C.1.2. Cambios en la entrega 5.0.10 (todavía no liberada)

Functionality added or changed:

- Security improvement: Applied a patch that addresses a `zlib` data vulnerability that could result in a buffer overflow and code execution. (CVE-2005-2096) (Bug #11844)
- The viewing of triggers and trigger metadata has been enhanced as follows:
  - An extension to the `SHOW` command has been added: `SHOW TRIGGERS` can be used to view a listing of triggers. See Sección 13.5.4.20, “Sintaxis de `SHOW TRIGGERS`” for details.

  - The `INFORMATION_SCHEMA` database now includes a `TRIGGERS` table. See Sección 22.1.16, “La tabla `INFORMATION_SCHEMA TRIGGERS`” for details. (Bug #9586)

- Triggers can now reference tables by name. See Sección 20.1, “Sintaxis de `CREATE TRIGGER`” for more information.
- The output of `perror --help` now displays the `--ndb` option. (Bug #11999)
- On Windows, the search path used by MySQL applications for `my.ini` now includes `..\my.ini` (that is, the application's parent directory, and hence, the installation directory). (Bug #10419)
- Added `mysql_get_character_set_info()` C API function for obtaining information about the default character set of the current connection.
- The bundled version of the `readline` library was upgraded to version 5.0.
- It is no longer necessary to issue an explicit `LOCK TABLES` for any tables accessed by a trigger prior to executing any statements that might invoke the trigger. (Bug #9581, Bug #8406)
- `MySQL Cluster`: A new `-p` option is available for use with the `ndb_mgmd` client. When called with this option, `ndb_mgmd` prints all configuration data to `stdout`, then exits.
- The namespace for triggers has changed. 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).

  Note: When upgrading from a previous version of MySQL 5 to MySQL 5.0.10 or newer, you must drop all triggers before upgrading and re-create them after or `DROP TRIGGER` will not work after the upgrade. (Bug #5892)

Bugs fixed:

- `NDB`: Attempting to create or drop tables during a backup would cause the cluster to shut down. (Bug #11942)
- When attempting to drop a table with a broken unique index, `NDB` failed to drop the table and erroneously report that the table was unknown. (Bug #11355)
- `SELECT ... NOT IN()` gave unexpected results when only static value present between the `()`. (Bug #11885)
- Fixed compile error when using GCC4 on AMD64. (Bug #12040)
- `NDB` ignored the `Hostname` option in the `NDBD DEFAULT` section of the Cluster configuration file. (Bug #12028)
- `SHOW PROCEDURE/FUNCTION STATUS` didn't work for users with limited access. (Bug #11577)
- MySQL server would crash is a fetch was performed after a `ROLLBACK` when cursors were involved. (Bug #10760)
- The temporary tables created by an `ALTER TABLE` on a cluster table were visible to all MySQL servers. (Bug #12055)
- `NDB_MGMD` was leaking file descriptors. (Bug #11898)
- IP addresses not shown in `ndb_mgm SHOW` command on second ndb_mgmd (or on ndb_mgmd restart). (Bug #11596)

- Functions that evaluate to constants (such as `NOW()` and `CURRENT_USER()` were being evaluated in the definition of a `VIEW` rather than included verbatim. (Bug #4663)
- Execution of `SHOW TABLES` failed to increment the `Com_show_tables` status variable. (Bug #11685)
- For execution of a stored procedure that refers to a view, changes to the view definition were not seen. The procedure continued to see the old contents of the view. (Bug #6120)
- For prepared statements, the SQL parser did not disallow '`?`' parameter markers immediately adjacent to other tokens, which could result in malformed statements in the binary log. (For example, `SELECT * FROM t WHERE? = 1` could become `SELECT * FROM t WHERE0 = 1`.) (Bug #11299)
- When two threads compete for the same table, a deadlock could occur if one thread has also a lock on another table through `LOCK TABLES` and the thread is attempting to remove the table in some manner and the other thread want locks on both tables. (Bug #10600)
- Aliasing the column names in a `VIEW` did not work when executing a `SELECT` query on the `VIEW`. (Bug #11399)
- Performing an `ORDER BY` on a `SELECT` from a `VIEW` produced unexpected results when `VIEW` and underlying table had the same column name on different columns. Bug #11709)
- The C API function `mysql_statement_reset()` did not clear error information. (Bug #11183)
- When used within a subquery, `SUBSTRING()` returned an empty string. (Bug #10269)
- Multiple-table `UPDATE` queries using `CONVERT_TZ()` would fail with an error. (Bug #9979)
- `mysql_fetch_fields()` returned incorrect length information for `MEDIUM` and `LONG TEXT` and `BLOB` columns. (Bug #9735)
- `mysqlbinlog` was failing the test suite on Windows due to `BOOL` being incorrectly cast to `INT`. (Bug #11567)
- `NDBCLuster`: Server left core files following shutdown if data nodes had failed. (Bug #11516)
- Creating a trigger in one database that references a table in another database was being allowed without generating errors. (Bug #8751)
- Duplicate trigger names were allowed within a single schema. (Bug #6182)
- Server did not accept some fully-qualified trigger names. (Bug #8758)
- The `traditional` SQL mode accepted invalid dates if the date value provided was the result of an implicit type conversion. (Bug #5906)
- The MySQL server had issues with certain combinations of basedir and datadir. (Bug #7249)
- `INFORMATION_SCHEMA.COLUMNS` had some inaccurate values for some data types. (Bug #11057)
- LIKE pattern matching using prefix index didn't return correct result. (Bug #11650)
- For several character sets, MySQL incorrectly converted the character code for the division sign to the `eucjpms` character set. (Bug #11717)
- When invoked within a view, `SUBTIME()` returned incorrect values. (Bug #11760)
- `SHOW BINARY LOGS` displayed a file size of 0 for all log files but the current one if the files were not located in the data directory. (Bug #12004)

- Server-side prepared statements failed for columns with a character set of `ucs2`. (Bug #9442)
- References to system variables in an SQL statement prepared with `PREPARE` were evaluated during `EXECUTE` to their values at prepare time, not to their values at execution time. (Bug #9359)
- For server shutdown on Windows, error messages of the form `Forcing close of thread n user: 'name'` were being written to the error log. Now connections are closed more gracefully without generating error messages. (Bug #7403)
- Increased the version number of the `libmysqlclient` shared library from 14 to 15 because it is binary incompatible with the MySQL 4.1 client library. (Bug #11893)
- A recent optimizer change caused `DELETE ... WHERE ... NOT LIKE` and `DELETE ... WHERE ... NOT BETWEEN` to not properly identify the rows to be deleted. (Bug #11853)
- Within a stored procedure that selects from a table, invoking another procedure that requires a write lock for the table caused that procedure to fail with a message that the table was read-locked. (Bug #9565)
- Within a stored procedure, selecting from a table through a view caused subsequent updates to the table to fail with a message that the table was read-locked. (Bug #9597)
- For a stored procedure defined with `SQL SECURITY DEFINER` characteristic, `CURRENT_USER()` incorrectly reported the use invoking the procedure, not the user who defined it. (Bug #7291)
- Creating a table with a `SET` or `ENUM` column with the `DEFAULT 0` clause caused a server crash if the table's character set was `utf8`. (Bug #11819)
- With strict SQL mode enabled, `ALTER TABLE` reported spurious “Invalid default value” messages for columns that had no `DEFAULT` clause. (Bug #9881)
- In SQL prepared statements, comparisons could fail for values not equally space-padded. For example, `SELECT 'a' = 'a ';` returns 1, but `PREPARE s FROM 'SELECT ?=?'; SET @a = 'a', @b = 'a '; PREPARE s FROM 'SELECT ?=?'; EXECUTE s USING @a, @b;` incorrectly returned 0. (Bug #9379)
- Labels in stored routines did not work if the character set was not `latin1`. (Bug #7088)
- Invoking the `DES_ENCRYPT()` function could cause a server crash if the server was started without the `--des-key-file` option. (Bug #11643)
- The server crashed upon execution of a statement that used a stored function indirectly (via a view) if the function was not yet in the connection-specific stored routine cache and the statement would update a `Handler_xxx` status variable. This fix allows the use of stored routines under `LOCK TABLES` without explicitly locking the `mysql.lock` table. However, you cannot use `mysql.proc` in statements that will combine locking of it with modifications for other tables. (Bug #11554)
- The server crashed when dropping a trigger that invoked a stored procedure, if the procedure was not yet in the connection-specific stored routine cache. (Bug #11889)
- Selecting the result of an aggregate function for an `ENUM` or `SET` column within a subquery could result in a server crash. (Bug #11821)
- Incorrect column values could be retrieved from views defined using statements of the form `SELECT * FROM tbl_name`. (Bug #11771)
- The `mysql.proc` table was not being created properly with the proper `utf8` character set and collation, causing server crashes for stored procedure operations if the server was using a multi-byte character set. To take advantage of the bug fix, `mysql_fix_privileges_tables` should be run to correct the structure of the `mysql.proc` table. (Bug #11365)

- Execution of a prepared statement that invoked a non-existent or dropped stored routine would crash the server. (Bug #11834)
- Executing a statement that invoked a trigger would cause problems unless a `LOCK TABLES` was first issued for any tables accessed by the trigger. **Note**: The exact nature of the problem depended upon the MySQL 5.0 release being used: prior to 5.0.3, this resulted in a crash; from 5.0.3 to 5.0.7, MySQL would issue a warning; in 5.0.9, the server would issue an error. (Bug #8406)

  The same issue caused `LOCK TABLES` to fail following `UNLOCK TABLES` if triggers were involved. (Bug #9581)
- In a shared Windows environment, MySQL could not find its configuration file unless the file was in the `C:\` directory. (Bug #5354)

C.1.3. Cambios en la entrega 5.0.9 (15 julio 2005)

Functionality added or changed:

- An attempt to create a `TIMESTAMP` column with a display width (for example, `TIMESTAMP(6)`) now results in a warning. Display widths have not been supported for `TIMESTAMP` since MySQL 4.1. (Bug #10466)
- `InnoDB`: When creating or extending an InnoDB data file, at most one megabyte at a time is allocated for initializing the file. Previously, InnoDB allocated and initialized 1 or 8 megabytes of memory, even if only a few 16-kilobyte pages were to be written. This improves the performance of `CREATE TABLE` in `innodb_file_per_table` mode.
- `InnoDB`: Various optimizations. Removed unreachable debug code from non-debug builds. Added hints for the branch predictor in `gcc`. Made assertions occupy less space.
- `InnoDB`: Make `innodb_thread_concurrency=20` by default. Bypass the concurrency checking if the setting is greater than or equal to 20.
- `InnoDB`: Make `CHECK TABLE` killable. (Bug #9730)
- Recursion in stored routines is now disabled because it was crashing the server. We plan to modify stored routines to allow this to operate safely in a future release. (Bug #11394)
- The handling of `BIT` columns has been improved, and should now be much more reliable in a number of cases. (Bug #10617, Bug #11091, Bug #11572)

Bugs fixed:

- `SHOW CREATE VIEW` did not take the `ANSI MODE` into account when quoting identifiers. (Bug #6903)
- The `mysql_config` script did not handle symbolic linking properly. (Bug #10986)
- Incorrect results when using `GROUP BY ... WITH ROLLUP` on a `VIEW`. (Bug #11639)
- Instances of the `VAR_SAMP()` function in view definitions were converted to `VARIANCE()`. This is incorrect because `VARIANCE()` is the same as `VAR_POP()`, not `VAR_SAMP()`. (Bug #10651)
- `mysqldump` failed when reloading a view if the view was defined in terms of a different view that had not yet been reloaded. `mysqldump` now creates a dummy table to handle this case. (Bug #10927)
- `mysqldump` could crash for illegal or nonexistent table names. (Bug #9358)
- The `--no-data` option for `mysqldump` was being ignored if table names were given after the database name. (Bug #9558)

- The `--master-data` option for `mysqldump` resulted in no error if the binary log was not enabled. Now an error occurs unless the `--force` option is given. (Bug #11678)
- `DES_ENCRYPT()` and `DES_DECRYPT()` require SSL support to be enabled, but were not checking for it. Checking for incorrect arguments or resource exhaustion was also improved for these functions. (Bug #10589)
- When used in joins, `SUBSTRING()` failed to truncate to zero any string values that could not be converted to numbers. (Bug #10124)
- `mysqldump --xml` did not format `NULL` column values correctly. (Bug #9657)
- There was a compression algorithm issue with `myisampack` for very large datasets (where the total size of of all records in a single column was on the order of 3 GB or more) on 64-bit platforms. (A fix for other platforms was made in MySQL 5.0.6.) (Bug #8321)
- Temporary tables were created in the data directory instead of `tmpdir`. (Bug #11440)
- MySQL would not compile correctly on QNX due to missing `rint()` function. (Bug #11544)
- A `SELECT DISTINCT col_name` would work correctly with a `MyISAM` table only when there was an index on `col_name`. (Bug #11484)
- The server would lose table-level `CREATE VIEW` and `SHOW VIEW` privileges following a `FLUSH PRIVILEGES` or server restart. (Bug #9795)
- In strict mode, an `INSERT` into a view that did not include a value for a `NOT NULL` column but that did include a `WHERE` test on the same column would succeed, This happened even though the `INSERT` should have been prevented due to the failure to supply a value for the `NOT NULL` column. (Bug #6443)
- Running a `CHECK TABLES` on multiple views crashed the server. (Bug #11337)
- When a table had a primary key containing a `BLOB` column, creation of another index failed with the error `BLOB/TEXT column used in key specification without keylength`, even when the new index did not contain a `BLOB` column. (Bug #11657)
- NDB Cluster: When trying to open a table that could not be discovered or unpacked, cluster would return error codes which the MySQL server falsely interpreted as operating system errors. (Bug #103651)
- Manually inserting a row with `host=''` into `mysql.tables_priv` and performing a `FLUSH PRIVILEGES` would cause the server to crash. (Bug #11330)
- A cursor using a query with a filter on a `DATE` or `DATETIME` column would cause the server to crash server after the data was fetched. (Bug #11172)
- Closing a cursor that was already closed would cause MySQL to hang. (Bug #9814)
- Using `CONCAT_WS` on a column set `NOT NULL` caused incorrect results when used in a `LEFT JOIN`. (Bug #11469)
- Signed `BIGINT` would not accept `-9223372036854775808` as a `DEFAULT` value. (Bug #11215)
- Views did not use indexes on all appropriate queries. (Bug #10031)
- For `MEMORY` tables, it was possible for for updates to be performed using outdated key statistics when the updates involved only very small changes in a very few rows. This resulted in the random failures of queries such as `UPDATE t SET col = col + 1 WHERE col_key = 2;` where the same query with no `WHERE` clause would succeed. (Bug #10178)

- Optimizer performed range check when comparing unsigned integers to negative constants, could cause errors. (Bug #11185)
- Wrong comparison method used in `VIEW` when relaxed date syntax used (i.e. `2005.06.10`). (Bug #11325)
- The `ENCRYPT()` and `SUBSTRING_INDEX()` functions would cause errors when used with a `VIEW`. (Bug #7024)
- Clients would hang following some errors with stored procedures. (Bug #9503)
- Combining cursors and subselects could cause server crash or memory leaks. (Bug #10736)
- If a prepared statement cursor is opened but not completely fetched, attempting to open a cursor for a second prepared statement will fail. (Bug #10794)

C.1.4. Cambios en la entrega 5.0.8 (not released)

Note: Starting with version 5.0.8, changes for MySQL Cluster can be found in the combined Change History.

Functionality added or changed:

- `MEMORY` tables now support indexes of up to 500 bytes. See Sección 14.3, “El motor de almacenamiento `MEMORY` (`HEAP`)”. (Bug #10566)
- New `SQL_MODE` - `NO_ENGINE_SUBSTITUTION` Prevents automatic substitution of storage engine when the requested storage engine is disabled or not compiled in. (Bug #6877)
- The statements `CREATE TABLE`, `TRUNCATE TABLE`, `DROP DATABASE`, and `CREATE DATABASE` cause an implicit commit. (Bug #6883)
- Expanded on information provided in general log and slow query log for prepared statements. (Bug #8367, Bug #9334)
- Where a `GROUP BY` query uses a grouping column from the query's `SELECT` clause, MySQL now issues a warning. This is because the SQL standard states that any grouping column must unambiguously reference a column of the table resulting from the query's `FROM` clause, and allowing columns from the `SELECT` clause to be used as grouping columns is a MySQL extension to the standard.

  By way of example, consider the following table:

```
CREATE TABLE users (
  userid INT NOT NULL PRIMARY KEY,
  username VARCHAR(25),
  usergroupid INT NOT NULL
);
```

  MySQL allows you to use the alias in this query:

```
SELECT usergroupid AS id, COUNT(userid) AS number_of_users
FROM users
GROUP BY id;
```

  However, the SQL standard requires that the column name be used, as shown here:

```
SELECT usergroupid AS id, COUNT(userid) AS number_of_users
FROM users
```

```
GROUP BY usergroupid;
```

Queries such as the first of the two shown above will continue to be supported in MySQL; however, beginning with MySQL 5.0.8, using a column alias in this fashion will generate a warning. Note that in the event of a collision between column names and/or aliases used in joins, MySQL attempts to resolve the conflict by giving preference to columns arising from tables named in the query's `FROM` clause. (Bug #11211)

- The granting or revocation of privileges on a stored routine is no longer performed when running the server with `--skip-grant-tables` even after the statement `SET @@global.automatic_sp_privileges=1;` has been executed. (Bug #9993)
- Added support for `B'10'` syntax for bit literal. (Bug #10650)

Bugs fixed:

- **Security fix**: On Windows systems, a user with any of the following privileges
  - `REFERENCES`
  - `CREATE TEMPORARY TABLES`
  - `GRANT OPTION`
  - `CREATE`
  - `SELECT`

  on `*.*` could crash `mysqld` by issuing a `USE LPT1;` or `USE PRN;` command. In addition, any of the commands `USE NUL;`, `USE CON;`, `USE COM1;`, or `USE AUX;` would report success even though the database was not in fact changed. **Note**: Although this bug was thought to be fixed previously, it was later discovered to be present in the MySQL 5.0.7-beta release for Windows. (Bug #9148, CVE-2005-0799)
- A `CREATE TABLE db_name.tbl_name LIKE ...` statement would crash the server when no database was selected. (Bug #11028)
- `SELECT DISTINCT` queries or `GROUP BY` queries without `MIN()` or `MAX()` could return inconsistent results for indexed columns. (Bug #11044)
- The `SHOW INSTANCE OPTIONS` command in MySQL Instance Manager displayed option values incorrectly for options for which no value had been given. (Bug #11200)
- An outer join with an empty derived table (a result from a subquery) returned no result. (Bug #11284)
- An outer join with an `ON` condition that evaluated to false could return an incorrect result. (Bug #11285)
- `mysqld_safe` would sometimes fail to remove the pid file for the old `mysql` process after a crash. As a result, the server would fail to start due to a false `A mysqld process already exists...` error. (Bug #11122)
- `CAST( ... AS DECIMAL) didn't work for strings. (Bug #11283)`
- `NULLIF()` function could produce incorrect results if first argument is `NULL`. (Bug #11142)
- Setting `@@SQL_MODE = NULL` caused an erroneous error message. (Bug #10732)
- Converting a `VARCHAR` column having an index to a different type (such as `TINYTEXT`) gave rise to an incorrect error message. (Bug #10543)

Note that this bugfix induces a slight change in the behaviour of indexes: If an index is defined to be the same length as a field (or is left to default to that field's length), and the length of the field is later changed, then the index will adopt the new length of the field. Previously, the size of the index did not change for some field types (such as `VARCHAR`) when the field type was changed.

- `sql_data_access` column of `routines` table of `INFORMATION_SCHEMA` was empty. (Bug #11055)
- A `CAST()` value could not be included in a `VIEW`. (Bug #11387)
- Server crashed when using `GROUP BY` on the result of a `DIV` operation on a `DATETIME` value. (Bug #11385)
- Possible `NULL` values in `BLOB` columns could crash the server when a `BLOB` was used in a `GROUP BY` query. (Bug #11295)
- Fixed 64 bit compiler warning for packet length in replication. (Bug #11064)
- Multiple range accesses in a subquery cause server crash. (Bug #11487)
- An issue with index merging could cause suboptimal index merge plans to be chosen when searching by indexes created on `DATE` columns. The same issue caused the InnoDB storage engine to issue the warning `using a partial-field key prefix in search`. (Bug #8441)
- The `mysqlhotcopy` script was not parsing the output of `SHOW SLAVE STATUS` correctly when called with the `--record_log_pos` option. (Bug #7967)
- `SELECT * FROM table` returned incorrect results when called from a stored procedure, where `table` had a primary key. (Bug #10136)
- When used in defining a view, the `TIME_FORMAT()` function failed with calculated values, for example, when passed the value returned by `SEC_TO_TIME()`. (Bug #7521)
- `SELECT DISTINCT ... GROUP BY constant` returned multiple rows (it should return a single row). (Bug #8614)
- `INSERT INTO SELECT FROM view` produced incorrect result when using `ORDER BY`. (Bug #11298)
- Fixed hang/crash with Boolean full-text search where a query contained more query terms that one-third of the query length (it could be achieved with truncation operator: 'a*b*c*d*'). (Bug #7858)
- Fixed column name generation in `VIEW` creation to ensure there are no duplicate column names. (Bug #7448)
- An `ORDER BY` clause sometimes had no effect on the ordering of a result when selecting specific columns (as opposed to using `SELECT *`) from a view. (Bug #7422)
- Some data definition statements (`CREATE TABLE` where the table was not a temporary table, `TRUNCATE TABLE`, `DROP DATABASE`, and `CREATE DATABASE`) were not being written to the binary log after a `ROLLBACK`. This also caused problems with replication. (Bug #6883)
- Calling a stored procedure that made use of an `INSERT ... SELECT ... UNION SELECT ...` query caused a server crash. (Bug #11060)
- Selecting from a view defined using `SELECT SUM(DISTINCT ...)` caused an error; attempting to execute a `SELECT * FROM INFORMATION_SCHEMA.TABLES` query after defining such a view crashed the server. (Bug #7015)
- The `mysql` client would output a prompt twice following input of very long strings, because it incorrectly assumed that a call to the `_cgets()` function would clear the input buffer. (Bug #10840)

- A three byte buffer overflow in the client functions caused improper exiting of the client when reading a command from the user. (Bug #10841)
- Fixed a problem where a stored procedure caused a server crash if the query cache was enabled. (Bug #9715)
- `SHOW CREATE DATABASE INFORMATION_SCHEMA` returned an “unknown database” error. (Bug #9434)
- Corrected a problem with `IFNULL()` returning an incorrect result on 64-bit systems. (Bug #11235)
- Fixed a problem resolving table names with `lower_case_table_names=2` when the table name lettercase differed in the `FROM` and `WHERE` clauses. (Bug #9500)
- Fixed server crash due to some internal functions not taking into account that for multi-byte character sets, `CHAR` columns could exceed 255 bytes and `VARCHAR` columns could exceed 65,535 bytes. (Bug #11167)
- Fixed locking problems for multiple-statement `DELETE` statements performed within a stored routine, such as incorrectly locking a to-be-modified table with a read lock rather than a write lock. (Bug #11158)
- Fixed a portability problem testing for `crypt()` support that caused compilation problems when using OpenSSL/yaSSL on HP-UX and Mac OS X. (Bug #10675, Bug #11150)
- The hostname cache was not working. (Bug #10931)
- On Windows, `mysqlshow` did not interpret wildcard characters properly if they were given in the table name argument. (Bug #10947)
- Using `PREPARE` to prepare a statement that invoked a stored routine that deallocated the prepared statement caused a server crash. This is prevented by disabling dynamic SQL within stored routines. (Bug #10975)
- Default hostname for MySQL server was always `mysql`. (Bug #11174)
- Using `PREPARE` to prepare a statement that invoked a stored routine that executed the prepared statement caused a `Packets out of order` error the second time the routine was invoked. This is prevented by disabling dynamic SQL within stored routines. (Bug #7115)
- Using prepared statements within a stored routine (`PREPARE`, `EXECUTE`, `DEALLOCATE`) could cause the client connection to be dropped after the routine returned. This is prevented by disabling dynamic SQL within stored routines. (Bug #10605)
- When using a cursor with a prepared statement, the first execution returned the correct result but was not cleaned up properly, causing subsequent executions to return incorrect results. (Bug #10729)
- MySQL Cluster: Connections between data nodes and management nodes were not being closed following shutdown of `ndb_mgmd`. (Bug #11132)
- MySQL Cluster: `mysqld` processes would not reconnect to cluster following restart of `ndb_mgmd`. (Bug #11221)
- MySQL Cluster: Fixed problem whereby data nodes would fail to restart on 64-bit Solaris (Bug #9025)
- MySQL Cluster: Calling `ndb_select_count()` crashed the cluster when running on Red Hat Enterprise 4/64-bit/Opteron. (Bug #10058)
- MySQL Cluster: Insert records were incorrectly applied by `ndb_restore`, thus making restoration from backup inconsistent if the binlog contained inserts. (Bug #11166)

- MySQL Cluster: Cluster would time out and crash after first query on 64-bit Solaris 9. (Bug #8918)
- MySQL Cluster: `ndb_mgm` client `show` command displayed incorrect output after master data node failure. (Bug #11050)
- MySQL Cluster: A delete performed as part of a transaction caused an erroneous result. (Bug #11133)
- MySQL Cluster: Not allowing sufficient parallelism in cluster configuration (e.g. `NoOfTransactions` too small) caused `ndb_restore` to fail without providing any error messages. (Bug #10294)
- MySQL Cluster: When using dynamically allocated ports on Linux, cluster would hang on initial startup. (Bug #10893)
- MySQL Cluster: Setting TransactionInactiveTimeout= 0 did not result in an infinite timeout. (Bug #11290)
- `InnoDB`: Enforce maximum `CHAR_LENGTH()` of UTF-8 data in `ON UPDATE CASCADE`. (Bug #10409)
- `InnoDB`: Pad UTF-8 variable-length `CHAR` columns with `0x20`. Pad UCS2 `CHAR` columns with `0x0020`. (Bug #10511)

C.1.5. Cambios en la entrega 5.0.7 (10 June 2005)

Functionality added or changed:

- Added `mysql_set_character_set()` C API function for setting the default character set of the current connection. This allows clients to affect the character set used by `mysql_real_escape_string()`. (Bug #8317)
- The behaviour of the `Last_query_cost` system variable has been changed. The default value is now 0 (rather than -1) and it now has session-level scope (rather than being global). See Sección 5.3.4, “Variables de estado del servidor” for additional information.
- All characters occuring on the same line following the `DELIMITER` keyword will be set as delimiter. For example, `DELIMITER ;` will set `;` as the delimiter. This behavior is now consistent between MySQL 5.1 and MySQL 5.0. (Bug #9879)
- The `table`, `type`, and `rows` columns of `EXPLAIN` output can now be `NULL`. This is required for using `EXPLAIN` on `SELECT` queries that use no tables (i.e. `EXPLAIN SELECT 1`). (Bug #9899)
- Placeholders now can be used for `LIMIT` in prepared statements. (Bug #7306)
- `SHOW BINARY LOGS` now displays a `File_size` column that indicates the size of each file.
- The `--delayed-insert` option for `mysqldump` has been disabled to avoid causing problems with storage engines that do not support `INSERT DELAYED`. (Bug #7815)
- Improved the optimizer to be able to use indexes for expressions of the form *`indexed_col`* `NOT IN (`*`val1, val2, ...`*`)` and *`indexed_col`* `NOT BETWEEN` *`val1`* `AND` *`val2`*.. (Bug #10561)
- Removed `mysqlshutdown.exe` and `mysqlwatch.exe` from the Windows “No Installer” distribution (they had already been removed from the “With Installer” distribution before). Removed those programs from the source distribution.
- Removed `WinMySQLAdmin` from the source distribution and from the “No Installer” Windows distribution (it had already been removed from the “With Installer” distribution before).
- `InnoDB`: In stored procedures and functions, `InnoDB` no longer takes full explicit table locks for every involved table. Only `intention' locks are taken, similar to those in the execution of an ordinary SQL statement. This greatly reduces the number of deadlocks.

Bugs fixed:

- **Security update**: A user with limited privileges could obtain information about the privileges of other users by querying objects in the `INFORMATION_SCHEMA` database for which that user did not have the requisite privileges. (Bug #10964)
- Triggers with dropped functions caused crashes. (Bug #5893)
- Failure of a `BEFORE` trigger did not prevent the triggering statement from performing its operation on the row for which the trigger error occurred. Now the triggering statement fails as described in Sección 20.3, “Utilización de disparadores”. (Bug #10902)
- Issuing a write lock for a table from one client prevented other clients from accessing the table's metadata. For example, if one client issued a `LOCK TABLES mydb.mytable WRITE`, then a second client attempting to execute a `USE mydb;` would hang. (Bug #9998)
- The `LAST_DAY()` failed to return `NULL` when supplied with an invalid argument. See Sección 12.5, “Funciones de fecha y hora”. (Bug #10568)
- The functions `COALESCE()`, `IF()`, and `IFNULL()` performed incorrect conversions of their arguments. (Bug #9939)
- The `TIME_FORMAT()` function returned incorrect results with some format specifiers. See Sección 12.5, “Funciones de fecha y hora”. (Bug #10590)
- Dropping stored routines when the MySQL server had been started with `--skip-grant-tables` generated extraneous warnings. (Bug #9993)
- A problem with the `my_global.h` file caused compilation of MySQL to fail on single-processor Linux systems running 2.6 kernels. (Bug #10364)
- The ucs2_turkish_ci collation failed with upper('i'). UPPER/LOWER now can return a string with different length. (Bug #8610)
- OPTIMIZE of InnoDB table does not return 'Table is full' if out of tablespace. (Bug #8135)
- GROUP BY queries with ROLLUP returned wrong results for expressions containing group by columns. (Bug #7894)
- Fixed bug in `FIELD()` function where value list contains `NULL`. (Bug #10944)
- Corrected a problem where an incorrect column type was returned in the result set metadata when using a prepared `SELECT DISTINCT` statement to select from a view. (Bug #11111)
- Fixed bug in the MySQL Instance manager that caused the version to always be `unknown` when `SHOW INSTANCE STATUS` was issued. (Bug #10229)
- Using `ORDER BY` to sort the results of an `IF()` that contained a `FROM_UNIXTIME()` expression returned incorrect results due to integer overflow. (Bug #9669)
- Fixed a server crash resulting from accessing `InnoDB` tables within stored functions. This is handled by prohibiting statements that do an explicit or explicit commit or rollback within stored functions or triggers. (Bug #10015)
- Fixed a server crash resulting from the second invocation of a stored procedure that selected from a view defined as a join that used `ON` in the join conditions. (Bug #6866)
- Using `ALTER TABLE` for a table that had a trigger caused a crash when executing a statement that activated the trigger, and also a crash later with `USE db_name` for the database containing the table. (Bug #5894)

- Fixed a server crash resulting from an attempt to allocate too much memory when `GROUP BY blob_col` and `COUNT(DISTINCT)` were used. (Bug #11088)
- Fixed a portability problem for compiling on Windows with Visual Studio 6. (Bug #11153)
- The incorrect sequence of statements `HANDLER tbl_name READ index_name NEXT` without a preceding `HANDLER tbl_name READ index_name = (value_list)` for an `InnoDB` table resulted in a server crash rather than an error. (Bug #5373)
- On Windows, with `lower_case_table_names` set to 2, using `ALTER TABLE` to alter a `MEMORY` or `InnoDB` table that had a mixed-case name also improperly changed the name to lowercase. (Bug #9660)
- The server timed out SSL connections too quickly on Windows. (Bug #8572)
- Executing `LOAD INDEX INTO CACHE` for a table while other threads where selecting from the table caused a deadlock. (Bug #10602)
- Fixed a server crash resulting from `CREATE TABLE ... SELECT` that selected from a table being altered by `ALTER TABLE`. (Bug #10224)
- The `FEDERATED` storage engine properly handled outer joins, but not inner joins. (Bug #10848)
- Consistently report `INFORMATION_SCHEMA` table names in uppercase in `SHOW TABLE STATUS` output. (Bug #10059)
- Fixed a failure of `WITH ROLLUP` to sum values properly. (Bug #10982)
- Triggers were not being activated for multiple-table `UPDATE` or `DELETE` statements. (Bug #5860)
- `INSERT BEFORE` triggers were not being activated for `INSERT ... SELECT` statements. (Bug #6812)
- `INSERT BEFORE` triggers were not being activated for implicit inserts (`LOAD DATA`). (Bug #8755)
- If a stored function contained a `FLUSH` statement, the function crashed when invoked. `FLUSH` now is disallowed within stored functions. (Bug #8409)
- Multiple-row `REPLACE` could fail on a duplicate-key error when having one `AUTO_INCREMENT` key and one unique key. (Bug #11080)
- Fixed a server crash resulting from invalid string pointer when inserting into the `mysql.host` table. (Bug #10181)
- Multiple-table `DELETE` did always delete on the fly from the first table that was to be deleted from. In some cases, when using many tables and it was necessary to access the same row twice in the first table, we could miss some rows-to-be-deleted from other tables. This is now fixed.
- The `mysql_next_result()` function could hang if you were executing many statements in a `mysql_real_query()` call and one of those statements raised an error. (Bug #9992)
- The combination of `COUNT()`, `DISTINCT`, and `CONCAT()` sometimes triggered a memory deallocation bug on Windows resulting in a server crash. (Bug #9593)
- `InnoDB`: Do very fast shutdown only if `innodb_fast_shutdown=2`, but wait for threads to exit and release allocated memory if `innodb_fast_shutdown=1`. Starting with MySQL/InnoDB 5.0.5, InnoDB would do brutal shutdown also when `innodb_fast_shutdown=1`. (Bug #9673)
- `InnoDB`: Fixed `InnoDB: Error: stored_select_lock_type is 0 inside ::start_stmt()!` in a stored procedure call if `innodb_locks_unsafe_for_binlog` was set in `my.cnf`. (Bug #10746)

- `InnoDB`: Fixed a duplicate key error that occurred with `REPLACE` in a table with an `AUTO-INC` column. (Bug #11005)
- Fixed that MySQL would pass a wrong key length to storage engines in `MIN()`. This could cause warnings `InnoDB: Warning: using a partial-field key prefix in search.` in the `.err` log. (Bug #11039)
- Fixed a server crash for `INSERT` or `UPDATE` when the `WHERE` clause contained a correlated subquery that referred to a column of the table being modified. (Bug #6384)
- Fixed a problem causing an incorrect result for columns that include an aggregate function as part of an expression when `WITH ROLLUP` is added to `GROUP BY`. (Bug #7914)
- Fixed a problem with returning an incorrect result from a view that selected a `COALESCE()` expression from the result of an outer join. (Bug #9938)
- MySQL was adding a `DEFAULT` clause to `ENUM` columns that included no explicit `DEFAULT` and were defined as `NOT NULL`. (This is supposed to happen only for columns that are `NULL`.) (Bug #6267)
- Corrected inappropriate error messages that were displayed when attempting to set the read-only `warning_count` and `error_count` system variables. (Bug #10339)

C.1.6. Cambios en la entrega 5.0.6 (26 May 2005)

Functionality added or changed:

- **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. Dump such tables with `mysqldump` before upgrading, and then reload them after upgrading. (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.) (Bug #10465, Bug #10625)
- Added the `div_precision_increment` system variable, which indicates the number of digits of precision by which to increase the result of division operations performed with the `/` operator.
- Added the `log_bin_trust_routine_creators` system variable, which applies when binary logging is enabled. It controls whether stored routine creators can be trusted not to create stored routines that will cause unsafe events to be written to the binary log.
- Added the `--log-bin-trust-routine-creators` server option for setting the `log_bin_trust_routine_creators` system variable from the command line.
- Implemented the `STMT_ATTR_PREFETCH_ROWS` option for the `mysql_stmt_attr_set()` C API function. This sets how many rows to fetch at a time when using cursors with prepared statements.
- The `GRANT` and `REVOKE` statements now support an *`object_type`* clause to be used for disambiguating whether the grant object is a table, a stored function, or a stored procedure. Use of this clause requires that you upgrade your grant tables. See Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”. (Bug #10246)
- Added `REFERENCED_TABLE_SCHEMA`, `REFERENCED_TABLE_NAME`, and `REFERENCED_COLUMN_NAME` columns to the `KEY_COLUMN_USAGE` table of `INFORMATION_SCHEMA`. (Bug #9587)
- Added a `--show-warnings` option to `mysql` to cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode. In interactive mode, `\w` and `\W` may be used to enable and disable warning display. (Bug #8684)
- Removed a limitation that prevented use of FIFOs as logging targets (such as for the general query log). This modification *does not apply* to the binary log and the relay log. (Bug #8271)

- Added a `--debug` option to `my_print_defaults`.
- When the server cannot read a table because it cannot read the `.frm` file, print a message that the table was created with a different version of MySQL. (This can happen if you create tables that use new features and then downgrade to an older version of MySQL.) (Bug #10435)
- `SHOW VARIABLES` now shows the `slave_compressed_protocol`, `slave_load_tmpdir` and `slave_skip_errors` system variables. (Bug #7800)
- Removed unused system variable `myisam_max_extra_sort_file_size`.
- Changed default value of `myisam_data_pointer_size` from 4 to 6. This allows us to avoid `table is full` errors for most cases.
- The variable `concurrent_insert` now takes 3 values. Setting this to 2 changes MyISAM to do concurrent inserts to end of table if table is in use by another thread.
- New `/*>` prompt for `mysql`. This prompt indicates that a `/* ... */` comment was begun on an earlier line and the closing `*/` sequence has not yet been seen. (Bug #9186)
- If strict SQL mode is enabled, `VARCHAR` and `VARBINARY` columns with a length greater than 65,535 no longer are silently converted to `TEXT` or `BLOB` columns. Instead, an error occurs. (Bug #8295, Bug #8296)
- The `INFORMATION_SCHEMA.SCHEMATA` table now has a `DEFAULT_COLLATION_NAME` column. (Bug #8998)
- `InnoDB`: When the maximum length of `SHOW INNODB STATUS` output would be exceeded, truncate the beginning of the list of active transactions, instead of truncating the end of the output. (Bug #5436)
- `InnoDB`: If `innodb_locks_unsafe_for_binlog` option is set and the isolation level of the transaction is not set to serializable then `InnoDB` uses a consistent read for select in clauses like `INSERT INTO ... SELECT` and `UPDATE ... (SELECT)` that do not specify `FOR UPDATE` or `IN SHARE MODE`. Thus no locks are set to rows read from selected table.
- Updated version of `libedit` to 2.9. (Bug #2596)
- Removed `mysqlshutdown.exe` and `mysqlwatch.exe` from the Windows “With Installer” distribution.

Bugs fixed:

- An error in the implementation of the `MyISAM` compression algorithm caused `myisampack` to fail with very large sets of data (total size of all the records in a single column needed to be >= 3 GB in order to trigger this issue). (Bug #8321)
- Statements that create and use stored routines were not being written to the binary log, which affects replication and data recovery options. (Bug #2610) Stored routine-related statements now are logged, subject to the issues and limitations discussed in Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”
- Disabled binary logging within stored routines to avoid writing spurious extra statements to the binary log. For example, if a routine `p()` executes an `INSERT` statement, then for `CALL p()`, the `CALL` statement appears in the binary log, but not the `INSERT` statement. (Bug #9100)
- Statements that create and drop triggers were not being written to the binary log, which affects replication and data recovery options. (Bug #10417) Trigger-related statements now are logged, subject to the issues and limitations discussed in Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”

- The `mysql_stmt_execute()` and `mysql_stmt_reset()` C API functions now close any cursor that is open for the statement, which prevents a server crash. (Bug #9478)
- The `mysql_stmt_attr_set()` C API function now returns an error for option values that are defined in `mysql.h` but not yet implemented, such as `CURSOR_TYPE_SCROLLABLE`. (Bug #9643)
- `MERGE` tables could fail on Windows due to incorrect interpretation of pathname separator characters for filenames in the `.MRG` file. (Bug #10687)
- Fixed a server crash for `INSERT ... ON DUPLICATE KEY UPDATE` with `MERGE` tables, which do not have unique indexes. (Bug #10400)
- Fix `FORMAT()` to do better rounding for double values (for example, `FORMAT(4.55,1)` returns `4.6`, not `4.5`). (Bug #9060)
- Disallow use of `SESSION` or `GLOBAL` for user variables or local variables in stored routines. (Bug #9286)
- Fixed a server crash when using `GROUP BY ... WITH ROLLUP` on an indexed column in an `InnoDB` table. (Bug #9798)
- In strict SQL mode, some assignments to numeric columns that should have been rejected were not (such as the result of an arithmetic expression or an explicit `CAST()` operation). (Bug #6961)
- `CREATE TABLE t AS SELECT UUID()` created a `VARCHAR(12)` column, which is too small to hold the 36-character result from `UUID()`. (Bug #9535)
- Fixed a server crash in the `BLACKHOLE` storage engine. (Bug #10175)
- Fixed a server crash resulting from repeated calls to `ABS()` when the argument evaluated to `NULL`. (Bug #10599)
- For a user-defined function invoked from within a prepared statement, the UDF's initialization routine was invoked for each execution of the statement, but the deinitialization routine was not. (It was invoked only when the statement was closed.) Similarly, when invoking a UDF from within a trigger, the initialization routine was invoked but the deinitialization routine was not. For UDFs that have an expensive deinit function (such as `myperl`, this bugfix will have negative performance consequences. (Bug #9913)
- Portability fix for Cygwin: Don't use `#pragma interface` in source files. (Bug #10241)
- Fix `CREATE TABLE ... LIKE` to work when `lower_case_table_names` is set on a case-sensitive filesystem and the source table name is not given in lowercase. (Bug #9761)
- Fixed a server crash resulting from a `CHECK TABLE` statement where the arguments were a view name followed by a table name. (Bug #9897)
- Within a stored procedure, attempting to update a view defined as an inner join failed with a `Table 'tbl_name' was locked with a READ lock and can't be updated` error. (Bug #9481)
- Fixed a problem with `INFORMATION_SCHEMA` tables being inaccessible depending on lettercase used to refer to them. (Bug #10018)
- `my_print_defaults` was ignoring the `--defaults-extra-file` option or crashing when the option was given. (Bug #9136, Bug #9851)
- The `INFORMATION_SCHEMA.COLUMNS` table was missing columns of views for which the user has access. (Bug #9838)
- Fixed a `mysqldump` crash that occurred with the `--complete-insert` option when dumping tables with a large number of long column names. (Bug #10286)

- Corrected a problem where `DEFAULT` values where not assigned properly to `BIT(1)` or `CHAR(1)` columns if certain other columns preceded them in the table definition. (Bug #10179)
- For `MERGE` tables, avoid writing absolute pathnames in the `.MRG` file for the names of the constituent `MyISAM` tables so that if the data directory is moved, `MERGE` tables will not break. For `mysqld`, write just the `MyISAM` table name if it is in the same database as the `MERGE` table, and a path relative to the data directory otherwise. For the embedded servers, absolute pathnames may still be used. (Bug #5964)
- Corrected a problem resolving outer column references in correlated subqueries when using the prepared statements. (Bug #10041)
- Corrected the error message for exceeding the `MAX_CONNECTIONS_PER_HOUR` limit to say `max_connections_per_hour` instead of `max_connections`. (Bug #9947)
- Fixed incorrect memory block allocation for the query cache in the embedded server. (Bug #9549)
- Corrected an inability to select from a view within a stored procedure. (Bug #9758)
- Fixed a server crash resulting from use of `AVG(DISTINCT)` with `GROUP BY ... WITH ROLLUP`. (Bug #9799)
- Fixed a server crash resulting from use of `DISTINCT AVG()` with `GROUP BY ... WITH ROLLUP`. (Bug #9800)
- Fixed a server crash resulting from use of a `CHAR` or `VARCHAR` column with `MIN()` or `MAX()` and `GROUP BY ... WITH ROLLUP`. (Bug #9820)
- Fixed a server crash resulting from use of `SELECT DISTINCT` with a prepared statement that uses a cursor. (Bug #9520)
- Fixed server crash resulting from multiple calls to a stored procedure that assigned the result of a subquery to a variable or compared it to a value with `IN`. (Bug #5963)
- Selecting from a single-table view defined on multiple-table views caused a server crash. (Bug #8528)
- If the file named by a `--defaults-extra-file` option does not exist or is otherwise inaccessible, an error now occurs. (Bug #5056)
- `net_read_timeout` and `net_write_timeout` were not being respected on Windows. (Bug #9721)
- `SELECT` from `INFORMATION_SCHEMA` tables failed if the statement has a `GROUP BY` clause and an aggregate function in the select list. (Bug #9404)
- Corrected some failures of prepared statements for SQL (`PREPARE` plus `EXECUTE`) to return all rows for some `SELECT` statements. (Bug #9096, Bug #9777)
- Remove extra slashes in `--tmpdir` value (for example, convert `/var//tmp` to `/var/tmp`, because they caused various errors. (Bug #8497)
- Added `Create_routine_priv`, `Alter_routine_priv`, and `Execute_priv` privileges to the `mysql.host` privilege table. (They had been added to `mysql.db` in MySQL 5.0.3 but not to the `host` table.) (Bug #8166)
- Fixed `configure` to properly recognize whether NTPL is available on Linux. (Bug #2173)
- Incomplete results were returned from `INFORMATION_SCHEMA.COLUMNS` for `INFORMATION_SCHEMA` tables for non-`root` users. (Bug #10261)
- Fixed a portability problem in compiling `mysql.cc` with `VC++` on Windows. (Bug #10245)

- `SELECT 0/0` returned `0` rather than `NULL`. (Bug #10404)
- `MAX()` for an `INT UNSIGNED` (unsigned 4-byte integer) column could return negative values if the column contained values larger than 2^31. (Bug #9298)
- `SHOW CREATE VIEW` got confused and could not find the view if there was a temporary table with the same name as the view. (Bug #8921)
- Fixed a deadlock resulting from use of `FLUSH TABLES WITH READ LOCK` while an `INSERT DELAYED` statement is in progress. (Bug #7823)
- The optimizer was choosing suboptimal execution plans for certain outer joins where the right table of a left join (or left table of a right join) had both `ON` and `WHERE` conditions. (Bug #10162)
- `RENAME TABLE` for an `ARCHIVE` table failed if the `.arn` file was not present. (Bug #9911)
- Invoking a stored function that executed a `SHOW` statement resulted in a server crash. (Bug #8408)
- Fixed problems with static variables and do not link with `libsupc++` to allow building on FreeBSD 5.3. (Bug #9714)
- Fixed some `awk` script portability problems in `cmd-line-utils/libedit/makelist.sh`. (Bug #9954)
- Fixed a problem with mishandling of `NULL` key parts in hash indexes on `VARCHAR` columns, resulting in incorrect query results. (Bug #9489, Bug #10176)
- `InnoDB`: Fixed a critical bug in InnoDB `AUTO_INCREMENT`: it could assign the same value for several rows. (Bug #10359) `InnoDB`: All InnoDB bug fixes from 4.1.12 and earlier versions, and also the fixes to bugs #10335 and #10607 listed in the 4.1.13 change notes.

C.1.7. Cambios en la entrega 5.0.5 (not released)

No public release of MySQL 5.0.5 was made. The changes described in this section are available in MySQL 5.0.6.

Functionality added or changed:

- Added support for the `BIT` data type to the `MEMORY`, `InnoDB`, and `BDB` storage engines.
- `SHOW VARIABLES` no longer displays the deprecated `log_update` system variable. (Bug #9738)
- The behavior controlled by the `--innodb-fast-shutdown` option now can be changed at runtime by setting the value of the global `innodb_fast_shutdown` system variable. It now accepts values 0, 1 and 2 (except on Netware where 2 is disabled). If set to 2, then when the MySQL server shuts down, `InnoDB` will just flush its logs and shut down brutally (and quickly) as if a MySQL crash had occurred; no committed transaction will be lost, but a crash recovery will be done at next startup.

Bugs fixed:

- **Security fix:** If `mysqld` was started with `--user=`*`non_existent_user`*, it would run using the privileges of the account it was invoked from, even if that was `root`. (Bug #9833)
- Corrected a failure to resolve a column reference correctly for a `LEFT JOIN` that compared a join column to an `IN` subquery. (Bug #9338)
- Fixed a problem where, after an internal temporary table in memory became too large and had to be converted to an on-disk table, the error indicator was not cleared and the query failed with error 1023 (`Can't find record in ''`). (Bug #9703)

- Multiple-table updates could produce spurious data-truncation warnings if they used a join across columns that are indexed using a column prefix. (Bug #9103)
- Fixed a string-length comparison problem that caused `mysql` to fail loading dump files containing certain '`\`'-sequences. (Bug #9756)
- Fixed a failure to resolve a column reference properly when an outer join involving a view contained a subquery and the column was used in the subquery and the outer query. (Bug #6106, Bug #6107)
- Use of a subquery that used `WITH ROLLUP` in the `FROM` clause of the main query sometimes resulted in a `Column cannot be null` error. (Bug #9681)
- Fixed a memory leak that occurred when selecting from a view that contained a subquery. (Bug #10107)
- Fixed an optimizer bug in computing the union of two ranges for the `OR` operator. (Bug #9348)
- Fixed a segmentation fault in `mysqlcheck` that occurred when the last table checked in `--auto-repair` mode returned an error (such as the table being a `MERGE` table). (Bug #9492)
- `SET @var= CAST(NULL AS [INTEGER|CHAR])` now sets the result type of the variable to `INTEGER`/`CHAR`. (Bug #6598)
- Incorrect results were returned for queries of the form `SELECT ... LEFT JOIN ... WHERE EXISTS (`*`subquery`*`)`, where the subquery selected rows based on an `IS NULL` condition. (Bug #9516)
- Executing `LOCK TABLES` and then calling a stored procedure caused an error and resulting in the server thinking that no stored procedures exist. (Bug #9566)
- Selecting from a view containing a subquery caused the server to hang. (Bug #8490)
- Within a stored procedure, attempting to execute a multiple-table `UPDATE` failed with a `Table '`*`tbl_name`*`' was locked with a READ lock and can't be updated` error. (Bug #9486)
- Starting `mysqld` with the `--skip-innodb` and `--default-storage-engine=innodb` (or `--default-table-type=innodb` caused a server crash. (Bug #9815)
- Queries containing `CURRENT_USER()` incorrectly were registered in the query cache. (Bug #9796)
- Setting the `storage_engine` system variable to `MEMORY` succeeded, but retrieving the variable resulted in a value of `HEAP` (the old name for the `MEMORY` storage engine) rather than `MEMORY`. (Bug #10039)
- `mysqlshow` displayed an incorrect row count for tables. (Bug #9391)
- The server died with signal 11 if a non-existent location was specified for the location of the binary log. Now the server exits after printing an appropriate error messsage. (Bug #9542)
- Fixed a problem in the client/server protocol where the server closed the connection before sending the final error message. The problem could show up as a `Lost connection to MySQL server during query` when attempting to connect to access a non-existent database. (Bug #6387, Bug #9455)
- Fixed a `readline`-related crash in `mysql` when the user pressed Control-R. (Bug #9568)
- For stored functions that should return a `YEAR` value, corrected a failure of the value to be in `YEAR` format. (Bug #8861)
- Fixed a server crash resulting from invocation of a stored function that returned a value having an `ENUM` or `SET` data type. (Bug #9775)

- Fixed a server crash resulting from invocation of a stored function that returned a value having a `BLOB` data type. (Bug #9102)
- Fixed a server crash resulting from invocation of a stored function that returned a value having a `BIT` data type. (Bug #7648)
- `TIMEDIFF()` with a negative time first argument and postive time second argument produced incorrect results. (Bug #8068)
- Fixed a problem with `OPTIMIZE TABLE` for `InnoDB` tables being written twice to the binary log. (Bug #9149)
- `InnoDB`: Prevent `ALTER TABLE` from changing the storage engine if there are foreign key constraints on the table. (Bug #5574, Bug #5670)
- `InnoDB`: Fixed a bug where next-key locking doesn't allow the insert which does not produce a phantom. (Bug #9354) If the range is of type `'a' <= uniquecolumn`, `InnoDB` lock only the RECORD, if the record with the column value `'a'` exists in a CLUSTERED index. This allows inserts before a range.
- `InnoDB`: When `FOREIGN_KEY_CHECKS=0`, `ALTER TABLE` and `RENAME TABLE` will ignore any type incompatibilities between referencing and referenced columns. Thus, it will be possible to convert the character sets of columns that participate in a foreign key. Be sure to convert all tables before modifying any data! (Bug #9802)
- Provide more informative error messages in clustered setting when a query is issued against a table that has been modified by another `mysqld` server. (Bug #6762)

C.1.8. Cambios en la entrega 5.0.4 (16 Apr 2005)

Functionality added or changed:

- Added `ENGINE=MyISAM` table option when creating `mysql.proc` table in `mysql_create_system_tables` script to make sure the table is created as a `MyISAM` table even if the default storage engine has been changed. (Bug #9496)
- `SHOW CREATE TABLE` for an `INFORMATION_SCHEMA` table no longer prints a `MAX_ROWS` value because the value has no meaning. (Bug #8941)
- Invalid `DEFAULT` values for `CREATE TABLE` now generate errors. (Bug #5902)
- Added `--show-table-type` option to `mysqlshow`, to display a column indicating the table type, as in `SHOW FULL TABLES`. (Bug #5036)
- The way the time zone information is stored into the binary log was changed, so that it's now possible to have a replication master and slave running with different global time zones. A drawback is that replication from 5.0.4 masters to pre-5.0.4 slaves is impossible.
- Added `--with-big-tables` compilation option to `configure`. (Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable large table support.) See Sección 2.8.2, “Opciones típicas de `configure`” for details.
- New configuration directives `!include` and `!includedir` implemented for including option files and searching directories for option files. See Sección 4.3.2, “Usar ficheros de opciones” for usage.

Bugs fixed:

- The use of `XOR` together with `NOT ISNULL()` erroneously resulted in some outer joins being converted to inner joins by the optimizer. (Bug #9017)

- Fixed an optimizer problem where extraneous comparisons between `NULL` values in indexed columns were being done for operators such as `=` that are never true for `NULL`. (Bug #8877)
- Fixed the client/server protocol for prepared statements so that reconnection works properly when the connection is killed while reconnect is enabled. (Bug #8866)
- A server installed as a Windows service and started with `--shared-memory` could not be stopped. (Bug #9665)
- Fixed a server crash resulting from multiple executions of a prepared statement involving a join of an `INFORMATION_SCHEMA` table with another table. (Bug #9383)
- Fixed `utf8_spanish2_ci` and `ucs2_spanish2_ci` collations to not consider '`r`' equal to '`rr`'. If you upgrade to this version from an earlier version, you should rebuild the indexes of affected tables. (Bug #9269)
- `mysqldump` dumped core when invoked with `--tmp` and `--single-transaction` options and a non-existent table name. (Bug #9175)
- Allow extra HKSCS and cp950 characters (`big5` extension characters) to be accepted in `big5` columns. (Bug #9357)
- `mysql.server` no longer uses non-portable `alias` command or LSB functions. (Bug #9852)
- Fixed a server crash resulting from `GROUP BY` on a decimal expression. (Bug #9210)
- In prepared statements, subqueries containing parameters were erroneously treated as `const` tables during preparation, resulting in a server crash. (Bug #8807)
- InnoDB: `ENUM` and `SET` columns were treated incorrectly as character strings. This bug did not manifest itself with `latin1` collations if there were less than about 100 elements in an `ENUM`, but it caused malfunction with `UTF-8`. Old tables will continue to work. In new tables, `ENUM` and `SET` will be internally stored as unsigned integers. (Bug #9526)
- InnoDB: Avoid test suite failures caused by a locking conflict between two server instances at server shutdown/startup. This conflict on advisory locks appears to be the result of a bug in the operating system; these locks should be released when the files are closed, but somehow that does not always happen immediately in Linux. (Bug #9381)
- InnoDB: True `VARCHAR`: InnoDB stored the 'position' of a row wrong in a column prefix primary key index; this could cause MySQL to complain `ERROR 1032: Can't find record …` in an update of the primary key, and also some `ORDER BY` or `DISTINCT` queries. (Bug #9314)
- InnoDB: Fix bug in MySQL/InnoDB 5.0.3: SQL statements were not rolled back on error. (Bug #8650)
- Fixed a `Commands out of sync` error when two prepared statements for single-row result sets were open simultaneously. (Bug #8880)
- Fixed a server crash after a call to `mysql_stmt_close()` for single-row result set. (Bug #9159)
- Fixed server crashes for `CREATE TABLE ... SELECT` or `INSERT INTO ... SELECT` when selecting from multiple-table view. (Bug #8703, Bug #9398)
- `TRADITIONAL` SQL mode should prevent inserts where a column with no default value is omitted or set to a value of `DEFAULT`. Fixed cases where this restriction was not enforced. (Bug #5986)
- Fixed a server crash when creating a `PRIMARY KEY` for a table, if the table contained a `BIT` column. (Bug #9571)

- Warning message from `GROUP_CONCAT()` did not always indicate correct number of lines. (Bug #8681)
- The commit count cache for `NDB` was not properly invalidated when deleting a record using a cursor. (Bug #8585)
- Fixed option-parsing code for the embedded server to understand `K`, `M`, and `G` suffixes for the `net_buffer_length` and `max_allowed_packet` options. (Bug #9472)
- Selecting a `BIT` column failed if the binary client/server protocol was used. (Bug #9608)
- Fixed a permissions problem whereby information in `INFORMATION_SCHEMA` could be exposed to a user with insufficient privileges. (Bug #7214)
- An error now occurs if you try to insert an invalid value via a stored procedure in `STRICT` mode. (Bug #5907)
- Link with `libsupc++` on Fedora Core 3 to get language support functions. (Bug #6554)
- The value of the `CHARACTER_MAXIMUM_LENGTH` and `CHARACTER_OCTET_LENGTH` columns of the `INFORMATION_SCHEMA.COLUMNS` table must be `NULL` for numeric columns, but were not. (Bug #9344)
- `DROP TABLE` did not drop triggers that were defined for the table. `DROP DATABASE` did not drop triggers in the database. (Bug #5859, Bug #6559)
- `CREATE OR REPLACE VIEW` and `ALTER VIEW` now require the `CREATE VIEW` and `DROP` privileges, not `CREATE VIEW` and `DELETE`. (`DELETE` is a row-level privilege, not a table-level privilege.) (Bug #9260)
- Some user variables were not being handled with “implicit” coercibility. (Bug #9425)
- Setting the `max_error_count` system variable to 0 resulted in a setting of 1. (Bug #9072)
- Fixed a collation coercibility problem that caused a union between binary and non-binary columns to fail. (Bug #6519)
- Fixed a bug in division of floating point numbers. It could cause nine zeroes (`000000000`) to be inserted in the middle of the quotient. (Bug #9501)
- `INFORMATION_SCHEMA` tables had an implicit upper limit for the number of rows. As a result, not all data could be returned for some queries. (Bug #9317)
- Fixed a problem with the `tee` command in `mysql` that resulted in `mysql` crashing. (Bug #8499)
- `CAST()` now produces warnings when casting incorrect `INTEGER` and `CHAR` values. This also applies to implicit `string` to `number` casts. (Bug #5912)
- `ALTER TABLE` now fails in `STRICT` mode if the alteration generates warnings.
- Using `CONVERT('0000-00-00',date)` or `CAST('0000-00-00' as date)` in `TRADITIONAL` SQL mode now produces a warning. (Bug #6145)
- Inserting a zero date in a `DATE`, `DATETIME` or `TIMESTAMP` column during `TRADITIONAL` mode now produces an error. (Bug #5933)
- Inserting a zero date into a `DATETIME` column in `TRADITIONAL` mode now produces an error.
- `STR_TO_DATE()` now produces errors in strict mode (and warnings otherwise) when given an illegal argument. (Bug #5902)
- Fixed a problem with `ORDER BY` that sometimes caused incorrect sorting of `utf8` data. (Bug #9309)

- Fixed server crash resulting from queries that combined `SELECT DISTINCT`, `SUM()`, and `ROLLUP`. (Bug #8615)
- Incorrect results were returned from queries that combined `SELECT DISTINCT`, `GROUP BY` , and `ROLLUP`. (Bug #8616)
- Too many rows were returned from queries that combined `ROLLUP` and `LIMIT` if `SQL_CALC_FOUND_ROWS` was given. (Bug #8617)
- If on replication master a `LOAD DATA INFILE` is interrupted in the middle (integrity constraint violation, killed connection...), the slave used to skip this `LOAD DATA INFILE` entirely, thus missing some changes if this command permanently inserted/updated some table records before being interrupted. This is now fixed. (Bug #3247)

C.1.9. Cambios en la entrega 5.0.3 (23 Mar 2005: Beta)

Note: This Beta release, as any other pre-production release, should not be installed on “production” level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has done its best to ensure a high level of quality, protect your data by making a backup as you would for any software beta release.

Functionality added or changed:

- New privilege `CREATE USER` was added.
- Security improvement: The server creates `.frm`, `.MYD`, `.MYI`, `.MRG`, `.ISD`, and `.ISM` table files only if a file with the same name does not already exist. Thanks to Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and informing us about this issue. (CVE-2005-0711)
- Security improvement: User-defined functions 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. `mysqld` by default no longer loads UDFs unless they have at least one auxiliary symbol defined in addition to the main symbol. The `--allow-suspicious-udfs` option controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off. `mysqld` also checks UDF filenames when it reads them from the `mysql.func` table and rejects those that contain directory pathname separator characters. (It already checked names as given in `CREATE FUNCTION` statements.) See Sección 27.2.3.1, “Secuencias de llamada UDF para funciones simples”, Sección 27.2.3.2, “Secuencias de llamada UDF para funciones agregadas”, and Sección 27.2.3.6, “Precauciones de seguridad en funciones definidas por usuarios”. Thanks to Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and informing us about this issue. (CVE-2005-0709, CVE-2005-0710)
- `my.cnf` in the compile-time datadir (usually `/usr/local/mysql/data/` in the binary tarball distributions) is not being read anymore. The value of the environment variable `MYSQL_HOME` is used instead of the hard-coded path.
- Support for the `ISAM` storage engine has been removed. If you have `ISAM` tables, you should convert them before upgrading. See Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0”.
- Support for `RAID` options in `MyISAM` tables has been removed. If you have tables that use these options, you should convert them before upgrading. See Sección 2.10.1, “Aumentar la versión de 4.1 a 5.0”.
- Added support for `AVG(DISTINCT)`.
- `ONLY_FULL_GROUP_BY` no longer is included in the `ANSI` composite SQL mode. (Bug #8510)
- `mysqld_safe` will create the directory where the UNIX socket file is to be located if the directory does not exist. This applies only to the last component of the directory pathname. (Bug #8513)

- The coercibility for the return value of functions such as `USER()` or `VERSION()` now is “system constant” rather than “implicit.” This makes these functions more coercible than column values so that comparisons of the two do not result in `Illegal mix of collations` errors. `COERCIBILITY()` was modified to accommodate this new coercibility value. See Sección 12.9.3, “Funciones de información”.
- User variable coercibility has been changed from “coercible” to “implicit.” That is, user variables have the same coercibility as column values.
- Boolean full-text phrase searching now requires only that matches contain exactly the same words as the phrase and in the same order. Non-word characters no longer need match exactly.
- `CHECKSUM TABLE` returns a warning for non-existing tables. The checksum value remains `NULL` as before. (Bug #8256)
- The server now includes a timestamp in the `Ready for connections` message that is written to the error log at startup. (Bug #8444)
- Added `SQL_NOTES` session variable to cause `Note`-level warnings not to be recorded. (Bug #6662)
- Allowed the service-installation command for Windows servers to specify a single option other than `--defaults-file` following the service name. This is for compatibility with MySQL 4.1. (Bug #7856)
- InnoDB: **Upgrading from 4.1:** The sorting order for end-space in `TEXT` columns for InnoDB tables has changed. Starting from 5.0.3, InnoDB compares `TEXT` columns as space-padded at the end. If you have a non-unique index on a `TEXT` column, you should run `CHECK TABLE` on it, and run `OPTIMIZE TABLE` if the check reports errors. If you have a `UNIQUE INDEX` on a `TEXT` column, you should rebuild the table with `OPTIMIZE TABLE`.
- `InnoDB`: Commit after every 10,000 copied rows when executing `ALTER TABLE`, `CREATE INDEX`, `DROP INDEX` or `OPTIMIZE TABLE`. This makes it much faster to recover from an aborted operation.
- Added `VAR_POP()` and `STDDEV_POP()` as standard SQL aliases for the `VARIANCE()` and `STDDEV()` functions that compute population variance and standard deviation. Added new `VAR_SAMP()` and `STDDEV_SAMP()` functions to compute sample variance and standard deviation. (Bug #3190)
- Fixed a problem with out-of-order packets being sent (`ERROR` after `OK` or `EOF`) following a `KILL QUERY` statement. (Bug #6804)
- Retrieving from a view defined as a `SELECT` that mixed `UNION ALL` and `UNION DISTINCT` resulted in a different result than retrieving from the original `SELECT`. (Bug #6565)
- Fixed a problem with non-optimal `index_merge` query execution plans being chosen on IRIX. (Bug #8578)
- `BIT` in column definitions now is a distinct data type; it no longer is treated as a synonym for `TINYINT(1)`.
- Bit-field values can be written using `b'value'` notation. *`value`* is a binary value written using 0s and 1s.
- From the Windows distribution, predefined accounts without passwords for remote users ("root@%", "@%") were removed (other distributions never had them).
- Added `mysql_library_init()` and `mysql_library_end()` as synonyms for the `mysql_server_init()` and `mysql_server_end()` C API functions. `mysql_library_init()` and `mysql_library_end()` are `#define` symbols, 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`. (Bug #6149)

- `SHOW COLUMNS` now displays `NO` rather than blank in the `Null` output column if the corresponding table column cannot be `NULL`.
- Changed XML format for `mysql` from `<col_name>col_value</col_name>` to `<field name="col_name">col_value</field>` to allow for proper encoding of column names that are not legal as element names. (Bug #7811)
- Added `--innodb-checksums` and `--innodb-doublewrite` options for `mysqld`.
- Added `--large-pages` option for `mysqld`.
- Added `multi_read_range` system variable.
- `SHOW DATABASES`, `SHOW TABLES`, `SHOW COLUMNS`, and so forth display information about the `INFORMATION_SCHEMA` database. Also, several `SHOW` statements now accept a `WHERE` clause specifying which output rows to display. See Capítulo 22, *La base de datos de información `INFORMATION_SCHEMA`*.
- Added the `CREATE ROUTINE` and `ALTER ROUTINE` privileges, and made the `EXECUTE` privilege operational.
- InnoDB: Corrected a bug in the crash recovery of `ROW_FORMAT=COMPACT` tables that caused corruption. (Bug #7973) There may still be bugs in the crash recovery, especially in `COMPACT` tables.
- When the `MyISAM` storage engine detects corruption of a `MyISAM` table, a message describing the problem now is written to the error log.
- InnoDB: When MySQL/InnoDB is compiled on Mac OS X 10.2 or earlier, detect the operating system version at run time and use the `fcntl()` file flush method on Mac OS X versions 10.3 and later. In Mac OS X, `fsync()` does not flush the write cache in the disk drive, but the special `fcntl()` does; however, the flush request is ignored by some external devices. Failure to flush the buffers may cause severe database corruption at power outages.
- InnoDB: Implemented fast `TRUNCATE TABLE`. The old approach (deleting rows one by one) may be used if the table is being referenced by foreign keys. (Bug #7150)
- Added `cp932` (SJIS for Windows Japanese) and `eucjpms` (UJIS for Windows Japanese) character sets.
- Added several `InnoDB` status variables. See Sección 5.3.4, “Variables de estado del servidor”.
- Added the `FEDERATED` storage engine. See Sección 14.6, “El motor de almacenamiento `FEDERATED`”.
- `SHOW CREATE TABLE` now uses `USING index_type` rather than `TYPE index_type` to specify an index type. (Bug #7233)
- InnoDB now supports a fast `TRUNCATE TABLE`. One visible change from this is that auto-increment values for this table are reset on `TRUNCATE`.
- Added an `error` member to the `MYSQL_BIND` data structure that is used in the C API for prepared statements. This member is used for reporting data truncation errors. Truncation reporting is enabled via the new `MYSQL_REPORT_DATA_TRUNCATION` option for the `mysql_options()` C API function.
- API change: the `reconnect` flag in the `MYSQL` structure is now set to 0 by `mysql_real_connect()`. Only those client programs which didn't 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 (after reconnection, table locks, temporary tables, user and session variables are lost).
- `FLUSH TABLES WITH READ LOCK` is now killable while it's waiting for running `COMMIT` statements to finish.

- `MEMORY` (`HEAP`) can have `VARCHAR()` fields.
- `VARCHAR` columns now remember end space. A `VARCHAR()` column can now contain up to 65535 bytes. For more details, see Sección C.1, “Cambios en la entrega 5.0.x (Desarrollo)”. If the table handler doesn't support the new `VARCHAR` type, then it's converted to a `CHAR` column. Currently this happens for `NDB` tables.
- InnoDB: Introduced a compact record format that does not store the number of columns or the lengths of fixed-size columns. The old format can be requested by specifying `ROW_FORMAT=REDUNDANT`. The new format (`ROW_FORMAT=COMPACT`) is the default. The new format typically saves 20 % of disk space and memory.
- InnoDB: Setting the initial `AUTO_INCREMENT` value for an `InnoDB` table using `CREATE TABLE ... AUTO_INCREMENT = n` now works, and `ALTER TABLE ... AUTO_INCREMENT = n` resets the current value.
- `Seconds_Behind_Master` is `NULL` (which means “unknown”) if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. It is zero if the SQL thread has caught up to the I/O thread. It no longer grows indefinitely if the master is idle.
- The MySQL server aborts immediately instead of simply issuing a warning if it is started with the `--log-bin` option but cannot initialize the binary log at startup (that is, an error occurs when writing to the binary log file or binary log index file).
- The binary log file and binary log index file now are handled the same way as `MyISAM` tables when there is a “disk full” or “quota exceeded” error. See Sección A.4.3, “Cómo se comporta MySQL ante un disco lleno”.
- The MySQL server now aborts when started with option `--log-bin-index` and without `--log-bin`, and when started with `--log-slave-updates` and without `--log-bin`.
- If the MySQL server is started without an argument to `--log-bin` and without `--log-bin-index`, thus not providing a name for the binary log index file, a warning is issued because MySQL falls back to using the hostname for that name, and this is prone to replication issues if the server's hostname's gets changed later. See Sección A.8.4, “Cuestiones abiertas en MySQL”.
- Added account-specific `MAX_USER_CONNECTIONS` limit, which allows you to specify the maximum number of concurrent connections for the account. Also, all limited resources now are counted per account (instead of being counted per user + host pair as it was before). Use the `--old-style-user-limits` option to get the old behavior.
- InnoDB: A shared record lock (`LOCK_REC_NOT_GAP`) is now taken for a matching record in the foreign key check because inserts can be allowed into gaps.
- InnoDB: Relaxed locking in `INSERT…SELECT`, single table `UPDATE…SELECT` and single table `DELETE…SELECT` clauses when `innodb_locks_unsafe_for_binlog` is used and isolation level of the transaction is not serializable. `InnoDB` uses consistent read in these cases for a selected table.
- Added a new global system variable `slave_transaction_retries`: if the replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or 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. The default is 10. (Bug #8325)
- When a client releases a user-level lock, `DO RELEASE_LOCK()` will not be written to the binary log anymore (this makes the binary log smaller); as a counterpart, the slave does not actually take the lock when it executes `GET_LOCK()`. This is mainly an optimization and should not affect existing setups. (Bug #7998)

- The way the character set information is stored into the binary log was changed, so that it's now possible to have a replication master and slave running with different global character sets. A drawback is that replication from 5.0.3 masters to pre-5.0.3 slaves is impossible.
- The `LOAD DATA` statement was extended to support user variables in the target column list, and an optional `SET` clause. Now one can perform some transformations on data after they have been read and before they are inserted into the table. For example:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

  Also, replication of `LOAD DATA` was changed, so you can't replicate such statements from a 5.0.3 master to pre-5.0.3 slaves.

Bugs fixed:

- If a `MyISAM` table on Windows had `INDEX DIRECTORY` or `DATA DIRECTORY` table options, `mysqldump` dumped the directory pathnames with single-backslash pathname separators. This would cause syntax errors when importing the dump file. `mysqldump` now changes '`\`' to '`/`' in the pathnames on Windows. (Bug #6660)
- `mysql_fix_privilege_tables` now fixes that the `mysql` privilege tables can be used in MySQL 4.1. This allows one to easily downgrade to 4.1 or run MySQL 5.0 and 4.1 with the same privilege files for testing purposes.
- Fixed bug creating user with GRANT fails with password but works without, (Bug #7905)
- `mysqldump` misinterpreted '`_`' and '`%`' characters in the names of tables to be dumped as wildcard characters. (Bug #9123)
- The definition of the enumeration-valued `sql_mode` column of the `mysql.proc` table was missing some of the current allowable SQL modes, so stored routines would not necessarily execute with the SQL mode in effect at the time of routine definition. (Bug #8902)
- `REPAIR TABLE` did not invalidate query results in the query cache that were generated from the table. (Bug #8480)
- In strict or traditional SQL mode, too-long string values assigned to string columns (`CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TEXT`, or `BLOB`) were correctly truncated, but the server returned an SQLSTATE value of `01000` (should be `22001`). (Bug #6999, Bug #9029)
- Stored functions that used cursors could return incorrect results. (Bug #8386)
- `AES_DECRYPT(`*`col_name`*`,`*`key`*`)` could fail to return `NULL` for invalid values in *`col_name`*, if *`col_name`* was declared as `NOT NULL`. (Bug #8669)
- Ordering by unsigned expression (more complex than a column reference) was treating the value as signed, producing incorrectly sorted results. (Bug #7425)
- `HAVING` was treating unsigned columns as signed. (Bug #7425)
- Fixed a problem with boolean full-text searches on `utf8` columns where a double quote in the search string caused a server crash. (Bug #8351)
- For a query with both `GROUP BY` and `COUNT(DISTINCT)` clauses and a `FROM` clause with a subquery, `NULL` was returned for any `VARCHAR` column selected by the subquery. (Bug #8218)

- Fixed a bug in `TRUNCATE`, which did not work within stored procedures. A workaround has been made so that within stored procedures, `TRUNCATE` is executed like `DELETE`. This was necessary because `TRUNCATE` is implicitly locking tables. (Bug #8850)
- Fixed an optimizer bug that caused incorrectly ordered result from a query that used a `FULLTEXT` index to retrieve rows and there was another index that was usable for `ORDER BY`. For such a query, `EXPLAIN` showed `fulltext` join type, but regular (not `FULLTEXT`) index in the `Key` column. (Bug #6635)
- If `SELECT DISTINCT` named an index column multiple times in the select list, the server tried to access different key fields for each instance of the column, which could result in a crash. (Bug #8532)
- For a stored function that refers to a given table, invoking the function while selecting from the same table resulted in a server crash. (Bug #8405)
- Comparison of a `DECIMAL` column containing `NULL` to a subquery that produced `DECIMAL` values resulted in a server crash. (Bug #8397)
- The `--set-character-set` option for `myisamchk` was changed to `--set-collation`. The value needed for specifying how to sort indexes is a collation name, not a character set name. (Bug #8349)
- Hostname matching didn't work if a netmask was specified for table-specific privileges. (Bug #3309)
- Corruption of `MyISAM` table indexes could occur with `TRUNCATE TABLE` if the table had already been opened. For example, this was possible if the table had been opened implicitly by selecting from a `MERGE` table that mapped to the `MyISAM` table. The server now issues an error message for `TRUNCATE TABLE` under these conditions. (Bug #8306)
- Setting the connection collation to a value different from the server collation followed by a `CREATE TABLE` statement that included a quoted default value resulted in a server crash. (Bug #8235)
- Fixed handling of table-name matching in `mysqlhotcopy` to accommodate `DBD::mysql` 2.9003 and up (which implement identifier quoting). (Bug #8136)
- Selecting from a view defined as a join caused a server crash if the query cache was enabled. (Bug #8054)
- Results in the query cache generated from a view were not properly invalidated after `ALTER VIEW` or `DROP VIEW` on that view. (Bug #8050)
- `FOUND_ROWS()` returned an incorrect value after a `SELECT SQL_CALC_FOUND_ROWS DISTINCT` statement that selected constants and included `GROUP BY` and `LIMIT` clauses. (Bug #7945)
- Selecting from an `INFORMATION_SCHEMA` table combined with a subselect on an `INFORMATION_SCHEMA` table caused an error with the message `Table tbl_name is corrupted`. (Bug #8164)
- Fixed a problem with equality propagation optimization for prepared statements and stored procedures that caused a server crash upon re-execution of the prepared statement or stored procedure. (Bug #8115, Bug #8849)
- `LEFT OUTER JOIN` between an empty base table and a view on an empty base table caused a server crash. (Bug #7433)
- Use of `GROUP_CONCAT()` in the select list when selecting from a view caused a server crash. (Bug #7116)
- Use of a view in a correlated subquery that contains `HAVING` but no `GROUP BY` caused a server crash. (Bug #6894)

- Handling by `mysql_list_fields()` of references to stored functions within views was incorrect and could result in a server crash. (Bug #6814)
- `mysqldump` now avoids writing `SET NAMES` to the dump output if the server is older than version 4.1 and would not understand that statement. (Bug #7997)
- Fixed problems when selecting from a view that had an `EXISTS` or `NOT EXISTS` subquery. Selecting columns by name caused a server crash. With `SELECT *`, a crash did not occur, but columns in outer query were not resolved properly. (Bug #6394)
- DDL statements for views were not being written to the binary log (and thus not subject to replication). (Bug #4838)
- The `CHAR()` function was not ignoring `NULL` arguments, contrary to the documentation. (Bug #6317)
- Creating a table using a name containing a character that is illegal in `character_set_client` resulted in the character being stripped from the name and no error. The character now is considered an error. (Bug #8041)
- Fixed a problem with the Cyrillic letters I and SHORT I being treated the same by the `utf8_general_ci` collation. (Bug #8385)
- Some `INFORMATION_SCHEMA` columns that contained catalog identifiers were of type `LONGTEXT`. These were changed to `VARCHAR(N)`, where *`N`* is the appropriate maximum identifier length. (Bug #7215)
- Some `INFORMATION_SCHEMA` columns that contained timestamp values were of type `VARBINARY`. These were changed to `TIMESTAMP`. (Bug #7217)
- An expression that tested a case-insensitive character column against string constants that differed in lettercase could fail because the constants were treated as having a binary collation. (For example, `WHERE city='London' AND city='london'` could fail.) (Bug #7098, Bug #8690)
- The output of the `STATUS` (`\s`) command in `mysql` had the values for the server and client character sets reversed. (Bug #7571)
- If the slave was running with `--replicate-*-table` options which excluded one temporary table and included another, and the two tables were used in a single `DROP TEMPORARY TABLE IF EXISTS` statement, as the ones the master automatically writes to its binary log upon client's disconnection when client has not explicitly dropped these, the slave could forget to delete the included replicated temporary table. Only the slave needs to be upgraded. (Bug #8055)
- When setting integer system variables to a negative value with `SET VARIABLES`, the value was treated as a positive value modulo 2^32. (Bug #6958)
- Corrected a problem with references to `DUAL` where statements such as `SELECT 1 AS a FROM DUAL` would succeed but statements such as `SELECT 1 AS a FROM DUAL LIMIT 1` would fail. (Bug #8023)
- Fixed a server crash caused by `DELETE FROM` *`tbl_name`* `... WHERE ... ORDER BY` *`tbl_name.col_name`* when the `ORDER BY` column was qualified with the table name. (Bug #8392)
- Fixed a bug in `MATCH ... AGAINST` in natural language mode that could cause a server crash if the `FULLTEXT` index was not used in a join (`EXPLAIN` did not show `fulltext` join mode) and the search query matched no rows in the table (Bug #8522).
- `InnoDB`: Honor the `--tmpdir` startup option when creating temporary files. Previously, `InnoDB` temporary files were always created in the temporary directory of the operating system. On Netware, `InnoDB` will continue to ignore `--tmpdir`. (Bug #5822)

- Platform and architecture information in version information produced for `--version` option on Windows was always `Win95/Win98 (i32)`. More accurately determine platform as `Win32` or `Win64` for 32-bit or 64-bit Windows, and architecture as `ia32` for x86, `ia64` for Itanium, and `axp` for Alpha. (Bug #4445)
- If multiple semicolon-separated statements were received in a single packet, they were written to the binary log as a single event rather than as separate per-statement events. For a server serving as a replication master, this caused replication to fail when the event was sent to slave servers. (Bug #8436)
- Fixed `LOAD INDEX` statement to actually load index in memory. (Bug #8452)
- Fixed a failure of multiple-table updates to replicate properly on slave servers when `--replicate-*-table` options had been specified. (Bug #7011)
- Fixed failure of `CREATE TABLE ... LIKE` Windows when the source or destination table was located in a symlinked database directory. (Bug #6607)
- With `lower_case_table_names` set to 1, `mysqldump` on Windows could write the same table name in different lettercase for different SQL statements. Fixed so that consistent lettercase is used. (Bug #5185)
- `mysqld_safe` now understands the `--help` option. Previously, it ignored the option and attempted to start the server anyway. (Bug #7931)
- Fixed problem in `NO_BACKSLASH_ESCAPES` SQL mode for strings that contained both the string quoting character and backslash. (Bug #6368)
- Fixed some portability issues with overflow in floating point values.
- Prepared statements now gives warnings on prepare.
- Fixed bug in prepared statements with `SUM(DISTINCT...)`.
- Fixed bug in prepared statements with `OUTER JOIN`.
- Fixed a bug in `CONV()` function returning unsigned `BIGINT` number (third argument is positive, and return value does not fit in 32 bits). (Bug #7751)
- Fixed a failure of the `IN()` operator to return correct result if all values in the list were constants and some of them were using substring functions, for example, `LEFT()`, `RIGHT()`, or `MID()`. (Bug #7716)
- Fixed a crash in `CONVERT_TZ()` function when its second or third argument was from a `const` table (see Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)”). (Bug #7705)
- Fixed a problem with calculation of number of columns in row comparison against subquery. (Bug #8020)
- Fixed erroneous output resulting from `SELECT DISTINCT` combined with a subquery and `GROUP BY`. (Bug #7946)
- Fixed server crash in comparing a nested row expression (for example `row(1,(2,3))`) with a subquery. (Bug #8022)
- Fixed server crash resulting from certain correlated subqueries with forward references (references to an alias defined later in the outer query). (Bug #8025)
- Fixed server crash resulting from re-execution of prepared statements containing subqueries. (Bug #8125)

- Fixed a bug where `ALTER TABLE` improperly would accept an index on a `TIMESTAMP` column that `CREATE TABLE` would reject. (Bug #7884)
- `SHOW CREATE TABLE` now reports `ENGINE=MEMORY` rather than `ENGINE=HEAP` for a `MEMORY` table (unless the `MYSQL323` SQL mode is enabled). (Bug #6659)
- Fixed a bug where the use of `GROUP_CONCAT()` with `HAVING` caused a server crash. (Bug #7769)
- Fixed a bug where comparing the result of a subquery to a non-existent column caused a server crash on Windows. (Bug #7885)
- Fixed a bug in a combination of `-not` and `trunc*` operators of full-text search. Using more than one truncated negative search term, was causing empty result set.
- InnoDB: Corrected the handling of trailing spaces in the `ucs2` character set. (Bug #7350, Bug #8771)
- InnoDB: Use native `tmpfile()` function on Netware. All InnoDB temporary files are created under `sys:\tmp`. Previously, InnoDB temporary files were never deleted on Netware.
- Fixed a bug in `max_heap_table_size` handling, that resulted in `Table is full` error when the table was still smaller than the limit. (Bug #7791).
- Fixed a symlink vulnerability in the `mysqlaccess` script. Reported by Javier Fernandez-Sanguino Pena and Debian Security Audit Team. (CVE-2005-0004)
- Fixed a bug that caused server crash if some error occured during filling of temporary table created for derived table or view handling. (Bug #7413)
- Fixed a bug which caused server crash if query containing `CONVERT_TZ()` function with constant arguments was prepared. (Bug #6849)
- Prevent adding `CREATE TABLE .. SELECT` query to the binary log when the insertion of new records partially failed. (Bug #6682)
- Fixed a bug which caused a crash when only the slave I/O thread was stopped and started. (Bug #6148)
- Giving `mysqld` a `SIGHUP` caused it to crash.
- Changed semantics of `CREATE/ALTER/DROP DATABASE` statements so that replication of `CREATE DATABASE` is possible when using `--binlog-do-db` and `--binlog-ignore-db`. (Bug #6391)
- A sequence of `BEGIN` (or `SET AUTOCOMMIT=0`), `FLUSH TABLES WITH READ LOCK`, transactional update, `COMMIT`, `FLUSH TABLES WITH READ LOCK` could hang the connection forever and possibly the MySQL server itself. This happened for example when running the `innobackup` script several times. (Bug #6732)
- `mysqlbinlog` did not print `SET PSEUDO_THREAD_ID` statements in front of `LOAD DATA INFILE` statements inserting into temporary tables, thus causing potential problems when rolling forward these statements after restoring a backup. (Bug #6671)
- InnoDB: Fixed a bug no error message for ALTER with InnoDB and AUTO_INCREMENT (Bug #7061). `InnoDB` now supports `ALTER TABLE...AUTO_INCREMENT = x` query to set auto increment value for a table.
- Made the MySQL server accept executing `SHOW CREATE DATABASE` even if the connection has an open transaction or locked tables; refusing it made `mysqldump --single-transaction` sometimes fail to print a complete `CREATE DATABASE` statement for some dumped databases. (Bug #7358)

- Fixed that, when encountering a “disk full” or “quota exceeded” write error, `MyISAM` sometimes didn't sleep and retry the write, thus resulting in a corrupted table. (Bug #7714)
- Fixed that `--expire-log-days` was not honored if using only transactions. (Bug #7236)
- Fixed that a slave could crash after replicating many `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `REPAIR TABLE` statements from the master. (Bug #6461, Bug #7658)
- `mysqlbinlog` forgot to add backquotes around the collation of user variables (causing later parsing problems as `BINARY` is a reserved word). (Bug #7793)
- Ensured that `mysqldump --single-transaction` sets its transaction isolation level to `REPEATABLE READ` before proceeding (otherwise if the MySQL server was configured to run with a default isolation level lower than `REPEATABLE READ` it could give an inconsistent dump). (Bug #7850)
- Fixed that when using the `RPAD()` function (or any function adding spaces to the right) in a query that had to be resolved by using a temporary table, all resulting strings had rightmost spaces removed (i.e. `RPAD()` did not work) (Bug #4048)
- Fixed that a 5.0.3 slave can connect to a master < 3.23.50 without hanging (the reason for the hang is a bug in these quite old masters -- `SELECT @@unknown_var` hangs them -- which was fixed in MySQL 3.23.50). (Bug #7965)
- InnoDB: Fixed a deadlock without any locking, simple select and update (Bug #7975). `InnoDB` now takes an exclusive lock when `INSERT ON DUPLICATE KEY UPDATE` is checking duplicate keys.
- Fixed a bug where MySQL was allowing concurrent updates (inserts, deletes) to a table if binary logging is enabled. Changed to ensure that all updates are executed in a serialized fashion, because they are executed serialized when binlog is replayed. (Bug #7879)
- Fixed a rare race condition which could lead to `FLUSH TABLES WITH READ LOCK` hanging. (Bug #8682)
- Fixed a bug that caused the slave to stop on statements that produced an error on the master. (Bug #8412)

C.1.10. Cambios en la entrega 5.0.2 (01 Dec 2004)

Functionality added or changed:

- **Warning: Incompatible change!** The precedence of `NOT` operator has changed so that expressions such as `NOT a BETWEEN b AND c` are parsed correctly as `NOT (a BETWEEN b AND c)` rather than as `(NOT a) BETWEEN b AND c`. The pre-5.0 higher-precedence behavior can be obtained by enabling the new `HIGH_NOT_PRECEDENCE` SQL mode.
- `SHOW STATUS` now shows the thread specific status variables and `SHOW GLOBAL STATUS` shows the status variables for the whole server.
- Added support for the `INFORMATION_SCHEMA` “information database” that provides database metadata. See Capítulo 22, *La base de datos de información `INFORMATION_SCHEMA`*.
- A `HAVING` clause in a `SELECT` statement now can refer to columns in the `GROUP BY` clause, as required by standard SQL.
- Added the `CREATE USER` and `RENAME USER` statements.
- Modify `DROP USER` so that it drops the account, including all its privileges. Formerly, it removed the account record only for an account that had had all privileges revoked.

- Added `IS [NOT] boolean_value` syntax, where *`boolean_value`* is `TRUE`, `FALSE`, or `UNKNOWN`.
- Added several `InnoDB` status variables. See Sección 5.3.4, “Variables de estado del servidor”.
- Implemented the `WITH CHECK OPTION` clause for `CREATE VIEW`.
- `CHECK TABLE` now works for views.
- The `SCHEMA` and `SCHEMAS` keywords are now accepted as synonyms for `DATABASE` and `DATABASES`.
- Added initial support for rudimentary triggers (the `CREATE TRIGGER` and `DROP TRIGGER` statements).
- Added basic support for read-only server side cursors.
- `mysqldump --single-transaction --master-data` is now able to take an online (non-blocking) dump of InnoDB and report the corresponding binary log coordinates, which makes a backup suitable for point-in-time recovery, roll-forward or replication slave creation. See Sección 8.7, “El programa de copia de seguridad de base de datos `mysqldump`”.
- Added `--start-datetime`, `--stop-datetime`, `--start-position`, `--stop-position` options to `mysqlbinlog` (makes point-in-time recovery easier).
- Made the MySQL server not react to signals `SIGHUP` and `SIGQUIT` on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals (reported as Bug #2030).
- New `--auto-increment-increment` and `--auto-increment-offset` startup options. These allow you to set up a server to generate auto-increment values that don't conflict with another server.
- MySQL now by default checks dates and in strict mode allows only fully correct dates. If you want MySQL to behave as before, you should enable the new `ALLOW_INVALID_DATES` SQL mode.
- Added `STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, and `TRADITIONAL` SQL modes. The `TRADITIONAL` mode is shorthand for all the preceding modes. When using mode `TRADITIONAL`, MySQL generates an error if you try to insert a wrong value in a column. It does not adjust the value to the closest possible legal value.
- MySQL now remembers which columns were declared to have default values. In `STRICT_TRANS_TABLES`/`STRICT_ALL_TABLES` mode, you now get an error if you do an `INSERT` without specifying all columns that don't have a default value. A side effect of this is that when you do `SHOW CREATE` for a new table, you no longer see a `DEFAULT` value for a column for which you didn't specify a default value.
- The compilation flag `DONT_USE_DEFAULT_FIELDS` was removed because you can get the same behavior by setting the `sql_mode` system variable to `STRICT_TRANS_TABLES`.
- Added `NO_AUTO_CREATE_USER` SQL mode to prevent `GRANT` from automatically creating new users if it would otherwise do so, unless a password also is specified.
- We now detect too-large floating point numbers during statement parsing and generate an error messages for them.
- Renamed the `sql_updatable_view_key` system variable to `updatable_views_with_limit`. This variable now can have only two values:
  - `1` or `YES`: Don't issue an error message (warning only) if a VIEW without presence of a key in the underlying table is used in queries with a `LIMIT` clause for updating. (This is the default value.)

  - `0` or `NO`: Prohibit update of a VIEW, which does not contain a key in the underlying table and the query uses a `LIMIT` clause (usually get from GUI tools).

- Reverted output format of `SHOW TABLES` to old pre-5.0.1 format that did not include a table type column. To get the additional column that lists the table type, use `SHOW FULL TABLES` now.
- The `mysql_fix_privilege_tables` script now initializes the global `CREATE VIEW` and `SHOW VIEW` privileges in the `user` table to the value of the `CREATE` privilege in that table.
- If the server finds that the `user` table has not been upgraded to include the view-related privilege columns, it treats each account as having view privileges that are the same as its `CREATE` privilege.
- InnoDB: If you specify the option `innodb_locks_unsafe_for_binlog` in `my.cnf`, InnoDB in an `UPDATE` or a `DELETE` only locks the rows that it updates or deletes. This greatly reduces the probability of deadlocks.
- A connection doing a rollback now displays "Rolling back" in the `State` column of `SHOW PROCESSLIST`.
- `mysqlbinlog` now prints an informative commented line (thread id, timestamp, server id, etc) before each `LOAD DATA INFILE`, like it does for other queries; unless `--short-form` is used.
- Two new server system variables were introduced. `auto_increment_increment` and `auto_increment_offset` can be set locally or globally, and are intended for use in controlling the behaviour of `AUTO_INCREMENT` columns in master-to-master replication. Note that these variables are not intended to take the place of sequences. See Sección 5.3.3, “Variables de sistema del servidor”.

Bugs fixed:

- Fixed that `mysqlbinlog --read-from-remote-server` sometimes couldn't accept two binary logfiles on the command line. (Bug #4507)
- Fixed that `mysqlbinlog --position --read-from-remote-server` had incorrect `# at` lines. (Bug #4506)
- Fixed that `CREATE TABLE ... TYPE=HEAP ... AS SELECT...` caused replication slave to stop. (Bug #4971)
- Fixed that `mysql_options(...,MYSQL_OPT_LOCAL_INFILE,...)` failed to disable `LOAD DATA LOCAL INFILE`. (Bug #5038)
- Fixed that `disable-local-infile` option had no effect if client read it from a configuration file using `mysql_options(...,MYSQL_READ_DEFAULT,...)`. (Bug #5073)
- Fixed that `SET GLOBAL SYNC_BINLOG` did not work on some platforms (Mac OS X). (Bug #5064)
- Fixed that `mysql-test-run` failed on the `rpl_trunc_binlog` test if running test from the installed (the target of 'make install') directory. (Bug #5050)
- Fixed that `mysql-test-run` failed on the `grant_cache` test when run as Unix user 'root'. (Bug #4678)
- Fixed an unlikely deadlock which could happen when using `KILL`. (Bug #4810)
- Fixed a crash when one connection got `KILL`ed while it was doing `START SLAVE`. (Bug #4827)
- Made `FLUSH TABLES WITH READ LOCK` block `COMMIT` if server is running with binary logging; this ensures that the binary log position can be trusted when doing a full backup of tables and the binary log. (Bug #4953)

- Fixed that the counter of an `auto_increment` column was not reset by `TRUNCATE TABLE` is the table was a temporary one. (Bug #5033)
- Fixed slave SQL thread so that the `SET COLLATION_SERVER...` statements it replicates don't advance its position (so that if it gets interrupted before the actual update query, it later redoes the `SET`). (Bug #5705)
- Fixed that if the slave SQL thread found a syntax error in a query (which should be rare, as the master parsed it successfully), it stops. (Bug #5711)
- Fixed that if a write to a MyISAM table fails because of a full disk or an exceeded disk quota, it prints a message to the error log every 10 minutes, and waits until disk becomes free. (Bug #3248)
- Fixed problem introduced in 4.0.21 where a connection starting a transaction, doing updates, then `FLUSH TABLES WITH READ LOCK`, then `COMMIT`, would cause replication slaves to stop (complaining about error 1223). Bug surfaced when using the InnoDB `innobackup` script. (Bug #5949)
- `OPTIMIZE TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE` are now replicated without any error code in the binary log. (Bug #5551)
- If a connection had an open transaction but had done no updates to transactional tables (for example if had just done a `SELECT FOR UPDATE` then executed a non-transactional update, that update automatically committed the transaction (thus releasing InnoDB's row-level locks etc). (Bug #5714)
- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the `BEGIN` and `ROLLBACK` binary log events; that caused superfluous slave stops. (Bug #6522)
- Fixed a bug which prevented `mysqlbinlog` from being able to read from `stdin`, for example, when piping the output from `zcat` to `mysqlbinlog`. (Bug #7853)

C.1.11. Cambios en la entrega 5.0.1 (27 Jul 2004)

Note: This build passes our test suite and fixes a lot of reported bugs found in the previous 5.0.0 release. However, please be aware that this is not a “standard MySQL build” in the sense that there are still some open critical bugs in our bugs database at http://bugs.mysql.com/ that affect this release as well. We are actively fixing these and will make a new release where these are fixed as soon as possible. However, this binary should be a good candidate for testing new MySQL 5.0 features for future products.

Functionality added or changed:

- **Warning: Incompatible change!** C API change: `mysql_shutdown()` now requires a second argument. This is a source-level incompatibility that affects how you compile client programs; it does not affect the ability of compiled clients to communicate with older servers. See Sección 24.2.3.56, “`mysql_shutdown()`”.
- When installing a MySQL server as a Windows service, the installation command can include a `--local-service` option following the service name to cause the server to run using the `LocalService` Windows account that has limited privileges. This is in addition to the `--defaults-file` option that also can be given following the service name.
- Added support for read-only and updatable views based on a single table or other updatable views. View use requires that you upgrade your grant tables to add the view-related privileges. See Sección 2.10.2, “Aumentar la versión de las tablas de privilegios”.
- Implemented a new “greedy search” optimizer that can significantly reduce the time spent on query optimization for some many-table joins. (You are affected if not only some particular `SELECT` is slow, but even using `EXPLAIN` for it takes a noticeable amount of time.) Two new system variables,

`optimizer_search_depth` and `optimizer_prune_level`, can be used to fine-tune optimizer behavior.

- A stored procedure is no longer “global.” That is, it now belongs to a specific database:
  - When a database is dropped, all routines belonging to that database are also dropped.
  - Procedure names may be qualified, for example, `db.p()`
  - When executed from another database, an implicit `USE db_name` is in effect.
  - Explicit `USE db_name` statements no longer are allowed in a stored procedure.

  See Capítulo 19, *Procedimientos almacenados y funciones*.
- Fixed `SHOW TABLES` output field name and values according to standard. Field name changed from `Type` to `table_type`, values are `BASE TABLE`, `VIEW` and `ERROR`. (Bug #4603)
- Added the `sql_updatable_view_key` system variable.
- Added the `--replicate-same-server-id` server option.
- Added `Last_query_cost` status variable that reports optimizer cost for last compiled query.
- Added the `--to-last-log` option to `mysqlbinlog`, for use in conjunction with `--read-from-remote-server`.
- Added the `--innodb-safe-binlog` server option, which adds consistency guarantees between the content of `InnoDB` tables and the binary log. See Sección 5.10.3, “El registro binario (Binary Log)”.
- `OPTIMIZE TABLE` for `InnoDB` tables is now mapped to `ALTER TABLE` instead of `ANALYZE TABLE`. This rebuilds the table, which updates index statistics and frees space in the clustered index.
- `sync_frm` is now a settable global variable (not only a startup option).
- For replication of `MEMORY` (`HEAP`) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a `MEMORY` table is opened for the first time since master's startup. This is for the case where the slave has replicated a non-empty `MEMORY` table, then the master is shut down and restarted: the table is now empty on master; the `DELETE FROM` empties it on slave too. Note that even with this fix, between the master's restart and the first use of the table on master, the slave still has out-of-date data in the table. But if you use the `--init-file` option to populate the `MEMORY` table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477)
- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.
- The MySQL server now returns an error if `SET SQL_LOG_BIN` is issued by a user without the `SUPER` privilege (in previous versions it just silently ignored the statement in this case).
- Changed that when the MySQL server has binary logging disabled (that is, no `--log-bin` option was used), then no transaction binary log cache is allocated for connections. This should save `binlog_cache_size` bytes of memory (32KB by default) for every connection.
- Added the `sync_binlog=N` global variable and startup option, which makes the MySQL server synchronize its binary log to disk (`fdatasync()`) after every Nth write to the binary log.
- Changed the slave SQL thread to print less useless error messages (no more message duplication; no more messages when an error is skipped because of `slave-skip-errors`).

- `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, single-table `DELETE`, and single-table `UPDATE` now are written to the binary log even if they changed nothing on the master (for example, even if a `DELETE` matched no rows). The old behavior sometimes caused bad surprises in replication setups.
- Replication and `mysqlbinlog` now have better support for the case that the session character set and collation variables are changed within a given session. See Sección 6.7, “Características de la replicación y problemas conocidos”.
- Killing a `CHECK TABLE` statement does not result in the table being marked as “corrupted” any more; the table remains as if `CHECK TABLE` had not even started. See Sección 13.5.5.3, “Sintaxis de KILL”.

Bugs fixed:

- Strange results with index (x, y) ... `WHERE x=val_1 AND y>=val_2 ORDER BY pk;` (Bug #3155)
- Subquery and order by (Bug #3118)
- `ALTER DATABASE` caused the client to hang if the database did not exist. (Bug #2333)
- `SLAVE START` (which is a deprecated syntax, `START SLAVE` should be used instead) could crash the slave. (Bug #2516)
- Multiple-table `DELETE` statements were never replicated by the slave if there were any `--replicate-*-table` options. (Bug #2527)
- The MySQL server did not report any error if a statement (submitted through `mysql_real_query()` or `mysql_stmt_prepare()`) was terminated by garbage characters. This can happen if you pass a wrong `length` parameter to these functions. The result was that the garbage characters were written into the binary log. (Bug #2703)
- Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, `OPTIMIZE TABLE` or `REPAIR TABLE`), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use `START SLAVE` to get replication going again. (Bug #1858)
- Made clearer the error message that one gets when an update is refused because of the `--read-only` option. (Bug #2757)
- Fixed that `--replicate-wild-*-table` rules apply to `ALTER DATABASE` when the table pattern is `%`, as is the case for `CREATE DATABASE` and `DROP DATABASE`. (Bug #3000)
- Fixed that when a `Rotate` event is found by the slave SQL thread in the middle of a transaction, the value of `Relay_Log_Pos` in `SHOW SLAVE STATUS` remains correct. (Bug #3017)
- Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #3015)
- Changed the column `Seconds_Behind_Master` in `SHOW SLAVE STATUS` to never show a value of -1. (Bug #2826)
- Changed that when a `DROP TEMPORARY TABLE` statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a `SELECT` on the master does not result in a superfluous error on the slave). (Bug #3063)
- Changed that when a thread handling `INSERT DELAYED` (also known as a `delayed_insert` thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081)
- Fixed deadlock when two `START SLAVE` commands were run at the same time. (Bug #2921)

- Fixed that a statement never triggers a superfluous error on the slave, if it must be excluded given the `--replicate-*` options. The bug was that if the statement had been killed on the master, the slave would stop. (Bug #2983)
- The `--local-load` option of `mysqlbinlog` now requires an argument.
- Fixed a segmentation fault when running `LOAD DATA FROM MASTER` after `RESET SLAVE`. (Bug #2922)
- `mysqlbinlog --read-from-remote-server` read all binary logs following the one that was requested. It now stops at the end of the requested file, the same as it does when reading a local binary log. There is an option `--to-last-log` to get the old behavior. (Bug #3204)
- Fixed `mysqlbinlog --read-from-remote-server` to print the exact positions of events in the "at #" lines. (Bug #3214)
- Fixed a rare error condition that caused the slave SQL thread spuriously to print the message `Binlog has bad magic number` and stop when it was not necessary to do so. (Bug #3401)
- Fixed `mysqlbinlog` not to forget to print a `USE` statement under rare circumstances where the binary log contained a `LOAD DATA INFILE` statement. (Bug #3415)
- Fixed a memory corruption when replicating a `LOAD DATA INFILE` when the master had version 3.23. (Bug #3422)
- Multiple-table `DELETE` statements were always replicated by the slave if there were some `--replicate-*-ignore-table` options and no `--replicate-*-do-table` options. (Bug #3461)
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (Bug #3568)
- Fixed that in some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357)
- If `server-id` was not set using startup options but with `SET GLOBAL`, the replication slave still complained that it was not set. (Bug #3829)
- `mysql_fix_privilege_tables` didn't correctly handle the argument of its `--password=#` option. (Bug #4240)
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017, CVE-2004-0836)
- During the installation process of the server RPM on Linux, `mysqld` was run as the `root` system user, and if you had `--log-bin=`*`somewhere_out_of_var_lib_mysql`* it created binary log files owned by `root` in this directory, which remained owned by `root` after the installation. This is now fixed by starting `mysqld` as the `mysql` system user instead. (Bug #4038)
- Made `DROP DATABASE` honor the value of `lower_case_table_names`. (Bug #4066)
- The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871)
- `mysqlbinlog` didn't escape the string content of user variables, and did not deal well when these variables were in non-ASCII character sets; this is now fixed by always printing the string content of user variables in hexadecimal. The character set and collation of the string is now also printed. (Bug #3875)
- Fixed incorrect destruction of expression that led to a server crash on complex `AND`/`OR` expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969, Bug #4494)

- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551)
- Fixed that when a multiple-table `DROP TABLE` failed to drop a table on the master server, the error code was not written to the binary log. (Bug #4553)
- When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326)

C.1.12. Cambios en la entrega 5.0.0 (22 Dec 2003: Alpha)

Functionality added or changed:

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because, for earlier versions, `InnoDB` is not aware of multiple tablespaces.
- Added support for `SUM(DISTINCT)`, `MIN(DISTINCT)`, and `MAX(DISTINCT)`.
- The `KILL` statement now takes `CONNECTION` and `QUERY` modifiers. The first is the same as `KILL` with no modifier (it kills a given connection thread). The second kills only the statement currently being executed by the connection.
- Added `TIMESTAMPADD()` and `TIMESTAMPDIFF()` functions.
- Added `WEEK` and `QUARTER` values as `INTERVAL` arguments for the `DATE_ADD()` and `DATE_SUB()` functions.
- New binary log format that enables replication of these session variables: `sql_mode`, `SQL_AUTO_IS_NULL`, `FOREIGN_KEY_CHECKS` (which was replicated since 4.0.14, but here it's done more efficiently and takes less space in the binary logs), `UNIQUE_CHECKS`. Other variables (like character sets, `SQL_SELECT_LIMIT`, ...) will be replicated in upcoming 5.0.x releases.
- Implemented Index Merge optimization for `OR` clauses. See Sección 7.2.6, “Index Merge Optimization”.
- Basic support for stored procedures (SQL:2003 style). See Capítulo 19, *Procedimientos almacenados y funciones*.
- Added `SELECT INTO` *`list_of_vars`*, which can be of mixed (that is, global and local) types. See Sección 19.2.9.3, “La sentencia `SELECT ... INTO`”.
- Easier replication upgrade (5.0.0 masters can read older binary logs and 5.0.0 slaves can read older relay logs). See Sección 6.5, “Compatibilidad entre versiones de MySQL con respecto a la replicación” for more details). The format of the binary log and relay log is changed compared to that of MySQL 4.1 and older.

Bugs fixed:

C.2. Cambios en MySQL Connector/ODBC (MyODBC)

C.2.1. Changes in Connector/ODBC 5.0.10 (14 December 2006)

Connector/ODBC 5.0.10 is the sixth BETA release.

Functionality added or changed:

- Added wide-string type info for `SQLGetTypeInfo()`.
- Added loose handling of retrieving some diagnostic data. (Bug #15782)

- Added initial support for removing braces when calling stored procedures and retrieving result sets from procedure calls. (Bug #24485)
- Added initial unicode support in data and metadata. (Bug #24837)
- Significant performance improvement when retrieving large text fields in pieces using `SQLGetData()` with a buffer smaller than the whole data. Mainly used in Access when fetching very large text fields. (Bug #24876)

Bugs fixed:

- String query parameters are new escaped correctly. (Bug #19078)
- Editing DSN no longer crashes ODBC data source administrator. (Bug #24675)

C.2.2. Changes in Connector/ODBC 5.0.9 (22 November 2006)

Connector/ODBC 5.0.9 is the fifth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Added recognition of `SQL_C_SHORT` and `SQL_C_TINYINT` as C types.
- Added support for column binding as SQL_NUMBERIC_STRUCT.

Bugs fixed:

- Fixed wildcard handling of and listing of catalogs and tables in `SQLTables`.
- Catch use of `SQL_ATTR_PARAMSET_SIZE` and report error until we fully support.
- Corrected retrieval multiple field types bit and blob/text.
- Fixed buffer length return for SQLDriverConnect.
- Added limit of display size when requested via `SQLColAttribute`/`SQL_DESC_DISPLAY_SIZE`.
- Fixed statistics to fail if it couldn't be completed.
- Fixed SQLGetData to clear the NULL indicator correctly during multiple calls.
- ODBC v2 behaviour in driver now supports ODBC v3 date/time types (since DriverManager maps them).

C.2.3. Changes in Connector/ODBC 5.0.8 (17 November 2006)

Connector/ODBC 5.0.8 is the fourth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Made distinction between `CHAR`/`BINARY` (and VAR versions).
- Wildcards now support escaped chars and underscore matching (needed to link tables with underscores in access).
- Also made `SQL_DESC_NAME` only fill in the name if there was a data pointer given, otherwise just the length.

- Fixed display size to be length if max length isn't available.

Bugs fixed:

- Length now used when handling bind parameter (needed in particular for `SQL_WCHAR`) - this enables updating char data in MS Access.
- Fixed string length to chars, not bytes, returned by SQLGetDiagRec.
- Fixed using wrong pointer for `SQL_MAX_DRIVER_CONNECTIONS` in `SQLGetInfo`.
- Fixed binding using `SQL_C_LONG`.
- Allow SQLDescribeCol to be called to retrieve the length of the column name, but not the name itself.
- Fix size return from `SQLDescribeCol`.
- Fixed hanlding of numeric pointers in SQLColAttribute.
- Fixed type returned for `MYSQL_TYPE_LONG` to `SQL_INTEGER` instead of `SQL_TINYINT`.
- Fixed MDiagnostic to use correct v2/v3 error codes.
- Set default return to `SQL_SUCCESS` if nothing is done for `SQLSpecialColumns`.
- Updated retrieval of descriptor fields to use the right pointer types.

C.2.4. Changes in Connector/ODBC 5.0.7 (08 November 2006)

Connector/ODBC 5.0.7 is the third BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Improved trace/log.
- Added support for `SQLStatistics` to `MYODBCShell`.

Bugs fixed:

- Fixed `SQLDescribeCol` returning column name length in bytes rather than chars.
- SQLBindParameter now handles `SQL_C_DEFAULT`.
- Corrected incorrect column index within `SQLStatistics`. Many more tables can now be linked into MS Access.

C.2.5. Changes in Connector/ODBC 5.0.6 (03 November 2006)

Connector/ODBC 5.0.6 is the second BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.
- Installation is provided in the form of a standard Microsoft System Installer (MSI).
- Connector/ODBC supports both `User` and `System` DSNs.

C.2.6. Changes in Connector/ODBC 5.0.5 (17 October 2006)

Connector/ODBC 5.0.5 is the first BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.

C.2.7. Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- The following ODBC API functions have been added in this release:
  - `SQLBindParameter`
  - `SQLBindCol`

C.2.8. Changes in Connector/ODBC 5.0.2 (Never released)

Connector/ODBC 5.0.2 was an internal implementation and testing release.

C.2.9. Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)

Features, limitations and notes on this release:

- Connector/ODBC 5.0 is Unicode aware.
- Connector/ODBC is currently limited to basic applications. ADO applications and Microsoft Office are not supported.
- Connector/ODBC must be used with a Driver Manager.
- The following ODBC API functions are implemented:
  - `SQLAllocHandle`
  - `SQLCloseCursor`
  - `SQLColAttribute`
  - `SQLColumns`
  - `SQLConnect`
  - `SQLCopyDesc`
  - `SQLDisconnect`
  - `SQLExecDirect`
  - `SQLExecute`

- SQLFetch
- SQLFreeHandle
- SQLFreeStmt
- SQLGetConnectAttr
- SQLGetData
- SQLGetDescField
- SQLGetDescRec
- SQLGetDiagField
- SQLGetDiagRec
- SQLGetEnvAttr
- SQLGetFunctions
- SQLGetStmtAttr
- SQLGetTypeInfo
- SQLNumResultCols
- SQLPrepare
- SQLRowcount
- SQLTables

The following ODBC API function are implemented, but not yet support all the available attributes/options:

- SQLSetConnectAttr
- SQLSetDescField
- SQLSetDescRec
- SQLSetEnvAttr
- SQLSetStmtAttr

C.2.10. Changes in Connector/ODBC 3.51.13 (Not yet released)

Functionality added or changed:

- Added support for the HENV handlers in SQLEndTran().
- Added auto-reconnect option to Connector/ODBC option parameters.
- Added auto is null option to Connector/ODBC option parameters. (Bug #10910)

Bugs fixed:

- Using `DataAdapter`, Connector/ODBC may continually consume memory when reading the same records within a loop (Windows Server 2003 SP1/SP2 only). (Bug #20459)
- Connector/ODBC may insert the wrong parameter values when using prepared statements under 64-bit Linux. (Bug #22446)
- Using Connector/ODBC, with `SQLBindCol` and binding the length to the return value from `SQL_LEN_DATA_AT_EXEC` fails with a memory allocation error. (Bug #20547)
- The `SQLDriverConnect()` ODBC method did not work with recent Connector/ODBC releases. (Bug #12393)

C.2.11. Cambios en MyODBC 3.51.12

Functionality added or changed:

- N/A

Bugs fixed:

- `SQLColumns()` returned no information for tables that had a column named using a reserved word. (Bug #9539)

C.2.12. Cambios en MyODBC 3.51.11

Functionality added or changed: No changes.

Bugs fixed:

- `mysql_list_dbcolumns()` and `insert_fields()` were retrieving all rows from a table. Fixed the queries generated by these functions to return no rows. (Bug #8198)
- `SQLGetTypoInfo()` returned `tinyblob` for `SQL_VARBINARY` and nothing for `SQL_BINARY`. Fixed to return `varbinary` for `SQL_VARBINARY`, `binary` for `SQL_BINARY`, and `longblob` for `SQL_LONGVARBINARY`. (Bug #8138)

C.3. Connector/NET Change History

C.3.1. Changes in MySQL Connector/NET Version 5.0.4 (Not yet released)

Bugs fixed:

- Incorrect values/formats would be applied when the `OldSyntax` connection string option was used. (Bug #25950)
- Registry would be incorrectly populated with installation locations. (Bug #25928)
- Filling a table schema through a stored procedure triggers a runtime error. (Bug #25609)
- `MySqlConnection` throws an exception when connecting to MySQL v4.1.7. (Bug #25726)
- Returned data types of a `DataTypes` collection do not contain the right correctl CLR Datatype. (Bug #25907)
- `GetSchema` and `DataTypes` would throw an exception due to an incorrect table name. (Bug #25906)
- Times with negative values would be returned incorrectly. (Bug #25912)

- `SELECT` did not work correctly when using a `WHERE` clause containing a UTF-8 string. (Bug #25651)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug #25614)
- When connecting to a MySQL Server earlier than version 4.1, the connection would hang when reading data. (Bug #25458)
- When connecting to a server, the return code from the connection could be zero, even though the hostname was incorrect. (Bug #24802)
- Using `ExecuteScalar()` with more than one query, where one query fails, will hang the connection. (Bug #25443)

C.3.2. Changes in MySQL Connector/NET Version 5.0.3 (05 January 2007)

Functionality added or changed:

- SSL support has been updated.
- The `ViewColumns GetSchema` collection has been updated.
- The `CommandBuilder.DeriveParameters` function has been updated to the procedure cache.
- Improved speed and performance by re-architecting certain sectionsof the code.
- Usage Advisor has been implemented. The Usage Advisor checks your queries and will report if you are usiing the connection inefficiently.
- The `MySqlCommand` object now supports asynchronous query methods. This is implemented useg the `BeginExecuteNonQuery` and `EndExecuteNonQuery` methods.
- Metadata from storaed procedures and stored function execution are cached.
- PerfMon hooks have been added to monitor the stored procedure cache hits and misses.
- The ShapZipLib library has been replaced with the deflate support provided within .NET 2.0.
- Support for the embedded server and client library have been removed from this release. Support will be added back to a later release.

Bugs fixed:

- An exception would be raised, or the process would hang, if `SELECT` privileges on a database were not granted and a stored procedure was used. (Bug #25033)
- Nested transactions (which are unsupported)do not raise an error or warning. (Bug #22400)
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. (Bug #25013)
- Additional text added to error message (Bug #25178)
- Stored procedure executions are not thread safe. (Bug #23905)
- Using `Driver.IsTooOld()` would return the wrong value. (Bug #24661)
- Deleting a connection to a disconnected server when using the Visual Studio Plugin would cause an assertion failure. (Bug #23687)

- When using a `DbNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DbNull.Value`. (Bug #24565)

C.3.3. Changes in MySQL Connector/NET Version 5.0.2 (06 November 2006)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

Bugs fixed:

- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. (Bug #23071)
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug #18186)
- Using Windows Vista (RC2) as a non-privileged user would raise a `Registry key 'Global' access denied`. (Bug #22882)
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve hostnames. (Bug #23758)
- Column names with accented characters were not parsed properly causing malformed column names in result sets. (Bug #23657)
- Connector/NET did not work as a data source for the `SqlDataSource` object used by ASP.NET 2.0. (Bug #16126)
- A `System.FormatException` exception would be raised when invoking a stored procedure with an `ENUM` input parameter. (Bug #23268)
- During installation, an antivirus error message would be raised (indicating a malicious script problem). (Bug #23245)
- An exception would be thrown when calling `GetSchemaTable` and `fields` was null. (Bug #23538)

C.3.4. Changes in MySQL Connector/NET Version 5.0.1 (01 October 2006)

Bugs fixed:

- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySqlConversionException` exception would be raised. (Bug #11991)
- An `MySql.Data.Types.MySqlConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug #9619)
- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. Bug (#14592)

- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a Connector/Net exception. (Bug #18391)
- Starting a transaction on a connection created by `MySql.Data.MySqlClient.MySqlClientFactory`, using `BeginTransaction` without specifying an isolation level, causes the SQL statement to fail with a syntax error. (Bug #22042)
- Connector/NET on a Tukish operating system, may fail to execute certain SQL statements correctly. (Bug #22452)
- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug #19994)
- The `MySqlexception` class is now derived from the `DbException` class. (Bug #21874)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug #7248)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug #21521)

C.3.5. Changes in MySQL Connector/NET Version 5.0.0 (08 August 2006)

This is a new Alpha development release, fixing recently discovered bugs.

NOTE: This Alpha release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at http://bugs.mysql.com/ for more details about the individual bugs fixed in this version.

Bugs fixed:

- CommandText: Question mark in comment line is being parsed as a parameter. (Bug #6214)

Functionality added or changed:

- Implemented Usage Advisor.
- Added Async query methods.
- Reimplemented PacketReader/PacketWriter support into `MySqlStream` class.
- Added internal implemention of SHA1 so we don't have to distribute the OpenNetCF on mobile devices.
- Added usage advisor warnings for requesting column values by the wrong type.
- Reworked connection string classes to be simpler and faster.
- Added procedure metadata caching.
- Added perfmon hooks for stored procedure cache hits and misses.
- Implemented `MySqlConnectionBuilder` class.
- Implemented `MySqlClientFactory` class.
- Implemented classes and interfaces for ADO.Net 2.0 support.
- Replaced use of ICSharpCode with .NET 2.0 internal deflate support.

- Refactored test suite to test all protocols in a single pass.
- Completely refactored how column values are handled to avoid boxing in some cases.

C.3.6. Changes in MySQL Connector/NET Version 1.0.9 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

Bugs fixed:

- Trying to fill a table schema through a stored procedure triggers a runtime error. (Bug #25609)
- `MySqlConnection` throws an exception when connecting to MySQL v4.1.7. (Bug #25726)
- `SELECT` did not work correctly when using a `WHERE` clause containing a UTF-8 string. (Bug #25651)
- Times with negative values would be returned incorrectly. (Bug #25912)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug #25614)
- When connecting to a server, the return code from the connection could be zero, even though the hostname was incorrect. (Bug #24802)
- Using `ExecuteScalar()` with more than one query, where one query fails, will hang the connection. (Bug #25443)
- Nested transactions do not raise an error or warning. (Bug #22400)
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. (Bug #25013)
- Additional text added to error message. (Bug #25178)
- The `CommandBuilder` would mistakenly add insert parameters for a table column with auto incrementation enabled. (Bug #23862)
- Stored procedure executions are not thread safe. (Bug #23905)
- Using `Driver.IsTooOld()` would return the wrong value. (Bug #24661)
- When using a `DbNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DbNull.Value`. (Bug #24565)
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug 18186)
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve hostnames. (Bug #23758)

- An `System.OverflowException` would be raised when accessing a varchar field over 255 bytes. Bug (#23749)

C.3.7. Changes in MySQL Connector/NET Version 1.0.8 (20 October 2006)

Bugs fixed:

- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySqlConversionException` exception would be raised. (Bug #11991)
- The `MySqlDateTime` class did not contain constructors. (Bug #15112)
- `DataReader` would show the value of the previous row (or last row with non-null data) if the current row contained a `datetime` field with a null value. (Bug #16884)
- An `MySql.Data.Types.MySqlConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug #9619)
- When using `MySqlDataAdapter`, connections to a MySQL server may remain open and active, even though the use of the connection has been completed and the data received. (Bug #8131)
- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. Bug (#14592)
- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a Connector/Net exception. (Bug #18391)
- Connector/NET on a Tukish operating system, may fail to execute certain SQL statements correctly. (Bug #22452)
- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug #19994)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug #7248)
- Called `MySqlCommandBuilder.DeriveParameters` for a stored procedure that has no paramers would cause an application crash. (Bug #15077)
- A `SELECT` query on a table with a date with a value of `'0000-00-00'` would hang the application. (Bug #17736)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug #21521)
- Calling `Close` on a connection after calling a stored procedure would trigger a `NullReferenceException`. (Bug #20581)
- `IDataRecord.GetString` would raise `NullPointerException` for null values in returned rows. Method now throws `SqlNullValueException`. (Bug #19294)
- An exception would be raised when using an output parameter to a `System.String` value. (Bug #17814)
- The DiscoverParameters function would fail when a stored procedure used a `NUMERIC` parameter type. (Bug #19515)
- When running a query that included a date comparison, a DateReader error would be raised. (Bug #19481)

- Parameter substitution in queries where the order of parameters and table fields did not match would substitute incorrect values. (Bug #19261)
- When working with multiple threads, character set initialization would generate errors. (Bug #17106)
- When using an unsigned 64-bit integer in a stored procedure, the unsigned bit would be lost stored. (Bug #16934)
- The connection string parser did not allow single or double quotes in the password. (Bug #16659)
- The CommandBuilder ignored Unsigned flag at Parameter creation. (Bug #17375)
- CHAR type added to MySqlDbType. (Bug #17749)
- Unsigned data types were not properly supported. (Bug #16788)

Functionality added or changed:

- Stored procedures are now cached.
- The method for retrieving stored procedured metadata has been changed so that users without `SELECT` privileges on the `mysql.proc` table can use a stored procedure.

C.3.8. Changes in MySQL Connector/NET Version 1.0.7 (21 November 2005)

- Unsigned `tinyint` (NET byte) would lead to and incorrectly determined parameter type from the parameter value. (Bug #18570)
- The parameter collection object's `Add()` method added parameters to the list without first checking to see whether they already existed. Now it updates the value of the existing parameter object if it exists. (Bug #13927)
- A `#42000Query was empty` exception occurred when executing a query built with `MySqlCommandBuilder`, if the query string ended with a semicolon. (Bug #14631)
- Implemented the `MySqlCommandBuilder.DeriveParameters` method that is used to discover the parameters for a stored procedure. (Bug #13632)
- Added support for the `cp932` character set. (Bug #13806)
- Calling a stored procedure where a parameter contained special characters (such as `'@'`) would produce an exception. Note that `ANSI_QUOTES` had to be enabled to make this possible. (Bug #13753)
- A statement that contained multiple references to the same parameter could not be prepared. (Bug #13541)
- The `Ping()` method did not update the `State` property of the `Connection` object. (Bug #13658)

C.3.9. Changes in MySQL Connector/NET Version 1.0.6 (03 October 2005)

- The `nant` build sequence had problems. (Bug #12978)
- Serializing a parameter failed if the first value passed in was `NULL`. (Bug #13276)
- Field names that contained the following characters caused errors: `()%<>/` (Bug #13036)
- The Connector/NET 1.0.5 installer would not install alongside Connector/NET 1.0.4. (Bug #12835)
- Connector/NET 1.0.5 could not connect on Mono. (Bug #13345)

C.3.10. Changes in MySQL Connector/NET Version 1.0.5 (29 August 2005)

- With multiple hosts in the connection string, Connector/NET would not connect to the last host in the list. (Bug #12628)
- Connector/NET interpreted the new decimal data type as a byte array. (Bug #11294)
- The `cp1250` character set was not supported. (Bug #11621)
- Connection could fail when .NET thread pool had no available worker threads. (Bug #10637)
- Decimal parameters caused syntax errors. (Bug #11550, Bug #10486, Bug #10152)
- A call to a stored procedure caused an exception if the stored procedure had no parameters. (Bug #11542)
- Certain malformed queries would trigger a `Connection must be valid and open` error message. (Bug #11490)
- The `MySqlCommandBuilder` class could not handle queries that referenced tables in a database other than the default database. (Bug #8382)
- Connector/NET could not work properly with certain regional settings. (WL#8228)
- Trying to use a stored procedure when `Connection.Database` was not populated generated an exception. (Bug #11450)
- Trying to read a `TIMESTAMP` column generated an exception. (Bug #7951)
- Parameters were not recognized when they were separated by linefeeds. (Bug #9722)
- Calling `MySqlConnection.clone` when a connection string had not yet been set on the original connection would generate an error. (Bug #10281)
- Added support to call a stored function from Connector/NET. (Bug #10644)
- Connector/NET could not connect to MySQL 4.1.14. (Bug #12771)
- The `ConnectionString` property could not be set when a `MySqlConnection` object was added with the designer. (Bug #12551, Bug #8724)

C.3.11. Changes in MySQL Connector/NET Version 1.0.4 (20 January 2005)

- Calling prepare causing exception. (Bug #7243)
- Fixed another small problem with prepared statements.
- `MySqlCommand.Connection` returns an IDbConnection. (Bug #7258)
- `MySqlAdapter.Fill` method throws error message `Non-negative number required`. (Bug #7345)
- Clone method bug in `MySqlCommand`. (Bug #7478)
- `MySqlDataReader.GetString(index)` returns non-Null value when field is `Null`. (Bug #7612)
- `MySqlReader.GetInt32` throws exception if column is unsigned. (Bug #7755)
- GetBytes is working no more. (Bug #7704).

- Quote character \222 not quoted in `EscapeString`. (Bug #7724)
- Fixed problem that causes named pipes to not work with some blob functionality.
- Fixed problem with shared memory connections.
- Problem with Multiple resultsets. (Bug #7436)
- Added or filled out several more topics in the API reference documentation.

C.3.12. Changes in MySQL Connector/NET Version 1.0.3-gamma (12 October 2004)

- Made MySQL the default named pipe name.
- Now `SHOW COLLATION` is used upon connection to retrieve the full list of charset ids.
- Fixed Invalid character set index: 200. (Bug #6547)
- Installer now includes options to install into GAC and create Start Menu items.
- Int64 Support in `MySqlCommand` Parameters. (Bug #6863)
- Connections now do not have to give a database on the connection string.
- `MySqlDataReader.GetChar(int i)` throws `IndexOutOfRange` exception. (Bug #6770)
- Fixed problem where multiple resultsets having different numbers of columns would cause a problem.
- Exception stack trace lost when re-throwing exceptions. (Bug #6983)
- Fixed major problem with detecting null values when using prepared statements.
- Errors in parsing stored procedure parameters. (Bug #6902)
- Integer "out" parameter from stored procedure returned as string. (Bug #6668)
- `MySqlDateTime` in Datatables sorting by Text, not Date. (Bug #7032)
- Invalid query string when using inout parameters (Bug #7133)
- Test suite fails with MySQL 4.0 because of case sensitivity of table names. (Bug #6831)
- Inserting `DateTime` causes `System.InvalidCastException` to be thrown. (Bug #7132)
- InvalidCast when using `DATE_ADD`-function. (Bug #6879)
- An Open Connection has been Closed by the Host System. (Bug #6634)
- Added `ServerThread` property to `MySqlConnection` to expose server thread id.
- Added Ping method to `MySqlConnection`.
- Changed the name of the test suite to `MySql.Data.Tests.dll`.

C.3.13. Changes in MySQL Connector/NET Version 1.0.2-gamma (15 November 2004)

- Fixed problem with MySqlBinary where string values could not be used to update extended text columns

- Fixed Installation directory ignored using custom installation (Bug #6329)
- Fixed problem where setting command text leaves the command in a prepared state
- Fixed double type handling in MySqlParameter(string parameterName, object value) (Bug #6428)
- Fixed Zero date "0000-00-00" is returned wrong when filling Dataset (Bug #6429)
- Fixed problem where calling stored procedures might cause an "Illegal mix of collations" problem.
- Added charset connection string option
- Fixed #HY000 Illegal mix of collations (latin1_swedish_ci,IMPLICIT) and (utf8_general_ (Bug #6322)
- Added the TableEditor CS and VB sample
- Fixed Charset-map for UCS-2 (Bug #6541)
- Updated the installer to include the new samples
- Fixed Long inserts take very long time (Bu #5453)
- Fixed Objects not being disposed (Bug #6649)
- Provider is now using character set specified by server as default

C.3.14. Changes in MySQL Connector/NET Version 1.0.1-beta2 (27 October 2004)

- Fixed BUG #5602 Possible bug in MySqlParameter(string, object) constructor
- Fixed BUG #5458 Calling GetChars on a longtext column throws an exception
- Fixed BUG #5474 cannot run a stored procedure populating mysqlcommand.parameters
- Fixed BUG #5469 Setting DbType throws NullReferenceException
- Fixed problem where connector was not issuing a CMD_QUIT before closing the socket
- Fixed BUG #5392 MySqlCommand sees "?" as parameters in string literals
- Fixed problem with ConnectionInternal where a key might be added more than once
- CP1252 is now used for Latin1 only when the server is 4.1.2 and later
- Fixed BUG #5388 DataReader reports all rows as NULL if one row is NULL
- Virtualized driver subsystem so future releases could easily support client or embedded server support
- Field buffers being reused to decrease memory allocations and increase speed
- Fixed problem where using old syntax while using the interfaces caused problems
- Using PacketWriter instead of Packet for writing to streams
- Refactored compression code into CompressedStream to clean up NativeDriver
- Added test case for resetting the command text on a prepared command

- Fixed problem where MySqlParameterCollection.Add() would throw unclear exception when given a null value (Bug #5621)
- Fixed construtor initialize problems in MySqlCommand() (Bug #5613)
- Fixed Parsing the ';' char (Bug #5876)
- Fixed missing Reference in DbType setter (Bug #5897)
- Fixed System.OverflowException when using YEAR datatype (Bug #6036)
- Added Aggregate function test (wasn't really a bug)
- Fixed serializing of floating point parameters (double, numeric, single, decimal) (Bug #5900)
- IsNullable error (Bug #5796)
- Fixed problem where connection lifetime on the connect string was not being respected
- Fixed problem where Min Pool Size was not being respected
- Fixed MySqlDataReader and 'show tables from ...' behavior (Bug #5256)
- Implemented SequentialAccess
- Fixed MySqlDateTime sets IsZero property on all subseq.records after first zero found (Bug #6006)
- Fixed Can't display Chinese correctly (Bug #5288)
- Fixed Russian character support as well
- Fixed Method TokenizeSql() uses only a limited set of valid characters for parameters (Bug #6217)
- Fixed NET Connector source missing resx files (Bug #6216)
- Fixed DBNull Values causing problems with retrieving/updating queries. (Bug #5798)
- Fixed Yet Another "object reference not set to an instance of an object" (Bug #5496)
- Fixed problem in PacketReader where it could try to allocate the wrong buffer size in EnsureCapacity
- Fixed GetBoolean returns wrong values (Bug #6227)
- Fixed IndexOutOfBounds when reading BLOB with DataReader with GetString(index) (Bug #6230)

C.3.15. Changes in MySQL Connector/NET Version 1.0.0 (01 September 2004)

- Thai encoding not correctly supported. (Bug #3889)
- Updated many of the test cases.
- Fixed problem with using compression.
- Bumped version number to 1.0.0 for beta 1 release.
- Added `COPYING.rtf` file for use in installer.
- Removed all of the XML comment warnings.
- Removed some last references to ByteFX.

C.3.16. Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)

- Added test fixture for prepared statements.
- All type classes now implement a `SerializeBinary` method for sending their data to a `PacketWriter`.
- Added `PacketWriter` class that will enable future low-memory large object handling.
- Fixed many small bugs in running prepared statements and stored procedures.
- Changed command so that an exception will not be throw in executing a stored procedure with parameters in old syntax mode.
- `SingleRow` behavior now working right even with limit.
- `GetBytes` now only works on binary columns.
- Logger now truncates long sql commands so blob columns don't blow out our log.
- host and database now have a default value of "" unless otherwise set.
- Connection Timeout seems to be ignored. (Bug #5214)
- Added test case for bug# 5051: GetSchema not working correctly.
- Fixed problem where `GetSchema` would return false for `IsUnique` when the column is key.
- `MySqlDataReader GetXXX` methods now using the field level `MySqlValue` object and not performing conversions.
- `DataReader` returning `NULL` for time column. (Bug #5097)
- Added test case for `LOAD DATA LOCAL INFILE`.
- Added replacetext custom nant task.
- Added `CommandBuilderTest` fixture.
- Added Last One Wins feature to `CommandBuilder`.
- Fixed persist security info case problem.
- Fixed `GetBool` so that 1, true, "true", and "yes" all count as true.
- Make parameter mark configurable.
- Added the "old syntax" connection string parameter to allow use of @ parameter marker.
- `MySqlCommandBuilder`. (Bug #4658)
- `ByteFX.MySqlClient` caches passwords if `Persist Security Info` is false. (Bug #4864)
- Updated license banner in all source files to include FLOSS exception.
- Added new .Types namespace and implementations for most current MySql types.
- Added `MySqlField41` as a subclass of `MySqlField`.
- Changed many classes to now use the new .Types types.

- Changed type `enum int` to `Int32`, `short` to `Int16`, and `bigint` to `Int64`.
- Added dummy types `UInt16`, `UInt32`, and `UInt64` to allow an unsigned parameter to be made.
- Connections are now reset when they are pulled from the connection pool.
- Refactored auth code in driver so it can be used for both auth and reset.
- Added `UserReset` test in `PoolingTests.cs`.
- Connections are now reset using `COM_CHANGE_USER` when pulled from the pool.
- Implemented `SingleResultSet` behavior.
- Implemented support of unicode.
- Added char set mappings for utf-8 and ucs-2.
- Time fields overflow using bytefx .net mysql driver (Bug #4520)
- Modified time test in data type test fixture to check for time spans where hours > 24.
- Wrong string with backslash escaping in `ByteFx.Data.MySqlClient.MySqlParameter`. (Bug #4505)
- Added code to Parameter test case TestQuoting to test for backslashes.
- `MySqlCommandBuilder` fails with multi-word column names. (Bug #4486)
- Fixed bug in `TokenizeSql` where underscore would terminate character capture in parameter name.
- Added test case for spaces in column names.
- `MySqlDataReader.GetBytes` don't works correctly. (Bug #4324)
- Added `GetBytes()` test case to `DataReader` test fixture.
- Now reading all server variables in `InternalConnection.Configure` into `Hashtable`.
- Now using `string[]` for index map in `CharSetMap`.
- Added CRInSQL test case for carriage returns in SQL.
- Setting maxPacketSize to default value in `Driver.ctor`.
- Setting `MySqlDbType` on a parameter doesn't set generic type. (Bug #4442)
- Removed obsolete data types `Long` and `LongLong`.
- Overflow exception thrown when using "use pipe" on connection string. (Bug #4071)
- Changed "use pipe" keyword to "pipe name" or just "pipe".
- Allow reading multiple resultsets from a single query.
- Added flags attribute to `ServerStatusFlags` enum.
- Changed name of `ServerStatus` enum to `ServerStatusFlags`.
- Inserted data row doesn't update properly.

- Error processing show create table. (Bug #4074)
- Change `Packet.ReadLenInteger` to `ReadPackedLong` and added `packet.ReadPackedInteger` that alwasy reads integers packed with 2,3,4.
- Added `syntax.cs` test fixture to test various SQL syntax bugs.
- Improper handling of time values. Now time value of 00:00:00 is not treated as null. (Bug #4149)
- Moved all test suite files into `TestSuite` folder.
- Fixed bug where null column would move the result packet pointer backward.
- Added new nant build script.
- Clear tablename so it will be regen'ed properly during the next `GenerateSchema`. (Bug #3917)
- `GetValues` was always returning zero and was also always trying to copy all fields rather than respecting the size of the array passed in. (Bug #3915)
- Implemented shared memory access protocol.
- Implemented prepared statements for MySQL 4.1.
- Implemented stored procedures for MySQL 5.0.
- Renamed `MySqlInternalConnection` to `InternalConnection`.
- SQL is now parsed as chars, fixes problems with other languages.
- Added logging and allow batch connection string options.
- `RowUpdating` event not set when setting the `DataAdapter` property. (Bug #3888)
- Fixed bug in char set mapping.
- Implemented 4.1 authentication.
- Improved open/auth code in driver.
- Improved how connection bits are set during connection.
- Database name is now passed to server during initial handshake.
- Changed namespace for client to `MySql.Data.MySqlClient`.
- Changed assembly name of client to `MySql.Data.dll`.
- Changed license text in all source files to GPL.
- Added the `MySqlClient.build` Nant file.
- Removed the mono batch files.
- Moved some of the unused files into notused folder so nant build file can use wildcards.
- Implemented shared memory accesss.
- Major revamp in code structure.
- Prepared statements now working for MySql 4.1.1 and later.

- Finished implementing auth for 4.0, 4.1.0, and 4.1.1.
- Changed namespace from `MySQL.Data.MySQLClient` back to `MySql.Data.MySqlClient`.
- Fixed bug in `CharSetMapping` where it was trying to use text names as ints.
- Changed namespace to `MySQL.Data.MySQLClient`.
- Integrated auth changes from UC2004.
- Fixed bug where calling any of the GetXXX methods on a datareader before or after reading data would not throw the appropriate exception (thanks Luca Morelli).
- Added `TimeSpan` code in parameter.cs to properly serialize a timespan object to mysql time format (thanks Gianluca Colombo).
- Added `TimeStamp` to parameter serialization code. Prevented `DataAdatper` updates from working right (thanks Michael King).
- Fixed a misspelling in `MySqlHelper.cs` (thanks Patrick Kristiansen).

C.3.17. Changes in MySQL Connector/NET Version 0.76

- Driver now using charset number given in handshake to create encoding.
- Changed command editor to point to `MySqlClient.Design`.
- Fixed bug in `Version.isAtLeast`.
- Changed `DBConnectionString` to support changes done to `MySqlConnectionString`.
- Removed `SqlCommandEditor` and `DataAdapterPreviewDialog`.
- Using new long return values in many places.
- Integrated new `CompressedStream` class.
- Changed `ConnectionString` and added attributes to allow it to be used in `MySqlClient.Design`.
- Changed `packet.cs` to support newer lengths in `ReadLenInteger`.
- Changed other classes to use new properties and fields of `MySqlConnectionString`.
- `ConnectionInternal` is now using PING to see whether the server is alive.
- Moved toolbox bitmaps into resource folder.
- Changed `field.cs` to allow values to come directly from row buffer.
- Changed to use the new driver.Send syntax.
- Using a new packet queueing system.
- Started work handling the "broken" compression packet handling.
- Fixed bug in `StreamCreator` where failure to connect to a host would continue to loop infinitly (thanks Kevin Casella).
- Improved connectstring handling.
- Moved designers into Pro product.

- Removed some old commented out code from `command.cs`.
- Fixed a problem with compression.
- Fixed connection object where an exception throw prior to the connection opening would not leave the connection in the connecting state (thanks Chris Cline).
- Added GUID support.
- Fixed sequence out of order bug (thanks Mark Reay).

C.3.18. Changes in MySQL Connector/NET Version 0.75

- Enum values now supported as parameter values (thanks Philipp Sumi).
- Year datatype now supported.
- Fixed compression.
- Fixed bug where a parameter with a `TimeSpan` as the value would not serialize properly.
- Fixed bug where default constructor would not set default connection string values.
- Added some XML comments to some members.
- Work to fix/improve compression handling.
- Improved `ConnectionString` handling so that it better matches the standard set by `SqlClient`.
- A `MySqlException` is now thrown if a username is not included in the connection string.
- Localhost is now used as the default if not specified on the connection string.
- An exception is now thrown if an attempt is made to set the connection string while the connection is open.
- Small changes to `ConnectionString` docs.
- Removed `MultiHostStream` and `MySqlStream`. Replaced it with `Common/StreamCreator`.
- Added support for Use Pipe connection string value.
- Added Platform class for easier access to platform utility functions.
- Fixed small pooling bug where new connection was not getting created after `IsAlive` fails.
- Added `Platform.cs` and `StreamCreator.cs`.
- Fixed `Field.cs` to properly handle 4.1 style timestamps.
- Changed `Common.Version` to `Common.DBVersion` to avoid name conflict.
- Fixed `field.cs` so that text columns return the right field type.
- Added `MySqlError` class to provide some reference for error codes (thanks Geert Veenstra).

C.3.19. Changes in MySQL Connector/NET Version 0.74

- Added Unix socket support (thanks Mohammad DAMT).

- Only calling `Thread.Sleep` when no data is available.
- Improved escaping of quote characters in parameter data.
- Removed misleading comments from `parameter.cs`.
- Fixed pooling bug.
- Fixed `ConnectionString` editor dialog (thanks marco p (pomarc)).
- `UserId` now supported in connection strings (thanks Jeff Neeley).
- Attempting to create a parameter that is not input throws an exception (thanks Ryan Gregg).
- Added much documentation.
- Checked in new `MultiHostStream` capability. Big thanks to Dan Guisinger for this. he originally submitted the code and idea of supporting multiple machines on the connect string.
- Added a lot of documentation.
- Fixed speed issue with 0.73.
- Changed to Thread.Sleep(0) in MySqlDataStream to help optimize the case where it doesn't need to wait (thanks Todd German).
- Prepopulating the idlepools to `MinPoolSize`.
- Fixed `MySqlPool` deadlock condition as well as stupid bug where CreateNewPooledConnection was not ever adding new connections to the pool. Also fixed `MySqlStream.ReadBytes` and `ReadByte` to not use `TicksPerSecond` which does not appear to always be right. (thanks Matthew J. Peddlesden)
- Fix for precision and scale (thanks Matthew J. Peddlesden).
- Added `Thread.Sleep(1)` to stream reading methods to be more cpu friendly (thanks Sean McGinnis).
- Fixed problem where `ExecuteReader` would sometime return null (thanks Lloyd Dupont).
- Fixed major bug with null field handling (thanks Naucki).
- Enclosed queries for `max_allowed_packet` and `characterset` inside try catch (and set defaults).
- Fixed problem where socket was not getting closed properly (thanks Steve!).
- Fixed problem where `ExecuteNonQuery` was not always returning the right value.
- Fixed `InternalConnection` to not use `@@session.max_allowed_packet` but use `@@max_allowed_packet`. (Thanks Miguel)
- Added many new XML doc lines.
- Fixed sql parsing to not send empty queries (thanks Rory).
- Fixed problem where the reader was not unpeeking the packet on close.
- Fixed problem where user variables were not being handled (thanks Sami Vaaraniemi).
- Fixed loop checking in the MySqlPool (thanks Steve M. Brown)
- Fixed `ParameterCollection.Add` method to match `SqlClient` (thanks Joshua Mouch).

- Fixed `ConnectionString` parsing to handle no and yes for boolean and not lowercase values (thanks Naucki).
- Added `InternalConnection` class, changes to pooling.
- Implemented Persist Security Info.
- Added `security.cs` and `version.cs` to project
- Fixed `DateTime` handling in `Parameter.cs` (thanks Burkhard Perkens-Golomb).
- Fixed parameter serialization where some types would throw a cast exception.
- Fixed `DataReader` to convert all returned values to prevent casting errors (thanks Keith Murray).
- Added code to `Command.ExecuteReader` to return null if the initial SQL command throws an exception (thanks Burkhard Perkens-Golomb).
- Fixed `ExecuteScalar` bug introduced with restructure.
- Restructure to allow for `LOCAL DATA INFILE` and better sequencing of packets.
- Fixed several bugs related to restructure.
- Early work done to support more secure passwords in Mysql 4.1. Old passwords in 4.1 not supported yet.
- Parameters appearing after system parameters are now handled correctly (Adam M. (adammil)).
- Strings can now be assigned directly to blob fields (Adam M.).
- Fixed float parameters (thanks Pent).
- Improved Parameter constructor and `ParameterCollection.Add` methods to better match SqlClient (thanks Joshua Mouch).
- Corrected `Connection.CreateCommand` to return a `MySqlCommand` type.
- Fixed connection string designer dialog box problem (thanks Abraham Guyt).
- Fixed problem with sending commands not always reading the response packet (thanks Joshua Mouch).
- Fixed parameter serialization where some blobs types were not being handled (thanks Sean McGinnis).
- Removed spurious `MessageBox.show` from `DataReader` code (thanks Joshua Mouch).
- Fixed a nasty bug in the split sql code (thanks everyone!).

C.3.20. Changes in MySQL Connector/NET Version 0.71

- Fixed bug in `MySqlStream` where too much data could attempt to be read (thanks Peter Belbin)
- Implemented `HasRows` (thanks Nash Pherson).
- Fixed bug where tables with more than 252 columns cause an exception (thanks Joshua Kessler).
- Fixed bug where SQL statements ending in ; would cause a problem (thanks Shane Krueger).
- Fixed bug in driver where error messages were getting truncated by 1 character (thanks Shane Krueger).

- Made `MySqlException` serializable (thanks Mathias Hasselmann).

C.3.21. Changes in MySQL Connector/NET Version 0.70

- Updated some of the character code pages to be more accurate.
- Fixed problem where readers could be opened on connections that had readers open.
- Moved test to separate assembly `MySqlClientTests`.
- Fixed stupid problem in driver with sequence out of order (Thanks Peter Belbin).
- Added some pipe tests.
- Increased default max pool size to 50.
- Compiles with Mono 0-24.
- Fixed connection and data reader dispose problems.
- Added `String` datatype handling to parameter serialization.
- Fixed sequence problem in driver that occurred after thrown exception (thanks Burkhard Perkens-Golomb).
- Added support for `CommandBehavior.SingleRow` to `DataReader`.
- Fixed command sql processing so quotes are better handled (thanks Theo Spears).
- Fixed parsing of double, single, and decimal values to account for non-English separators. You still have to use the right syntax if you using hard coded sql, but if you use parameters the code will convert floating point types to use '.' appropriately internal both into the server and out.
- Added `MySqlStream` class to simplify timeouts and driver coding.
- Fixed `DataReader` so that it is closed properly when the associated connection is closed. [thanks smishra]
- Made client more SqlClient compliant so that DataReaders have to be closed before the connection can be used to run another command.
- Improved `DBNull.Value` handling in the fields.
- Added several unit tests.
- Fixed `MySqlException` base class.
- Improved driver coding
- Fixed bug where NextResult was returning false on the last resultset.
- Added more tests for MySQL.
- Improved casting problems by equating unsigned 32bit values to Int64 and usigned 16bit values to Int32, and so forth.
- Added new constructor for `MySqlParameter` for (name, type, size, srccol)
- Fixed bug in `MySqlDataReader` where it didn't check for null fieldlist before returning field count.

- Started adding `MySqlClient` unit tests (added `MySqlClient/Tests` folder and some test cases).
- Fixed some things in Connection String handling.
- Moved `INIT_DB` to `MySqlPool`. I may move it again, this is in preparation of the conference.
- Fixed bug inside `CommandBuilder` that prevented inserts from happening properly.
- Reworked some of the internals so that all three execute methods of Command worked properly.
- Fixed many small bugs found during benchmarking.
- The first cut of `CoonectionPooling` is working. "min pool size" and "max pool size" are respected.
- Work to enable multiple resultsets to be returned.
- Character sets are handled much more intelligently now. The driver queries MySQL at startup for the default character set. That character set is then used for conversions if that code page can be loaded. If not, then the default code page for the current OS is used.
- Added code to save the inferred type in the name,value constructor of `Parameter`.
- Also, inferred type if value of null parameter is changed using `Value` property.
- Converted all files to use proper Camel case. MySQL is now MySql in all files. PgSQL is now PgSql.
- Added attribute to PgSql code to prevent designer from trying to show.
- Added `MySQLDbType` property to Parameter object and added proper conversion code to convert from `DbType` to `MySQLDbType`).
- Removed unused `ObjectToString` method from `MySQLParameter.cs`.
- Fixed `Add(..)` method in `ParameterCollection` so that it doesn't use `Add(name, value)` instead.
- Fixed `IndexOf` and `Contains` in `ParameterCollection` to be aware that parameter names are now stored without @.
- Fixed `Command.ConvertSQLToBytes` so it only allows characters that can be in MySQL variable names.
- Fixed `DataReader` and `Field` so that blob fields read their data from `Field.cs` and `GetBytes` works right.
- Added simple query builder editor to `CommandText` property of `MySQLCommand`.
- Fixed `CommandBuilder` and `Parameter` serialization to account for Parameters not storing @ in their names.
- Removed `MySQLFieldType` enum from Field.cs. Now using `MySQLDbType` enum.
- Added `Designer` attribute to several classes to prevent designer view when using VS.Net.
- Fixed Initial catalog typo in `ConnectionString` designer.
- Removed 3 parameter constructor for `MySQLParameter` that conflicted with (name, type, value).
- Changed `MySQLParameter` so `paramName` is now stored without leading @ (this fixed null inserts when using designer).

- Changed `TypeConverter` for `MySQLParameter` to use the constructor with all properties.

C.3.22. Changes in MySQL Connector/NET Version 0.68

- Fixed sequence issue in driver.
- Added `DbParametersEditor` to make parameter editing more like `SqlClient`.
- Fixed `Command` class so that parameters can be edited using the designer
- Update connection string designer to support `Use Compression` flag.
- Fixed string encoding so that European characters like ä will work correctly.
- Creating base classes to aid in building new data providers.
- Added support for UID key in connection string.
- Field, parameter, command now using DBNull.Value instead of null.
- `CommandBuilder` using `DBNull.Value`.
- `CommandBuilder` now builds insert command correctly when an auto_insert field is not present.
- Field now uses typeof keyword to return `System.Types` (performance).

C.3.23. Changes in MySQL Connector/NET Version 0.65

- `MySQLCommandBuilder` now implemented.
- Transaction support now implemented (not all table types support this).
- `GetSchemaTable` fixed to not use xsd (for Mono).
- Driver is now Mono-compatible.
- TIME data type now supported.
- More work to improve Timestamp data type handling.
- Changed signatures of all classes to match corresponding `SqlClient` classes.

C.3.24. Changes in MySQL Connector/NET Version 0.60

- Protocol compression using SharpZipLib (www.icsharpcode.net).
- Named pipes on Windows now working properly.
- Work done to improve `Timestamp` data type handling.
- Implemented `IEnumerable` on `DataReader` so `DataGrid` would work.

C.3.25. Changes in MySQL Connector/NET Version 0.50

- Speed increased dramatically by removing bugging network sync code.
- Driver no longer buffers rows of data (more ADO.Net compliant).
- Conversion bugs related to `TIMESTAMP` and `DATETIME` fields fixed.

C.4. MySQL Visual Studio Plugin Change History

C.4.1. Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)

Bugs fixed:

- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. (Bug #23071)
- The Add Connection dialog of the Server Explorer would freeze when accessing databases with capitalized characters in their name. (Bug #24875)

C.4.2. Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)

This is a bug fix release to resolve an incompatibility issue with Connector/NET 5.0.1.

It is critical that this release only be used with Connector/NET 5.0.1. After installing Connector/NET 5.0.1, you will need to make a small change in your machine.config file. This file should be located at `%win%\Microsoft.Net\Framework\v2.0.50727\CONFIG\machine.config` (`%win%` should be the location of your Windows folder). Near the bottom of the file you will see a line like this:

```
<add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data"/>
```

It needs to be changed to be like this:

```
<add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data,
Version=5.0.1.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
```

C.4.3. Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)

This is the first beta release.

Features in this release:

- DDEX (Data Designer Extensibility) compatibility.
- Ability to work with MySQL objects (tables, views, stored procedures, etc) from within Server Explorer.

C.5. MySQL Connector/J Change History

C.5.1. Changes in MySQL Connector/J 5.1.x

C.5.1.1. Changes in MySQL Connector/J 5.1.0 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is false (i.e. Connector/J does not use server-side prepared statements).

C.5.2. Changes in MySQL Connector/J 5.0.x

C.5.2.1. Changes in MySQL Connector/J 5.0.5 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is false (i.e. Connector/J does not use server-side prepared statements).

Bugs fixed:

- Comments auomatically generated by the Hibernate framework could cause problems in the parser when using batched inserts. (Bug #25025)

C.5.2.2. Changes in MySQL Connector/J 5.0.4 (20 October 2006)

Bugs fixed:

- Column names don't match metadata in cases where server doesn't return original column names (column functions) thus breaking compatibility with applications that expect 1-1 mappings between findColumn() and rsmd.getColumnName(), usually manifests itself as "Can't find column (")" exceptions. (Bug #21379)
- When using information_schema for metadata, COLUMN_SIZE for getColumns() is not clamped to range of java.lang.Integer as is the case when not using information_schema, thus leading to a truncation exception that isn't present when not using information_schema. (Bug #21544)
- Newlines causing whitespace to span confuse procedure parser when getting parameter metadata for stored procedures. (Bug #22024)
- Driver was using milliseconds for Statement.setQueryTimeout() when specification says argument is to be in seconds. (Bug #22359)
- Workaround for server crash when calling stored procedures via a server-side prepared statement (driver now detects prepare(stored procedure) and substitutes client-side prepared statement). (Bug #22297)
- Added new _ci collations to CharsetMapping - utf8_unicode_ci not working. (Bug #22456)
- Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement). (Bug #22290)
- DBMD.getColumns() does not return expected COLUMN_SIZE for the SET type, now returns length of largest possible set disregarding whitespace or the "," delimitters to be consistent with the ODBC driver. (Bug #22613)

Other changes:

- Fixed configuration property `jdbcCompliantTruncation` was not being used for reads of result set values.

- Driver now supports `{call sp}` (without "()" if procedure has no arguments).
- Driver now sends numeric 1 or 0 for client-prepared statement `setBoolean()` calls instead of '1' or '0'.
- DatabaseMetaData correctly reports true for `supportsCatalog*()` methods.

C.5.2.3. Changes in MySQL Connector/J 5.0.3 (26 July 2006)

- Fixed `Statement.cancel()` causes `NullPointerException` if underlying connection has been closed due to server failure. (Bug #20650)
- Added configuration option `noAccessToProcedureBodies` which will cause the driver to create basic parameter metadata for `CallableStatements` when the user does not have access to procedure bodies via `SHOW CREATE PROCEDURE` or selecting from `mysql.proc` instead of throwing an exception. The default value for this option is `false`

Bugs fixed:

- If the connection to the server has been closed due to a server failure, then the cleanup process will call `Statement.cancel()`, triggering a `NullPointerException`, even though there is no active connection. (Bug #20650)

C.5.2.4. Changes in MySQL Connector/J 5.0.2-beta (11 July 2006)

- Fixed can't use `XAConnection` for local transactions when no global transaction is in progress. (Bug #17401)
- Fixed driver fails on non-ASCII platforms. The driver was assuming that the platform character set would be a superset of MySQL's `latin1` when doing the handshake for authentication, and when reading error messages. We now use Cp1252 for all strings sent to the server during the handshake phase, and a hard-coded mapping of the `language` systtem variable to the character set that is used for error messages. (Bug #18086)
- Fixed `ConnectionProperties` (and thus some subclasses) are not serializable, even though some J2EE containers expect them to be. (Bug #19169)
- Fixed `MysqlValidConnectionChecker` for JBoss doesn't work with `MySQLXADataSources`. (Bug #20242)
- Better caching of character set converters (per-connection) to remove a bottleneck for multibyte character sets.
- Added connection/datasource property `pinGlobalTxToPhysicalConnection` (defaults to `false`). When set to `true`, when using `XAConnections`, the driver ensures that operations on a given XID are always routed to the same physical connection. This allows the `XAConnection` to support `XA START ... JOIN` after `XA END` has been called, and is also a workaround for transaction managers that don't maintain thread affinity for a global transaction (most either always maintain thread affinity, or have it as a configuration option).
- `MysqlXaConnection.recover(int flags)` now allows combinations of `XAResource.TMSTARTRSCAN` and `TMENDRSCAN`. To simulate the “scanning” nature of the interface, we return all prepared XIDs for `TMSTARTRSCAN`, and no new XIDs for calls with `TMNOFLAGS`, or `TMENDRSCAN` when not in combination with `TMSTARTRSCAN`. This change was made for API compliance, as well as integration with IBM WebSphere's transaction manager.

C.5.2.5. Changes in MySQL Connector/J 5.0.1-beta (Not Released)

Not released due to a packaging error

C.5.2.6. Changes in MySQL Connector/J 5.0.0-beta (22 December 2005)

- `XADataSource` implemented (ported from 3.2 branch which won't be released as a product). Use `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` as your datasource class name in your application server to utilize XA transactions in MySQL-5.0.10 and newer.
- `PreparedStatement.setString()` didn't work correctly when `sql_mode` on server contained `NO_BACKSLASH_ESCAPES` and no characters that needed escaping were present in the string.
- Attempt detection of the MySQL type `BINARY` (it's an alias, so this isn't always reliable), and use the `java.sql.Types.BINARY` type mapping for it.
- Moved `-bin-g.jar` file into separate `debug` subdirectory to avoid confusion.
- Don't allow `.setAutoCommit(true)`, or `.commit()` or `.rollback()` on an XA-managed connection as per the JDBC specification.
- If the connection `useTimezone` is set to `true`, then also respect time zone conversions in escape-processed string literals (for example, `"{ts ...}"` and `"{t ...}"`).
- Return original column name for `RSMD.getColumnName()` if the column was aliased, alias name for `.getColumnLabel()` (if aliased), and original table name for `.getTableName()`. Note this only works for MySQL-4.1 and newer, as older servers don't make this information available to clients.
- Setting `useJDBCCompliantTimezoneShift=true` (it's not the default) causes the driver to use GMT for *all* `TIMESTAMP`/`DATETIME` time zones, and the current VM time zone for any other type that refers to time zones. This feature can not be used when `useTimezone=true` to convert between server and client time zones.
- Add one level of indirection of internal representation of `CallableStatement` parameter metadata to avoid class not found issues on JDK-1.3 for `ParameterMetadata` interface (which doesn't exist prior to JDBC-3.0).
- Added unit tests for `XADatasource`, as well as friendlier exceptions for XA failures compared to the "stock" `XAException` (which has no messages).
- Idle timeouts cause `XAConnections` to whine about rolling themselves back. (Bug #14729)
- Added support for Connector/MXJ integration via url subprotocol `jdbc:mysql:mxj://...`.
- Moved all `SQLException` constructor usage to a factory in `SQLError` (ground-work for JDBC-4.0 `SQLState`-based exception classes).
- Removed Java5-specific calls to `BigDecimal` constructor (when result set value is `''`, `(int)0` was being used as an argument indirectly via method return value. This signature doesn't exist prior to Java5.)
- Added service-provider entry to `META-INF/services/java.sql.Driver` for JDBC-4.0 support.
- Return "[VAR]BINARY" for `RSMD.getColumnTypeName()` when that is actually the type, and it can be distinguished (MySQL-4.1 and newer).
- When fix for Bug #14562 was merged from 3.1.12, added functionality for `CallableStatement`'s parameter metadata to return correct information for `.getParameterClassName()`.
- Fuller synchronization of `Connection` to avoid deadlocks when using multithreaded frameworks that multithread a single connection (usually not recommended, but the JDBC spec allows it anyways), part of fix to Bug #14972).

- Implementation of `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL-5.0.0 or newer server, require a separate connection to issue the `KILL QUERY` statement, and in the case of `setQueryTimeout()` creates an additional thread to handle the timeout functionality.

  Note: Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeExceptions` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

C.5.3. Changes in MySQL Connector/J 3.1.x

C.5.3.1. Changes in MySQL Connector/J 3.1.15 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is false (i.e. Connector/J does not use server-side prepared statements).

C.5.3.2. Changes in MySQL Connector/J 3.1.14 (10-19-2006)

- Fixed updatable result set throws ClassCastException when there is row data and moveToInsertRow() is called. (Fixes Bug#20479)
- Fixed Updatable result set that contains a BIT column fails when server-side prepared statements are used. (Fixes Bug#20485)
- Fixed memory leak with profileSQL=true. (Fixes Bug#16987)
- Connection fails to localhost when using timeout and IPv6 is configured. (Fixes Bug#19726)
- Fixed NullPointerException in MysqlDataSourceFactory due to Reference containing RefAddrs with null content. (Fixes Bug#16791)
- Fixed ResultSet.getShort() for UNSIGNED TINYINT returns incorrect values when using server-side prepared statements. (Fixes Bug#20306)
- Fixed can't pool server-side prepared statements, exception raised when re-using them. (Fixes Bug#20687)
- Fixed BUG#21062 - ResultSet.getSomeInteger() doesn't work for BIT(>1).
- Fixed BUG#18880 - ResultSet.getFloatFromString() can't retrieve values near Float.MIN/MAX_VALUE.
- Fixed BUG#20888 - escape of quotes in client-side prepared statements parsing not respected. Patch covers more than bug report, including NO_BACKSLASH_ESCAPES being set, and stacked quote characters forms of escaping (i.e. '' or "").
- Fixed BUG#19993 - ReplicationDriver does not always round-robin load balance depending on URL used for slaves list.
- Fixed calling toString() on ResultSetMetaData for driver-generated (i.e. from DatabaseMetaData method calls, or from getGeneratedKeys()) result sets would raise a NullPointerException.

- Fixed Bug#21207 - Driver throws NPE when tracing prepared statements that have been closed (in asSQL()).
- Removed logger autodectection altogether, must now specify logger explicitly if you want to use a logger other than one that logs to STDERR.
- Fixed BUG#22290 - Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement).
- Driver now sends numeric 1 or 0 for client-prepared statement setBoolean() calls instead of '1' or '0'.
- Fixed bug where driver would not advance to next host if roundRobinLoadBalance=true and the last host in the list is down.
- Fixed BUG#18258 - DatabaseMetaData.getTables(), columns() with bad catalog parameter threw exception rather than return empty result set (as required by spec).
- Check and store value for continueBatchOnError property in constructor of Statements, rather than when executing batches, so that Connections closed out from underneath statements don't cause NullPointerExceptions when it's required to check this property.
- Fixed bug when calling stored functions, where parameters weren't numbered correctly (first parameter is now the return value, subsequent parameters if specified start at index "2").

C.5.3.3. Changes in MySQL Connector/J 3.1.13 (26 May 2006)

- `INOUT` parameter does not store `IN` value. (Bug #15464)
- Exception thrown for new decimal type when using updatable result sets. (Bug #14609)
- No "dos" character set in MySQL > 4.1.0. (Bug #15544)
- `PreparedStatement.setObject()` serializes `BigInteger` as object, rather than sending as numeric value (and is thus not complementary to `.getObject()` on an `UNSIGNED LONG` type). (Bug #15383)
- `ResultSet.getShort()` for `UNSIGNED TINYINT` returned wrong values. (Bug #11874)
- `lib-nodist` directory missing from package breaks out-of-box build. (Bug #15676)
- `DBMD.getColumns()` returns wrong type for `BIT`. (Bug #15854)
- Fixed issue where driver was unable to initialize character set mapping tables. Removed reliance on `.properties` files to hold this information, as it turns out to be too problematic to code around class loader hierarchies that change depending on how an application is deployed. Moved information back into the `CharsetMapping` class. (Bug #14938)
- Fixed updatable result set doesn't return `AUTO_INCREMENT` values for `insertRow()` when multiple column primary keys are used. (the driver was checking for the existence of single-column primary keys and an autoincrement value > 0 instead of a straightforward `isAutoIncrement()` check). (Bug #16841)
- Fixed `Statement.getGeneratedKeys()` throws `NullPointerException` when no query has been processed. (Bug #17099)
- Fixed driver trying to call methods that don't exist on older and newer versions of Log4j. The fix is not trying to auto-detect presense of log4j, too many different incompatible versions out there in the wild to do this reliably. (Bug #13469)

If you relied on autodetection before, you will need to add "logger=com.mysql.jdbc.log.Log4JLogger" to your JDBC URL to enable Log4J usage, or alternatively use the new "CommonsLogger" class to take care of this.

- Added support for Apache Commons logging, use "com.mysql.jdbc.log.CommonsLogger" as the value for the "logger" configuration property.
- LogFactory now prepends "com.mysql.jdbc.log" to log class name if it can't be found as-specified. This allows you to use "short names" for the built-in log factories, for example "logger=CommonsLogger" instead of "logger=com.mysql.jdbc.log.CommonsLogger".
- Fixed issue with `ReplicationConnection` incorrectly copying state, doesn't transfer connection context correctly when transitioning between the same read-only states. (Bug #15570)
- Fixed issue where server-side prepared statements don't cause truncation exceptions to be thrown when truncation happens. (Bug #18041)
- Added performance feature, re-writing of batched executes for `Statement.executeBatch()` (for all DML statements) and `PreparedStatement.executeBatch()` (for INSERTs with VALUE clauses only). Enable by using "rewriteBatchedStatements=true" in your JDBC URL.
- Fixed `CallableStatement.registerOutParameter()` not working when some parameters pre-populated. Still waiting for feedback from JDBC experts group to determine what correct parameter count from `getMetaData()` should be, however. (Bug #17898)
- Fixed calling `clearParameters()` on a closed prepared statement causes NPE. (Bug #17587)
- Map "latin1" on MySQL server to CP1252 for MySQL > 4.1.0.
- Added additional accessor and mutator methods on ConnectionProperties so that DataSource users can use same naming as regular URL properties.
- Fixed data truncation and `getWarnings()` only returns last warning in set. (Bug #18740)
- Improved performance of retrieving `BigDecimal`, `Time`, `Timestamp` and `Date` values from server-side prepared statements by creating fewer short-lived instances of `Strings` when the native type is not an exact match for the requested type. Fixes Bug #18496 for `BigDecimals`.
- Fixed aliased column names where length of name > 251 are corrupted. (Bug #18554)
- Fixed `ResultSet.wasNull()` not always reset correctly for booleans when done via conversion for server-side prepared statements. (Bug #17450)
- Fixed invalid classname returned for `ResultSetMetaData.getColumnClassName()` for `BIGINT type`. (Bug #19282)
- Fixed case where driver wasn't reading server status correctly when fetching server-side prepared statement rows, which in some cases could cause warning counts to be off, or multiple result sets to not be read off the wire.
- Driver now aware of fix for `BIT` type metadata that went into MySQL-5.0.21 for server not reporting length consistently (Bug #13601).
- `Fixed PreparedStatement.setObject(int, Object, int)` doesn't respect scale of BigDecimals. (Bug #19615)
- Fixed `ResultSet.wasNull()` returns incorrect value when extracting native string from server-side prepared statement generated result set. (Bug #19282)

C.5.3.4. Changes in MySQL Connector/J 3.1.12 (30 November 2005)

- Fixed client-side prepared statement bug with embedded `?` characters inside quoted identifiers (it was recognized as a placeholder, when it was not).
- Don't allow `executeBatch()` for `CallableStatements` with registered `OUT`/`INOUT` parameters (JDBC compliance).
- Fall back to platform-encoding for `URLDecoder.decode()` when parsing driver URL properties if the platform doesn't have a two-argument version of this method.
- Java type conversion may be incorrect for `MEDIUMINT`. (Bug #14562)
- Added configuration property `useGmtMillisForDatetimes` which when set to `true` causes `ResultSet.getDate()`, `.getTimestamp()` to return correct millis-since GMT when `.getTime()` is called on the return value (currently default is `false` for legacy behavior).
- Fixed `DatabaseMetaData.stores*Identifiers()`:
  - If `lower_case_table_names=0` (on server):
    - `storesLowerCaseIdentifiers()` returns `false`
    - `storesLowerCaseQuotedIdentifiers()` returns `false`
    - `storesMixedCaseIdentifiers()` returns `true`
    - `storesMixedCaseQuotedIdentifiers()` returns `true`
    - `storesUpperCaseIdentifiers()` returns `false`
    - `storesUpperCaseQuotedIdentifiers()` returns `true`
  - If `lower_case_table_names=1` (on server):
    - `storesLowerCaseIdentifiers()` returns `true`
    - `storesLowerCaseQuotedIdentifiers()` returns `true`
    - `storesMixedCaseIdentifiers()` returns `false`
    - `storesMixedCaseQuotedIdentifiers() returns false`
    - `storesUpperCaseIdentifiers()` returns `false`
    - `storesUpperCaseQuotedIdentifiers()` returns `true`
- `DatabaseMetaData.getColumns()` doesn't return `TABLE_NAME` correctly. (Bug #14815)
- Escape processor replaces quote character in quoted string with string delimiter. (Bug #14909)
- OpenOffice expects `DBMD.supportsIntegrityEnhancementFacility()` to return `true` if foreign keys are supported by the datasource, even though this method also covers support for check constraints, which MySQL *doesn't* have. Setting the configuration property `overrideSupportsIntegrityEnhancementFacility` to `true` causes the driver to return `true` for this method. (Bug #12975)
- Added `com.mysql.jdbc.testsuite.url.default` system property to set default JDBC url for testsuite (to speed up bug resolution when I'm working in Eclipse).

- Unable to initialize character set mapping tables (due to J2EE classloader differences). (Bug #14938)
- Deadlock while closing server-side prepared statements from multiple threads sharing one connection. (Bug #14972)
- `logSlowQueries` should give better info. (Bug #12230)
- Extraneous sleep on `autoReconnect`. (Bug #13775)
- Driver incorrectly closes streams passed as arguments to `PreparedStatements`. Reverts to legacy behavior by setting the JDBC configuration property `autoClosePStmtStreams` to `true` (also included in the 3-0-Compat configuration "bundle"). (Bug #15024)
- `maxQuerySizeToLog` is not respected. Added logging of bound values for `execute()` phase of server-side prepared statements when `profileSQL=true` as well. (Bug #13048)
- Usage advisor complains about unreferenced columns, even though they've been referenced. (Bug #15065)
- Don't increase timeout for failover/reconnect. (Bug #6577)
- Process escape tokens in `Connection.prepareStatement(...)`. (Bug #15141) You can disable this behavior by setting the JDBC URL configuration property `processEscapeCodesForPrepStmts` to `false`.
- Reconnect during middle of `executeBatch()` should not occur if `autoReconnect` is enabled. (Bug #13255)

C.5.3.5. Changes in MySQL Connector/J 3.1.11-stable (07 October 2005)

- Spurious `!` on console when character encoding is `utf8`. (Bug #11629)
- Fixed statements generated for testcases missing `;` for "plain" statements.
- Incorrect generation of testcase scripts for server-side prepared statements. (Bug #11663)
- Fixed regression caused by fix for Bug #11552 that caused driver to return incorrect values for unsigned integers when those integers where within the range of the positive signed type.
- Moved source code to Subversion repository.
- Escape tokenizer doesn't respect stacked single quotes for escapes. (Bug #11797)
- `GEOMETRY` type not recognized when using server-side prepared statements.
- `ReplicationConnection` won't switch to slave, throws "Catalog can't be null" exception. (Bug #11879)
- Properties shared between master and slave with replication connection. (Bug #12218)
- `Statement.getWarnings()` fails with NPE if statement has been closed. (Bug #10630)
- Only get `char[]` from SQL in `PreparedStatement.ParseInfo()` when needed.
- Geometry types not handled with server-side prepared statements. (Bug #12104)
- `StringUtils.getBytes()` doesn't work when using multi-byte character encodings and a length in *characters* is specified. (Bug #11614)

- `Pstmt.setObject(...., Types.BOOLEAN)` throws exception. (Bug #11798)
- `maxPerformance.properties` mis-spells “elideSetAutoCommits”. (Bug #11976)
- `DBMD.storesLower/Mixed/UpperIdentifiers()` reports incorrect values for servers deployed on Windows. (Bug #11575)
- `ResultSet.moveToCurrentRow()` fails to work when preceded by a call to `ResultSet.moveToInsertRow()`. (Bug #11190)
- `VARBINARY` data corrupted when using server-side prepared statements and `.setBytes()`. (Bug #11115)
- `explainSlowQueries` hangs with server-side prepared statements. (Bug #12229)
- Escape processor didn't honor strings demarcated with double quotes. (Bug #11498)
- Lifted restriction of changing streaming parameters with server-side prepared statements. As long as `all` streaming parameters were set before execution, `.clearParameters()` does not have to be called. (due to limitation of client/server protocol, prepared statements can not reset *individual* stream data on the server side).
- Reworked `Field` class, `*Buffer`, and `MysqlIO` to be aware of field lengths > `Integer.MAX_VALUE`.
- Updated `DBMD.supportsCorrelatedQueries()` to return `true` for versions > 4.1, `supportsGroupByUnrelated()` to return `true` and `getResultSetHoldability()` to return `HOLD_CURSORS_OVER_COMMIT`.
- Handling of catalog argument in `DatabaseMetaData.getIndexInfo()`, which also means changes to the following methods in `DatabaseMetaData`: (Bug #12541)
  - `getBestRowIdentifier()`
  - `getColumns()`
  - `getCrossReference()`
  - `getExportedKeys()`
  - `getImportedKeys()`
  - `getIndexInfo()`
  - `getPrimaryKeys()`
  - `getProcedures()` (and thus indirectly `getProcedureColumns()`)
  - `getTables()`

  The `catalog` argument in all of these methods now behaves in the following way:

  - Specifying `NULL` means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set `nullCatalogMeansCurrent=true` in your JDBC URL properties.
  - Specifying `""` means “current” catalog, even though this isn't quite JDBC spec compliant, it's there for legacy users.
  - Specifying a catalog works as stated in the API docs.

    - Made `Connection.clientPrepare()` available from “wrapped” connections in the `jdbc2.optional` package (connections built by `ConnectionPoolDataSource` instances).

- Added `Connection.isMasterConnection()` for clients to be able to determine if a multi-host master/slave connection is connected to the first host in the list.
- Tokenizer for `=` in URL properties was causing `sessionVariables=...` to be parameterized incorrectly. (Bug #12753)
- Foreign key information that is quoted is parsed incorrectly when `DatabaseMetaData` methods use that information. (Bug #11781)
- The `sendBlobChunkSize` property is now clamped to `max_allowed_packet` with consideration of stream buffer size and packet headers to avoid `PacketTooBigExceptions` when `max_allowed_packet` is similar in size to the default `sendBlobChunkSize` which is 1M.
- `CallableStatement.clearParameters()` now clears resources associated with `INOUT`/`OUTPUT` parameters as well as `INPUT` parameters.
- `Connection.prepareCall()` is database name case-sensitive (on Windows systems). (Bug #12417)
- `cp1251` incorrectly mapped to `win1251` for servers newer than 4.0.x. (Bug #12752)
- `java.sql.Types.OTHER` returned for `BINARY` and `VARBINARY` columns when using `DatabaseMetaData.getColumns()`. (Bug #12970)
- `ServerPreparedStatement.getBinding()` now checks if the statement is closed before attempting to reference the list of parameter bindings, to avoid throwing a `NullPointerException`.
- `ResultSetMetaData` from `Statement.getGeneratedKeys()` caused a `NullPointerException` to be thrown whenever a method that required a connection reference was called. (Bug #13277)
- Backport of `Field` class, `ResultSetMetaData.getColumnClassName()`, and `ResultSet.getObject(int)` changes from 5.0 branch to fix behavior surrounding `VARCHAR BINARY`/`VARBINARY` and related types.
- Fixed `NullPointerException` when converting `catalog` parameter in many `DatabaseMetaDataMethods` to `byte[]`s (for the result set) when the parameter is `null`. (`null` isn't technically allowed by the JDBC specification, but we've historically allowed it).
- Backport of `VAR[BINARY|CHAR] [BINARY]` types detection from 5.0 branch.
- Read response in `MysqlIO.sendFileToServer()`, even if the local file can't be opened, otherwise next query issued will fail, because it's reading the response to the empty `LOAD DATA INFILE` packet sent to the server.
- Workaround for Bug #13374: `ResultSet.getStatement()` on closed result set returns `NULL` (as per JDBC 4.0 spec, but not backward-compatible). Set the connection property `retainStatementAfterResultSetClose` to `true` to be able to retrieve a `ResultSet`'s statement after the `ResultSet` has been closed via `.getStatement()` (the default is `false`, to be JDBC-compliant and to reduce the chance that code using JDBC leaks `Statement` instances).
- URL configuration parameters don't allow ‘`&`’ or ‘`=`’ in their values. The JDBC driver now parses configuration parameters as if they are encoded using the application/x-www-form-urlencoded format as specified by `java.net.URLDecoder` (http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html). (Bug #13453)

If the '%' character is present in a configuration property, it must now be represented as %25, which is the encoded form of '%' when using application/x-www-form-urlencoded encoding.

- The configuration property sessionVariables now allows you to specify variables that start with the '@' sign.
- When gatherPerfMetrics is enabled for servers older than 4.1.0, a NullPointerException is thrown from the constructor of ResultSet if the query doesn't use any tables. (Bug #13043)

C.5.3.6. Changes in MySQL Connector/J 3.1.10-stable (23 June 2005)

- Fixed connecting without a database specified raised an exception in MysqlIO.changeDatabaseTo().
- Initial implemention of ParameterMetadata for PreparedStatement.getParameterMetadata(). Only works fully for CallableStatements, as current server-side prepared statements return every parameter as a VARCHAR type.

C.5.3.7. Changes in MySQL Connector/J 3.1.9-stable (22 June 2005)

- Overhaul of character set configuration, everything now lives in a properties file.
- Driver now correctly uses CP932 if available on the server for Windows-31J, CP932 and MS932 java encoding names, otherwise it resorts to SJIS, which is only a close approximation. Currently only MySQL-5.0.3 and newer (and MySQL-4.1.12 or .13, depending on when the character set gets backported) can reliably support any variant of CP932.
- com.mysql.jdbc.PreparedStatement.ParseInfo does unnecessary call to toCharArray(). (Bug #9064)
- Memory leak in ServerPreparedStatement if serverPrepare() fails. (Bug #10144)
- Actually write manifest file to correct place so it ends up in the binary jar file.
- Added createDatabaseIfNotExist property (default is false), which will cause the driver to ask the server to create the database specified in the URL if it doesn't exist. You must have the appropriate privileges for database creation for this to work.
- Unsigned SMALLINT treated as signed for ResultSet.getInt(), fixed all cases for UNSIGNED integer values and server-side prepared statements, as well as ResultSet.getObject() for UNSIGNED TINYINT. (Bug #10156)
- Double quotes not recognized when parsing client-side prepared statements. (Bug #10155)
- Made enableStreamingResults() visible on com.mysql.jdbc.jdbc2.optional.StatementWrapper.
- Made ServerPreparedStatement.asSql() work correctly so auto-explain functionality would work with server-side prepared statements.
- Made JDBC2-compliant wrappers public in order to allow access to vendor extensions.
- Cleaned up logging of profiler events, moved code to dump a profiler event as a string to com.mysql.jdbc.log.LogUtils so that third parties can use it.
- DatabaseMetaData.supportsMultipleOpenResults() now returns true. The driver has supported this for some time, DBMD just missed that fact.

- Driver doesn't support `{?=CALL(...)}` for calling stored functions. This involved adding support for function retrieval to `DatabaseMetaData.getProcedures()` and `getProcedureColumns()` as well. (Bug #10310)
- `SQLException` thrown when retrieving `YEAR(2)` with `ResultSet.getString()`. The driver will now always treat `YEAR` types as `java.sql.Dates` and return the correct values for `getString()`. Alternatively, the `yearIsDateType` connection property can be set to `false` and the values will be treated as `SHORT`s. (Bug #10485)
- The datatype returned for `TINYINT(1)` columns when `tinyInt1isBit=true` (the default) can be switched between `Types.BOOLEAN` and `Types.BIT` using the new configuration property `transformedBitIsBoolean`, which defaults to `false`. If set to `false` (the default), `DatabaseMetaData.getColumns()` and `ResultSetMetaData.getColumnType()` will return `Types.BOOLEAN` for `TINYINT(1)` columns. If `true`, `Types.BOOLEAN` will be returned instead. Regardless of this configuration property, if `tinyInt1isBit` is enabled, columns with the type `TINYINT(1)` will be returned as `java.lang.Boolean` instances from `ResultSet.getObject(...)`, and `ResultSetMetaData.getColumnClassName()` will return `java.lang.Boolean`.
- `SQLException` is thrown when using property `characterSetResults` with `cp932` or `eucjpms`. (Bug #10496)
- Reorganized directory layout. Sources now are in `src` folder. Don't pollute parent directory when building, now output goes to `./build`, distribution goes to `./dist`.
- Added support/bug hunting feature that generates `.sql` test scripts to `STDERR` when `autoGenerateTestcaseScript` is set to `true`.
- 0-length streams not sent to server when using server-side prepared statements. (Bug #10850)
- Setting `cachePrepStmts=true` now causes the `Connection` to also cache the check the driver performs to determine if a prepared statement can be server-side or not, as well as caches server-side prepared statements for the lifetime of a connection. As before, the `prepStmtCacheSize` parameter controls the size of these caches.
- Try to handle `OutOfMemoryErrors` more gracefully. Although not much can be done, they will in most cases close the connection they happened on so that further operations don't run into a connection in some unknown state. When an OOM has happened, any further operations on the connection will fail with a “Connection closed” exception that will also list the OOM exception as the reason for the implicit connection close event.
- Don't send `COM_RESET_STMT` for each execution of a server-side prepared statement if it isn't required.
- Driver detects if you're running MySQL-5.0.7 or later, and does not scan for `LIMIT ?[,?]` in statements being prepared, as the server supports those types of queries now.
- `VARBINARY` data corrupted when using server-side prepared statements and `ResultSet.getBytes()`. (Bug #11115)
- `Connection.setCatalog()` is now aware of the `useLocalSessionState` configuration property, which when set to `true` will prevent the driver from sending `USE ...` to the server if the requested catalog is the same as the current catalog.
- Added the following configuration bundles, use one or many via the `useConfigs` configuration property:
  - `maxPerformance` — maximum performance without being reckless
  - `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can

  - `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality

- Added `maintainTimeStats` configuration property (defaults to `true`), which tells the driver whether or not to keep track of the last query time and the last successful packet sent to the server's time. If set to `false`, removes two syscalls per query.
- `autoReconnect` ping causes exception on connection startup. (Bug #11259)
- Connector/J dumping query into `SQLException` twice. (Bug #11360)
- Fixed `PreparedStatement.setClob()` not accepting `null` as a parameter.
- Production package doesn't include JBoss integration classes. (Bug #11411)
- Removed nonsensical “costly type conversion” warnings when using usage advisor.

C.5.3.8. Changes in MySQL Connector/J 3.1.8-stable (14 April 2005)

- Fixed `DatabaseMetaData.getTables()` returning views when they were not asked for as one of the requested table types.
- Added support for new precision-math `DECIMAL` type in MySQL 5.0.3 and up.
- Fixed `ResultSet.getTime()` on a `NULL` value for server-side prepared statements throws NPE.
- Made `Connection.ping()` a public method.
- `DATE_FORMAT()` queries returned as `BLOB`s from `getObject()`. (Bug #8868)
- `ServerPreparedStatements` now correctly “stream” `BLOB`/`CLOB` data to the server. You can configure the threshold chunk size using the JDBC URL property `blobSendChunkSize` (the default is 1MB).
- `BlobFromLocator` now uses correct identifier quoting when generating prepared statements.
- Server-side session variables can be preset at connection time by passing them as a comma-delimited list for the connection property `sessionVariables`.
- Fixed regression in `ping()` for users using `autoReconnect=true`.
- `PreparedStatement.addBatch()` doesn't work with server-side prepared statements and streaming `BINARY` data. (Bug #9040)
- `DBMD.supportsMixedCase*Identifiers()` returns wrong value on servers running on case-sensitive filesystems. (Bug #8800)
- Cannot use `UTF-8` for characterSetResults configuration property. (Bug #9206)
- A continuation of Bug #8868, where functions used in queries that should return non-string types when resolved by temporary tables suddenly become opaque binary strings (work-around for server limitation). Also fixed fields with type of `CHAR(n) CHARACTER SET BINARY` to return correct/matching classes for `RSMD.getColumnClassName()` and `ResultSet.getObject()`. (Bug #9236)
- `DBMD.supportsResultSetConcurrency()` not returning `true` for forward-only/read-only result sets (we obviously support this). (Bug #8792)
- `DATA_TYPE` column from `DBMD.getBestRowIdentifier()` causes `ArrayIndexOutOfBoundsException` when accessed (and in fact, didn't return any value). (Bug #8803)

- Check for empty strings (`''`) when converting `CHAR`/`VARCHAR` column data to numbers, throw exception if `emptyStringsConvertToZero` configuration property is set to `false` (for backward-compatibility with 3.0, it is now set to `true` by default, but will most likely default to `false` in 3.2).
- `PreparedStatement.getMetaData()` inserts blank row in database under certain conditions when not using server-side prepared statements. (Bug #9320)
- `Connection.canHandleAsPreparedStatement()` now makes “best effort” to distinguish `LIMIT` clauses with placeholders in them from ones without in order to have fewer false positives when generating work-arounds for statements the server cannot currently handle as server-side prepared statements.
- Fixed `build.xml` to not compile `log4j` logging if `log4j` not available.
- Added support for the c3p0 connection pool's (http://c3p0.sf.net/) validation/connection checker interface which uses the lightweight `COM_PING` call to the server if available. To use it, configure your c3p0 connection pool's `connectionTesterClassName` property to use `com.mysql.jdbc.integration.c3p0.MysqlConnectionTester`.
- Better detection of `LIMIT` inside/outside of quoted strings so that the driver can more correctly determine whether a prepared statement can be prepared on the server or not.
- Stored procedures with same name in different databases confuse the driver when it tries to determine parameter counts/types. (Bug #9319)
- Added finalizers to `ResultSet` and `Statement` implementations to be JDBC spec-compliant, which requires that if not explicitly closed, these resources should be closed upon garbage collection.
- Stored procedures with `DECIMAL` parameters with storage specifications that contained ‘`,`’ in them would fail. (Bug #9682)
- `PreparedStatement.setObject(int, Object, int type, int scale)` now uses scale value for `BigDecimal` instances.
- `Statement.getMoreResults()` could throw NPE when existing result set was `.close()`d. (Bug #9704)
- The performance metrics feature now gathers information about number of tables referenced in a SELECT.
- The logging system is now automatically configured. If the value has been set by the user, via the URL property `logger` or the system property `com.mysql.jdbc.logger`, then use that, otherwise, autodetect it using the following steps:

  1. Log4j, if it's available,
  2. Then JDK1.4 logging,
  3. Then fallback to our `STDERR` logging.

- `DBMD.getTables()` shouldn't return tables if views are asked for, even if the database version doesn't support views. (Bug #9778)
- Fixed driver not returning `true` for `-1` when `ResultSet.getBoolean()` was called on result sets returned from server-side prepared statements.
- Added a `Manifest.MF` file with implementation information to the `.jar` file.

- More tests in `Field.isOpaqueBinary()` to distinguish opaque binary (that is, fields with type `CHAR(n)` and `CHARACTER SET BINARY`) from output of various scalar and aggregate functions that return strings.
- Should accept `null` for catalog (meaning use current) in DBMD methods, even though it's not JDBC-compliant for legacy's sake. Disable by setting connection property `nullCatalogMeansCurrent` to `false` (which will be the default value in C/J 3.2.x). (Bug #9917)
- Should accept `null` for name patterns in DBMD (meaning '`%`'), even though it isn't JDBC compliant, for legacy's sake. Disable by setting connection property `nullNamePatternMatchesAll` to `false` (which will be the default value in C/J 3.2.x). (Bug #9769)

C.5.3.9. Changes in MySQL Connector/J 3.1.7-stable (18 February 2005)

- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug #7686)
- Timestamps converted incorrectly to strings with server-side prepared statements and updatable result sets. (Bug #7715)
- Detect new `sql_mode` variable in string form (it used to be integer) and adjust quoting method for strings appropriately.
- Added `holdResultsOpenOverStatementClose` property (default is `false`), that keeps result sets open over statement.close() or new execution on same statement (suggested by Kevin Burton).
- Infinite recursion when “falling back” to master in failover configuration. (Bug #7952)
- Disable multi-statements (if enabled) for MySQL-4.1 versions prior to version 4.1.10 if the query cache is enabled, as the server returns wrong results in this configuration.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Removed `dontUnpackBinaryResults` functionality, the driver now always stores results from server-side prepared statements as is from the server and unpacks them on demand.
- Emulated locators corrupt binary data when using server-side prepared statements. (Bug #8096)
- Fixed synchronization issue with `ServerPreparedStatement.serverPrepare()` that could cause deadlocks/crashes if connection was shared between threads.
- By default, the driver now scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this by passing `emulateUnsupportedPstmts=false` in your JDBC URL. (Bug #4718)
- Remove `_binary` introducer from parameters used as in/out parameters in `CallableStatement`.
- Always return `byte[]`s for output parameters registered as `*BINARY`.
- Send correct value for “boolean” `true` to server for `PreparedStatement.setObject(n, "true", Types.BIT)`.
- Fixed bug with Connection not caching statements from `prepareStatement()` when the statement wasn't a server-side prepared statement.

- Choose correct “direction” to apply time adjustments when both client and server are in GMT time zone when using `ResultSet.get(..., cal)` and `PreparedStatement.set(...., cal)`.
- Added `dontTrackOpenResources` option (default is `false`, to be JDBC compliant), which helps with memory use for non-well-behaved apps (that is, applications that don't close `Statement` objects when they should).
- `ResultSet.getString()` doesn't maintain format stored on server, bug fix only enabled when `noDatetimeStringSync` property is set to `true` (the default is `false`). (Bug #8428)
- Fixed NPE in `ResultSet.realClose()` when using usage advisor and result set was already closed.
- `PreparedStatements` not creating streaming result sets. (Bug #8487)
- Don't pass `NULL` to `String.valueOf()` in `ResultSet.getNativeConvertToString()`, as it stringifies it (that is, returns `null`), which is not correct for the method in question.
- `ResultSet.getBigDecimal()` throws exception when rounding would need to occur to set scale. The driver now chooses a rounding mode of “half up” if non-rounding `BigDecimal.setScale()` fails. (Bug #8424)
- Added `useLocalSessionState` configuration property, when set to `true` the JDBC driver trusts that the application is well-behaved and only sets autocommit and transaction isolation levels using the methods provided on `java.sql.Connection`, and therefore can manipulate these values in many cases without incurring round-trips to the database server.
- Added `enableStreamingResults()` to `Statement` for connection pool implementations that check `Statement.setFetchSize()` for specification-compliant values. Call `Statement.setFetchSize(>=0)` to disable the streaming results for that statement.
- Added support for `BIT` type in MySQL-5.0.3. The driver will treat `BIT(1-8)` as the JDBC standard `BIT` type (which maps to `java.lang.Boolean`), as the server does not currently send enough information to determine the size of a bitfield when < 9 bits are declared. `BIT(>9)` will be treated as `VARBINARY`, and will return `byte[]` when `getObject()` is called.

C.5.3.10. Changes in MySQL Connector/J 3.1.6-stable (23 December 2004)

- Fixed hang on `SocketInputStream.read()` with `Statement.setMaxRows()` and multiple result sets when driver has to truncate result set directly, rather than tacking a `LIMIT` *n* on the end of it.
- `DBMD.getProcedures()` doesn't respect catalog parameter. (Bug #7026)

C.5.3.11. Changes in MySQL Connector/J 3.1.5-gamma (02 December 2004)

- Fix comparisons made between string constants and dynamic strings that are converted with either `toUpperCase()` or `toLowerCase()` to use `Locale.ENGLISH`, as some locales “override” case rules for English. Also use `StringUtils.indexOfIgnoreCase()` instead of `.toUpperCase().indexOf()`, avoids creating a very short-lived transient `String` instance.
- Server-side prepared statements did not honor `zeroDateTimeBehavior` property, and would cause class-cast exceptions when using `ResultSet.getObject()`, as the all-zero string was always returned. (Bug #5235)
- Fixed batched updates with server prepared statements weren't looking if the types had changed for a given batched set of parameters compared to the previous set, causing the server to return the error “Wrong arguments to mysql_stmt_execute()”.
- Handle case when string representation of timestamp contains trailing ‘`.`’ with no numbers following it.

- Inefficient detection of pre-existing string instances in `ResultSet.getNativeString()`. (Bug #5706)
- Don't throw exceptions for `Connection.releaseSavepoint()`.
- Use a per-session `Calendar` instance by default when decoding dates from `ServerPreparedStatements` (set to old, less performant behavior by setting property `dynamicCalendars=true`).
- Added experimental configuration property `dontUnpackBinaryResults`, which delays unpacking binary result set values until they're asked for, and only creates object instances for non-numerical values (it is set to `false` by default). For some usecase/jvm combinations, this is friendlier on the garbage collector.
- `UNSIGNED BIGINT` unpacked incorrectly from server-side prepared statement result sets. (Bug #5729)
- `ServerSidePreparedStatement` allocating short-lived objects unnecessarily. (Bug #6225)
- Removed unwanted new `Throwable()` in `ResultSet` constructor due to bad merge (caused a new object instance that was never used for every result set created). Found while profiling for Bug #6359.
- Fixed too-early creation of `StringBuffer` in `EscapeProcessor.escapeSQL()`, also return `String` when escaping not needed (to avoid unnecessary object allocations). Found while profiling for Bug #6359.
- Use null-safe-equals for key comparisons in updatable result sets.
- `SUM()` on `DECIMAL` with server-side prepared statement ignores scale if zero-padding is needed (this ends up being due to conversion to `DOUBLE` by server, which when converted to a string to parse into `BigDecimal`, loses all “padding” zeros). (Bug #6537)
- Use `DatabaseMetaData.getIdentifierQuoteString()` when building DBMD queries.
- Use 1MB packet for sending file for `LOAD DATA LOCAL INFILE` if that is < `max_allowed_packet` on server.
- `ResultSetMetaData.getColumnDisplaySize()` returns incorrect values for multi-byte charsets. (Bug #6399)
- Make auto-deserialization of `java.lang.Objects` stored in `BLOB` columns configurable via `autoDeserialize` property (defaults to `false`).
- Re-work `Field.isOpaqueBinary()` to detect `CHAR(n) CHARACTER SET BINARY` to support fixed-length binary fields for `ResultSet.getObject()`.
- Use our own implementation of buffered input streams to get around blocking behavior of `java.io.BufferedInputStream`. Disable this with `useReadAheadInput=false`.
- Failing to connect to the server when one of the addresses for the given host name is IPV6 (which the server does not yet bind on). The driver now loops through *all* IP addresses for a given host, and stops on the first one that `accepts()` a `socket.connect()`. (Bug #6348)

C.5.3.12. Changes in MySQL Connector/J 3.1.4-beta (04 September 2004)

- Connector/J 3.1.3 beta does not handle integers correctly (caused by changes to support unsigned reads in `Buffer.readInt()` -> `Buffer.readShort()`). (Bug #4510)
- Added support in `DatabaseMetaData.getTables()` and `getTableTypes()` for views, which are now available in MySQL server 5.0.x.

- `ServerPreparedStatement.execute*()` sometimes threw `ArrayIndexOutOfBoundsException` when unpacking field metadata. (Bug #4642)
- Optimized integer number parsing, enable “old” slower integer parsing using JDK classes via `useFastIntParsing=false` property.
- Added `useOnlyServerErrorMessages` property, which causes message text in exceptions generated by the server to only contain the text sent by the server (as opposed to the SQLState's “standard” description, followed by the server's error message). This property is set to `true` by default.
- `ResultSet.wasNull()` does not work for primatives if a previous `null` was returned. (Bug #4689)
- Track packet sequence numbers if `enablePacketDebug=true`, and throw an exception if packets received out-of-order.
- `ResultSet.getObject()` returns wrong type for strings when using prepared statements. (Bug #4482)
- Calling `MysqlPooledConnection.close()` twice (even though an application error), caused NPE. Fixed.
- `ServerPreparedStatements` dealing with return of `DECIMAL` type don't work. (Bug #5012)
- `ResultSet.getObject()` doesn't return type `Boolean` for pseudo-bit types from prepared statements on 4.1.x (shortcut for avoiding extra type conversion when using binary-encoded result sets obscured test in `getObject()` for “pseudo” bit type). (Bug #5032)
- You can now use URLs in `LOAD DATA LOCAL INFILE` statements, and the driver will use Java's built-in handlers for retreiving the data and sending it to the server. This feature is not enabled by default, you must set the `allowUrlInLocalInfile` connection property to `true`.
- The driver is more strict about truncation of numerics on `ResultSet.get*()`, and will throw an `SQLException` when truncation is detected. You can disable this by setting `jdbcCompliantTruncation` to `false` (it is enabled by default, as this functionality is required for JDBC compliance).
- Added three ways to deal with all-zero datetimes when reading them from a `ResultSet`: `exception` (the default), which throws an `SQLException` with an SQLState of `S1009`; `convertToNull`, which returns `NULL` instead of the date; and `round`, which rounds the date to the nearest closest value which is `'0001-01-01'`.
- Fixed `ServerPreparedStatement` to read prepared statement metadata off the wire, even though it's currently a placeholder instead of using `MysqlIO.clearInputStream()` which didn't work at various times because data wasn't available to read from the server yet. This fixes sporadic errors users were having with `ServerPreparedStatements` throwing `ArrayIndexOutOfBoundExceptions`.
- Use `com.mysql.jdbc.Message`'s classloader when loading resource bundle, should fix sporadic issues when the caller's classloader can't locate the resource bundle.

C.5.3.13. Changes in MySQL Connector/J 3.1.3-beta (07 July 2004)

- Mangle output parameter names for `CallableStatements` so they will not clash with user variable names.
- Added support for `INOUT` parameters in `CallableStatements`.
- Null bitmask sent for server-side prepared statements was incorrect. (Bug #4119)

- Use SQL Standard SQL states by default, unless `useSqlStateCodes` property is set to `false`.
- Added packet debuging code (see the `enablePacketDebug` property documentation).
- Added constants for MySQL error numbers (publicly accessible, see `com.mysql.jdbc.MysqlErrorNumbers`), and the ability to generate the mappings of vendor error codes to SQLStates that the driver uses (for documentation purposes).
- Externalized more messages (on-going effort).
- Error in retrieval of `mediumint` column with prepared statements and binary protocol. (Bug #4311)
- Support new time zone variables in MySQL-4.1.3 when `useTimezone=true`.
- Support for unsigned numerics as return types from prepared statements. This also causes a change in `ResultSet.getObject()` for the `bigint unsigned` type, which used to return `BigDecimal` instances, it now returns instances of `java.lang.BigInteger`.

C.5.3.14. Changes in MySQL Connector/J 3.1.2-alpha (09 June 2004)

- Fixed stored procedure parameter parsing info when size was specified for a parameter (for example, `char()`, `varchar()`).
- Enabled callable statement caching via `cacheCallableStmts` property.
- Fixed case when no output parameters specified for a stored procedure caused a bogus query to be issued to retrieve out parameters, leading to a syntax error from the server.
- Fixed case when no parameters could cause a `NullPointerException` in `CallableStatement.setOutputParameters()`.
- Removed wrapping of exceptions in `MysqlIO.changeUser()`.
- Fixed sending of split packets for large queries, enabled nio ability to send large packets as well.
- Added `.toString()` functionality to `ServerPreparedStatement`, which should help if you're trying to debug a query that is a prepared statement (it shows SQL as the server would process).
- Added `gatherPerformanceMetrics` property, along with properties to control when/where this info gets logged (see docs for more info).
- `ServerPreparedStatements` weren't actually de-allocating server-side resources when `.close()` was called.
- Added `logSlowQueries` property, along with `slowQueriesThresholdMillis` property to control when a query should be considered “slow.”
- Correctly map output parameters to position given in `prepareCall()` versus. order implied during `registerOutParameter()`. (Bug #3146)
- Correctly detect initial character set for servers >= 4.1.0.
- Cleaned up detection of server properties.
- Support placeholder for parameter metadata for server >= 4.1.2.
- `getProcedures()` does not return any procedures in result set. (Bug #3539)
- `getProcedureColumns()` doesn't work with wildcards for procedure name. (Bug #3540)

- `DBMD.getSQLStateType()` returns incorrect value. (Bug #3520)
- Added `connectionCollation` property to cause driver to issue `set collation_connection=...` query on connection init if default collation for given charset is not appropriate.
- Fixed `DatabaseMetaData.getProcedures()` when run on MySQL-5.0.0 (output of `SHOW PROCEDURE STATUS` changed between 5.0.0 and 5.0.1.
- `getWarnings()` returns `SQLWarning` instead of `DataTruncation`. (Bug #3804)
- Don't enable server-side prepared statements for server version 5.0.0 or 5.0.1, as they aren't compatible with the '4.1.2+' style that the driver uses (the driver expects information to come back that isn't there, so it hangs).

C.5.3.15. Changes in MySQL Connector/J 3.1.1-alpha (14 February 2004)

- Fixed bug with `UpdatableResultSets` not using client-side prepared statements.
- Fixed character encoding issues when converting bytes to ASCII when MySQL doesn't provide the character set, and the JVM is set to a multi-byte encoding (usually affecting retrieval of numeric values).
- Unpack “unknown” data types from server prepared statements as `Strings`.
- Implemented long data (Blobs, Clobs, InputStreams, Readers) for server prepared statements.
- Implemented `Statement.getWarnings()` for MySQL-4.1 and newer (using `SHOW WARNINGS`).
- Default result set type changed to `TYPE_FORWARD_ONLY` (JDBC compliance).
- Centralized setting of result set type and concurrency.
- Refactored how connection properties are set and exposed as `DriverPropertyInfo` as well as `Connection` and `DataSource` properties.
- Support for NIO. Use `useNIO=true` on platforms that support NIO.
- Support for transaction savepoints (MySQL >= 4.0.14 or 4.1.1).
- Support for `mysql_change_user()`. See the `changeUser()` method in `com.mysql.jdbc.Connection`.
- Reduced number of methods called in average query to be more efficient.
- Prepared `Statements` will be re-prepared on auto-reconnect. Any errors encountered are postponed until first attempt to re-execute the re-prepared statement.
- Ensure that warnings are cleared before executing queries on prepared statements, as-per JDBC spec (now that we support warnings).
- Support “old” `profileSql` capitalization in `ConnectionProperties`. This property is deprecated, you should use `profileSQL` if possible.
- Optimized `Buffer.readLenByteArray()` to return shared empty byte array when length is 0.
- Allow contents of `PreparedStatement.setBlob()` to be retained between calls to `.execute*()`.
- Deal with 0-length tokens in `EscapeProcessor` (caused by callable statement escape syntax).
- Check for closed connection on delete/update/insert row operations in `UpdatableResultSet`.

- Fix support for table aliases when checking for all primary keys in `UpdatableResultSet`.
- Removed `useFastDates` connection property.
- Correctly initialize datasource properties from JNDI Refs, including explicitly specified URLs.
- `DatabaseMetaData` now reports `supportsStoredProcedures()` for MySQL versions >= 5.0.0
- Fixed stack overflow in `Connection.prepareCall()` (bad merge).
- Fixed `IllegalAccessError` to `Calendar.getTimeInMillis()` in `DateTimeValue` (for JDK < 1.4).
- `DatabaseMetaData.getColumns()` is not returning correct column ordinal info for non-`'%'` column name patterns. (Bug #1673)
- Merged fix of datatype mapping from MySQL type `FLOAT` to `java.sql.Types.REAL` from 3.0 branch.
- Detect collation of column for `RSMD.isCaseSensitive()`.
- Fixed sending of queries larger than 16M.
- Added named and indexed input/output parameter support to `CallableStatement`. MySQL-5.0.x or newer.
- Fixed `NullPointerException` in `ServerPreparedStatement.setTimestamp()`, as well as year and month descrepencies in `ServerPreparedStatement.setTimestamp()`, `setDate()`.
- Added ability to have multiple database/JVM targets for compliance and regression/unit tests in `build.xml`.
- Fixed NPE and year/month bad conversions when accessing some datetime functionality in `ServerPreparedStatements` and their resultant result sets.
- Display where/why a connection was implicitly closed (to aid debugging).
- `CommunicationsException` implemented, that tries to determine why communications was lost with a server, and displays possible reasons when `.getMessage()` is called.
- `NULL` values for numeric types in binary encoded result sets causing `NullPointerExceptions`. (Bug #2359)
- Implemented `Connection.prepareCall()`, and `DatabaseMetaData`. `getProcedures()` and `getProcedureColumns()`.
- Reset `long binary` parameters in `ServerPreparedStatement` when `clearParameters()` is called, by sending `COM_RESET_STMT` to the server.
- Merged prepared statement caching, and `.getMetaData()` support from 3.0 branch.
- Fixed off-by-1900 error in some cases for years in `TimeUtil.fastDate`/`TimeCreate()` when unpacking results from server-side prepared statements.
- Fixed charset conversion issue in `getTables()`. (Bug #2502)
- Implemented multiple result sets returned from a statement or stored procedure.
- Server-side prepared statements were not returning datatype `YEAR` correctly. (Bug #2606)
- Enabled streaming of result sets from server-side prepared statements.

- Class-cast exception when using scrolling result sets and server-side prepared statements. (Bug #2623)
- Merged unbuffered input code from 3.0.
- Fixed `ConnectionProperties` that weren't properly exposed via accessors, cleaned up `ConnectionProperties` code.
- `NULL` fields were not being encoded correctly in all cases in server-side prepared statements. (Bug #2671)
- Fixed rare buffer underflow when writing numbers into buffers for sending prepared statement execution requests.
- Use DocBook version of docs for shipped versions of drivers.

C.5.3.16. Changes in MySQL Connector/J 3.1.0-alpha (18 February 2003)

- Added `requireSSL` property.
- Added `useServerPrepStmts` property (default `false`). The driver will use server-side prepared statements when the server version supports them (4.1 and newer) when this property is set to `true`. It is currently set to `false` by default until all bind/fetch functionality has been implemented. Currently only DML prepared statements are implemented for 4.1 server-side prepared statements.
- Track open `Statements`, close all when `Connection.close()` is called (JDBC compliance).

C.5.4. Changes in MySQL Connector/J 3.0.x

C.5.4.1. Changes in MySQL Connector/J 3.0.17-ga (23 June 2005)

- `Timestamp`/`Time` conversion goes in the wrong “direction” when `useTimeZone=true` and server time zone differs from client time zone. (Bug #5874)
- `DatabaseMetaData.getIndexInfo()` ignored `unique` parameter. (Bug #7081)
- Support new protocol type `MYSQL_TYPE_VARCHAR`.
- Added `useOldUTF8Behavior`' configuration property, which causes JDBC driver to act like it did with MySQL-4.0.x and earlier when the character encoding is `utf-8` when connected to MySQL-4.1 or newer.
- Statements created from a pooled connection were returning physical connection instead of logical connection when `getConnection()` was called. (Bug #7316)
- `PreparedStatements` don't encode Big5 (and other multi-byte) character sets correctly in static SQL strings. (Bug #7033)
- Connections starting up failed-over (due to down master) never retry master. (Bug #6966)
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug #7061)
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug #7686)
- Backported SQLState codes mapping from Connector/J 3.1, enable with `useSqlStateCodes=true` as a connection property, it defaults to `false` in this release, so that we don't break legacy applications (it defaults to `true` starting with Connector/J 3.1).

- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug #7601)
- Escape sequence {fn convert(..., type)} now supports ODBC-style types that are prepended by `SQL_`.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Handle streaming result sets with more than 2 billion rows properly by fixing wraparound of row number counter.
- `MS932`, `SHIFT_JIS`, and `Windows_31J` not recognized as aliases for `sjis`. (Bug #7607)
- Adding `CP943` to aliases for `sjis`. (Bug #6549, fixed while fixing Bug #7607)
- Which requires hex escaping of binary data when using multi-byte charsets with prepared statements. (Bug #8064)
- `NON_UNIQUE` column from `DBMD.getIndexInfo()` returned inverted value. (Bug #8812)
- Workaround for server Bug #9098: Default values of `CURRENT_*` for `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` columns can't be distinguished from `string` values, so `UpdatableResultSet.moveToInsertRow()` generates bad SQL for inserting default values.
- `EUCKR` charset is sent as `SET NAMES euc_kr` which MySQL-4.1 and newer doesn't understand. (Bug #8629)
- `DatabaseMetaData.supportsSelectForUpdate()` returns correct value based on server version.
- Use hex escapes for `PreparedStatement.setBytes()` for double-byte charsets including “aliases” `Windows-31J`, `CP934`, `MS932`.
- Added support for the `EUC_JP_Solaris` character encoding, which maps to a MySQL encoding of `eucjpms` (backported from 3.1 branch). This only works on servers that support `eucjpms`, namely 5.0.3 or later.

C.5.4.2. Changes in MySQL Connector/J 3.0.16-ga (15 November 2004)

- Re-issue character set configuration commands when re-using pooled connections and/or `Connection.changeUser()` when connected to MySQL-4.1 or newer.
- Fixed `ResultSetMetaData.isReadOnly()` to detect non-writable columns when connected to MySQL-4.1 or newer, based on existence of “original” table and column names.
- `ResultSet.updateByte()` when on insert row throws `ArrayOutOfBoundsException`. (Bug #5664)
- Fixed `DatabaseMetaData.getTypes()` returning incorrect (this is, non-negative) scale for the `NUMERIC` type.
- Off-by-one bug in `Buffer.readString(string)`. (Bug #5664)
- Made `TINYINT(1)` -> `BIT`/`Boolean` conversion configurable via `tinyInt1isBit` property (default `true` to be JDBC compliant out of the box).
- Only set `character_set_results` during connection establishment if server version >= 4.1.1.
- Fixed regression where `useUnbufferedInput` was defaulting to `false`.

- `ResultSet.getTimestamp()` on a column with `TIME` in it fails. (Bug #5664)

C.5.4.3. Changes in MySQL Connector/J 3.0.15-production (04 September 2004)

- `StringUtils.escapeEasternUnicodeByteStream` was still broken for GBK. (Bug #4010)
- Failover for `autoReconnect` not using port numbers for any hosts, and not retrying all hosts. (**Warning**: This required a change to the `SocketFactory connect()` method signature, which is now `public Socket connect(String host, int portNumber, Properties props)`; therefore, any third-party socket factories will have to be changed to support this signature. (Bug #4334)
- Logical connections created by `MysqlConnectionPoolDataSource` will now issue a `rollback()` when they are closed and sent back to the pool. If your application server/connection pool already does this for you, you can set the `rollbackOnPooledClose` property to `false` to avoid the overhead of an extra `rollback()`.
- Removed redundant calls to `checkRowPos()` in `ResultSet`.
- `DOUBLE` mapped twice in `DBMD.getTypeInfo()`. (Bug #4742)
- Added FLOSS license exemption.
- Calling `.close()` twice on a `PooledConnection` causes NPE. (Bug #4808)
- `DBMD.getColumns()` returns incorrect JDBC type for unsigned columns. This affects type mappings for all numeric types in the `RSMD.getColumnType()` and `RSMD.getColumnTypeNames()` methods as well, to ensure that “like” types from `DBMD.getColumns()` match up with what `RSMD.getColumnType()` and `getColumnTypeNames()` return. (Bug #4138, Bug #4860)
- “Production” is now “GA” (General Availability) in naming scheme of distributions.
- `RSMD.getPrecision()` returning 0 for non-numeric types (should return max length in chars for non-binary types, max length in bytes for binary types). This fix also fixes mapping of `RSMD.getColumnType()` and `RSMD.getColumnTypeName()` for the `BLOB` types based on the length sent from the server (the server doesn't distinguish between `TINYBLOB`, `BLOB`, `MEDIUMBLOB` or `LONGBLOB` at the network protocol level). (Bug #4880)
- `ResultSet` should release `Field[]` instance in `.close()`. (Bug #5022)
- `ResultSet.getMetaData()` should not return incorrectly initialized metadata if the result set has been closed, but should instead throw an `SQLException`. Also fixed for `getRow()` and `getWarnings()` and traversal methods by calling `checkClosed()` before operating on instance-level fields that are nullified during `.close()`. (Bug #5069)
- Parse new time zone variables from 4.1.x servers.
- Use `_binary` introducer for `PreparedStatement.setBytes()` and `set*Stream()` when connected to MySQL-4.1.x or newer to avoid misinterpretation during character conversion.

C.5.4.4. Changes in MySQL Connector/J 3.0.14-production (28 May 2004)

- Fixed URL parsing error.

C.5.4.5. Changes in MySQL Connector/J 3.0.13-production (27 May 2004)

- Using a `MySQLDatasource` without server name fails. (Bug #3848)
- `No Database Selected` when using `MysqlConnectionPoolDataSource`. (Bug #3920)

- `PreparedStatement.getGeneratedKeys()` method returns only 1 result for batched insertions. (Bug #3873)

C.5.4.6. Changes in MySQL Connector/J 3.0.12-production (18 May 2004)

- Add unsigned attribute to `DatabaseMetaData.getColumns()` output in the `TYPE_NAME` column.
- Added `failOverReadOnly` property, to allow end-user to configure state of connection (read-only/writable) when failed over.
- Backported “change user” and “reset server state” functionality from 3.1 branch, to allow clients of `MysqlConnectionPoolDataSource` to reset server state on `getConnection()` on a pooled connection.
- Don't escape SJIS/GBK/BIG5 when using MySQL-4.1 or newer.
- Allow `url` parameter for `MysqlDataSource` and `MysqlConnectionPool DataSource` so that passing of other properties is possible from inside appservers.
- Map duplicate key and foreign key errors to SQLState of `23000`.
- Backport documentation tooling from 3.1 branch.
- Return creating statement for `ResultSets` created by `getGeneratedKeys()`. (Bug #2957)
- Allow `java.util.Date` to be sent in as parameter to `PreparedStatement.setObject()`, converting it to a `Timestamp` to maintain full precision. (Bug #103).
- Don't truncate `BLOB` or `CLOB` values when using `setBytes()` and/or `setBinary/CharacterStream()`. (Bug #2670).
- Dynamically configure character set mappings for field-level character sets on MySQL-4.1.0 and newer using `SHOW COLLATION` when connecting.
- Map `binary` character set to `US-ASCII` to support `DATETIME` charset recognition for servers >= 4.1.2.
- Use `SET character_set_results` during initialization to allow any charset to be returned to the driver for result sets.
- Use `charsetnr` returned during connect to encode queries before issuing `SET NAMES` on MySQL >= 4.1.0.
- Add helper methods to `ResultSetMetaData` (`getColumnCharacterEncoding()` and `getColumnCharacterSet()`) to allow end-users to see what charset the driver thinks it should be using for the column.
- Only set `character_set_results` for MySQL >= 4.1.0.
- `StringUtils.escapeSJISByteStream()` not covering all eastern double-byte charsets correctly. (Bug #3511)
- Renamed `StringUtils.escapeSJISByteStream()` to more appropriate `escapeEasternUnicodeByteStream()`.
- Not specifying database in URL caused `MalformedURL` exception. (Bug #3554)
- Auto-convert MySQL encoding names to Java encoding names if used for `characterEncoding` property.

- Added encoding names that are recognized on some JVMs to fix case where they were reverse-mapped to MySQL encoding names incorrectly.
- Use `junit.textui.TestRunner` for all unit tests (to allow them to be run from the command line outside of Ant or Eclipse).
- `UpdatableResultSet` not picking up default values for `moveToInsertRow()`. (Bug #3557)
- Inconsistent reporting of data type. The server still doesn't return all types for *BLOBs *TEXT correctly, so the driver won't return those correctly. (Bug #3570)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug #3520)
- Fixed regression in `PreparedStatement.setString()` and eastern character encodings.
- Made `StringRegressionTest` 4.1-unicode aware.

C.5.4.7. Changes in MySQL Connector/J 3.0.11-stable (19 February 2004)

- Trigger a `SET NAMES utf8` when encoding is forced to `utf8` *or* `utf-8` via the `characterEncoding` property. Previously, only the Java-style encoding name of `utf-8` would trigger this.
- `AutoReconnect` time was growing faster than exponentially. (Bug #2447)
- Fixed failover always going to last host in list. (Bug #2578)
- Added `useUnbufferedInput` parameter, and now use it by default (due to JVM issue http://developer.java.sun.com/developer/bugParade/bugs/4401235.html)
- Detect `on`/`off` or `1`, `2`, `3` form of `lower_case_table_names` value on server.
- Return `java.lang.Integer` for `TINYINT` and `SMALLINT` types from `ResultSetMetaData.getColumnClassName()`. (Bug #2852)
- Return `java.lang.Double` for `FLOAT` type from `ResultSetMetaData.getColumnClassName()`. (Bug #2855)
- Return `[B` instead of `java.lang.Object` for `BINARY`, `VARBINARY` and `LONGVARBINARY` types from `ResultSetMetaData.getColumnClassName()` (JDBC compliance).
- Issue connection events on all instances created from a `ConnectionPoolDataSource`.

C.5.4.8. Changes in MySQL Connector/J 3.0.10-stable (13 January 2004)

- Don't count quoted IDs when inside a 'string' in `PreparedStatement` parsing. (Bug #1511)
- “Friendlier” exception message for `PacketTooLargeException`. (Bug #1534)
- Backported fix for aliased tables and `UpdatableResultSets` in `checkUpdatability()` method from 3.1 branch.
- Fix for `ArrayIndexOutOfBounds` exception when using `Statement.setMaxRows()`. (Bug #1695)
- Barge blobs and split packets not being read correctly. (Bug #1576)
- Fixed regression of `Statement.getGeneratedKeys()` and `REPLACE` statements.
- Subsequent call to `ResultSet.updateFoo()` causes NPE if result set is not updatable. (Bug #1630)
- Fix for 4.1.1-style authentication with no password.

- Foreign Keys column sequence is not consistent in `DatabaseMetaData.getImported/Exported/CrossReference()`. (Bug #1731)
- `DatabaseMetaData.getSystemFunction()` returning bad function `VResultsSion`. (Bug #1775)
- Cross-database updatable result sets are not checked for updatability correctly. (Bug #1592)
- `DatabaseMetaData.getColumns()` should return `Types.LONGVARCHAR` for MySQL `LONGTEXT` type.
- `ResultSet.getObject()` on `TINYINT` and `SMALLINT` columns should return Java type `Integer`. (Bug #1913)
- Added `alwaysClearStream` connection property, which causes the driver to always empty any remaining data on the input stream before each query.
- Added more descriptive error message `Server Configuration Denies Access to DataSource`, as well as retrieval of message from server.
- Autoreconnect code didn't set catalog upon reconnect if it had been changed.
- Implement `ResultSet.updateClob()`.
- `ResultSetMetaData.isCaseSensitive()` returned wrong value for `CHAR`/`VARCHAR` columns.
- Connection property `maxRows` not honored. (Bug #1933)
- Statements being created too many times in `DBMD.extractForeignKeyFromCreateTable()`. (Bug #1925)
- Support escape sequence {fn convert ... }. (Bug #1914)
- `ArrayIndexOutOfBounds` when parameter number == number of parameters + 1. (Bug #1958)
- `ResultSet.findColumn()` should use first matching column name when there are duplicate column names in `SELECT` query (JDBC-compliance). (Bug #2006)
- Removed static synchronization bottleneck from `PreparedStatement.setTimestamp()`.
- Removed static synchronization bottleneck from instance factory method of `SingleByteCharsetConverter`.
- Enable caching of the parsing stage of prepared statements via the `cachePrepStmts`, `prepStmtCacheSize`, and `prepStmtCacheSqlLimit` properties (disabled by default).
- Speed up parsing of `PreparedStatements`, try to use one-pass whenever possible.
- Fixed security exception when used in Applets (applets can't read the system property `file.encoding` which is needed for `LOAD DATA LOCAL INFILE`).
- Use constants for SQLStates.
- Map charset `ko18_ru` to `ko18r` when connected to MySQL-4.1.0 or newer.
- Ensure that `Buffer.writeString()` saves room for the `\0`.
- Fixed exception `Unknown character set 'danish'` on connect with JDK-1.4.0
- Fixed mappings in SQLError to report deadlocks with SQLStates of `41000`.

- `maxRows` property would affect internal statements, so check it for all statement creation internal to the driver, and set to 0 when it is not.

C.5.4.9. Changes in MySQL Connector/J 3.0.9-stable (07 October 2003)

- Faster date handling code in `ResultSet` and `PreparedStatement` (no longer uses `Date` methods that synchronize on static calendars).
- Fixed test for end of buffer in `Buffer.readString()`.
- Fixed `ResultSet.previous()` behavior to move current position to before result set when on first row of result set. (Bug #496)
- Fixed `Statement` and `PreparedStatement` issuing bogus queries when `setMaxRows()` had been used and a `LIMIT` clause was present in the query.
- `refreshRow` didn't work when primary key values contained values that needed to be escaped (they ended up being doubly escaped). (Bug #661)
- Support `InnoDB` contraint names when extracting foreign key information in `DatabaseMetaData` (implementing ideas from Parwinder Sekhon). (Bug #517, Bug #664)
- Backported 4.1 protocol changes from 3.1 branch (server-side SQL states, new field information, larger client capability flags, connect-with-database, and so forth).
- Fix `UpdatableResultSet` to return values for `get`*`XXX`*`()` when on insert row. (Bug #675)
- The `insertRow` in an `UpdatableResultSet` is now loaded with the default column values when `moveToInsertRow()` is called. (Bug #688)
- `DatabaseMetaData.getColumns()` wasn't returning `NULL` for default values that are specified as `NULL`.
- Change default statement type/concurrency to `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY` (spec compliance).
- Don't try and reset isolation level on reconnect if MySQL doesn't support them.
- Don't wrap `SQLExceptions` in `RowDataDynamic`.
- Don't change timestamp TZ twice if `useTimezone==true`. (Bug #774)
- Fixed regression in large split-packet handling. (Bug #848)
- Better diagnostic error messages in exceptions for “streaming” result sets.
- Issue exception on `ResultSet.get`*`XXX`*`()` on empty result set (wasn't caught in some cases).
- Don't hide messages from exceptions thrown in I/O layers.
- Don't fire connection closed events when closing pooled connections, or on `PooledConnection.getConnection()` with already open connections. (Bug #884)
- Clip +/- INF (to smallest and largest representative values for the type in MySQL) and NaN (to 0) for `setDouble`/`setFloat()`, and issue a warning on the statement when the server does not support +/- INF or NaN.
- Double-escaping of `'\'` when charset is SJIS or GBK and `'\'` appears in non-escaped input. (Bug #879)

- When emptying input stream of unused rows for “streaming” result sets, have the current thread `yield()` every 100 rows in order to not monopolize CPU time.
- `DatabaseMetaData.getColumns()` getting confused about the keyword “set” in character columns. (Bug #1099)
- Fixed deadlock issue with `Statement.setMaxRows()`.
- Fixed `CLOB.truncate()`. (Bug #1130)
- Optimized `CLOB.setChracterStream()`. (Bug #1131)
- Made `databaseName`, `portNumber`, and `serverName` optional parameters for `MysqlDataSourceFactory`. (Bug #1246)
- `ResultSet.get/setString` mashing char 127. (Bug #1247)
- Backported authentication changes for 4.1.1 and newer from 3.1 branch.
- Added `com.mysql.jdbc.util.BaseBugReport` to help creation of testcases for bug reports.
- Added property to “clobber” streaming results, by setting the `clobberStreamingResults` property to `true` (the default is `false`). This will cause a “streaming” `ResultSet` to be automatically closed, and any oustanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.

C.5.4.10. Changes in MySQL Connector/J 3.0.8-stable (23 May 2003)

- Allow bogus URLs in `Driver.getPropertyInfo()`.
- Return list of generated keys when using multi-value `INSERTS` with `Statement.getGeneratedKeys()`.
- Use JVM charset with filenames and `LOAD DATA [LOCAL] INFILE`.
- Fix infinite loop with `Connection.cleanup()`.
- Changed Ant target `compile-core` to `compile-driver`, and made testsuite compilation a separate target.
- Fixed result set not getting set for `Statement.executeUpdate()`, which affected `getGeneratedKeys()` and `getUpdateCount()` in some cases.
- Unicode character 0xFFFF in a string would cause the driver to throw an `ArrayOutOfBoundsException`. (Bug #378).
- Return correct number of generated keys when using `REPLACE` statements.
- Fix problem detecting server character set in some cases.
- Fix row data decoding error when using *very* large packets.
- Optimized row data decoding.
- Issue exception when operating on an already closed prepared statement.
- Fixed SJIS encoding bug, thanks to Naoto Sato.
- Optimized usage of `EscapeProcessor`.

- Allow multiple calls to `Statement.close()`.

C.5.4.11. Changes in MySQL Connector/J 3.0.7-stable (08 April 2003)

- Fixed `MysqlPooledConnection.close()` calling wrong event type.
- Fixed `StringIndexOutOfBoundsException` in `PreparedStatement.setClob()`.
- 4.1 Column Metadata fixes.
- Remove synchronization from `Driver.connect()` and `Driver.acceptsUrl()`.
- `IOExceptions` during a transaction now cause the `Connection` to be closed.
- Fixed missing conversion for `YEAR` type in `ResultSetMetaData.getColumnTypeName()`.
- Don't pick up indexes that start with `pri` as primary keys for `DBMD.getPrimaryKeys()`.
- Throw `SQLExceptions` when trying to do operations on a forcefully closed `Connection` (that is, when a communication link failure occurs).
- You can now toggle profiling on/off using `Connection.setProfileSql(boolean)`.
- Fixed charset issues with database metadata (charset was not getting set correctly).
- Updatable `ResultSets` can now be created for aliased tables/columns when connected to MySQL-4.1 or newer.
- Fixed `LOAD DATA LOCAL INFILE` bug when file > `max_allowed_packet`.
- Fixed escaping of 0x5c (`'\'`) character for GBK and Big5 charsets.
- Fixed `ResultSet.getTimestamp()` when underlying field is of type `DATE`.
- Ensure that packet size from `alignPacketSize()` does not exceed `max_allowed_packet` (JVM bug)
- Don't reset `Connection.isReadOnly()` when autoReconnecting.

C.5.4.12. Changes in MySQL Connector/J 3.0.6-stable (18 February 2003)

- Fixed `ResultSetMetaData` to return `""` when catalog not known. Fixes `NullPointerExceptions` with Sun's `CachedRowSet`.
- Fixed `DBMD.getTypeInfo()` and `DBMD.getColumns()` returning different value for precision in `TEXT` and `BLOB` types.
- Allow ignoring of warning for “non transactional tables” during rollback (compliance/usability) by setting `ignoreNonTxTables` property to `true`.
- Fixed `SQLExceptions` getting swallowed on initial connect.
- Fixed `Statement.setMaxRows()` to stop sending `LIMIT` type queries when not needed (performance).
- Clean up `Statement` query/method mismatch tests (that is, `INSERT` not allowed with `.executeQuery()`).
- More checks added in `ResultSet` traversal method to catch when in closed state.

- Fixed `ResultSetMetaData.isWritable()` to return correct value.
- Add “window” of different `NULL` sorting behavior to `DBMD.nullsAreSortedAtStart` (4.0.2 to 4.0.10, true; otherwise, no).
- Implemented `Blob.setBytes()`. You still need to pass the resultant `Blob` back into an updatable `ResultSet` or `PreparedStatement` to persist the changes, because MySQL does not support “locators”.
- Backported 4.1 charset field info changes from Connector/J 3.1.

C.5.4.13. Changes in MySQL Connector/J 3.0.5-gamma (22 January 2003)

- Fixed `Buffer.fastSkipLenString()` causing `ArrayIndexOutOfBounds` exceptions with some queries when unpacking fields.
- Implemented an empty `TypeMap` for `Connection.getTypeMap()` so that some third-party apps work with MySQL (IBM WebSphere 5.0 Connection pool).
- Added missing `LONGTEXT` type to `DBMD.getColumns()`.
- Retrieve `TX_ISOLATION` from database for `Connection.getTransactionIsolation()` when the MySQL version supports it, instead of an instance variable.
- Quote table names in `DatabaseMetaData.getColumns()`, `getPrimaryKeys()`, `getIndexInfo()`, `getBestRowIdentifier()`.
- Greatly reduce memory required for `setBinaryStream()` in `PreparedStatements`.
- Fixed `ResultSet.isBeforeFirst()` for empty result sets.
- Added update options for foreign key metadata.

C.5.4.14. Changes in MySQL Connector/J 3.0.4-gamma (06 January 2003)

- Added quoted identifiers to database names for `Connection.setCatalog`.
- Added support for quoted identifiers in `PreparedStatement` parser.
- Streamlined character conversion and `byte[]` handling in `PreparedStatements` for `setByte()`.
- Reduce memory footprint of `PreparedStatements` by sharing outbound packet with `MysqlIO`.
- Added `strictUpdates` property to allow control of amount of checking for “correctness” of updatable result sets. Set this to `false` if you want faster updatable result sets and you know that you create them from `SELECT` statements on tables with primary keys and that you have selected all primary keys in your query.
- Added support for 4.0.8-style large packets.
- Fixed `PreparedStatement.executeBatch()` parameter overwriting.

C.5.4.15. Changes in MySQL Connector/J 3.0.3-dev (17 December 2002)

- Changed `charsToByte` in `SingleByteCharConverter` to be non-static.
- Changed `SingleByteCharConverter` to use lazy initialization of each converter.
- Fixed charset handling in `Fields.java`.

- Implemented `Connection.nativeSQL()`.
- More robust escape tokenizer: Recognize `--` comments, and allow nested escape sequences (see `testsuite.EscapeProcessingTest`).
- `DBMD.getImported/ExportedKeys()` now handles multiple foreign keys per table.
- Fixed `ResultSetMetaData.getPrecision()` returning incorrect values for some floating-point types.
- Fixed `ResultSetMetaData.getColumnTypeName()` returning `BLOB` for `TEXT` and `TEXT` for `BLOB` types.
- Fixed `Buffer.isLastDataPacket()` for 4.1 and newer servers.
- Added `CLIENT_LONG_FLAG` to be able to get more column flags (`isAutoIncrement()` being the most important).
- Because of above, implemented `ResultSetMetaData.isAutoIncrement()` to use `Field.isAutoIncrement()`.
- Honor `lower_case_table_names` when enabled in the server when doing table name comparisons in `DatabaseMetaData` methods.
- Some MySQL-4.1 protocol support (extended field info from selects).
- Use non-aliased table/column names and database names to fullly qualify tables and columns in `UpdatableResultSet` (requires MySQL-4.1 or newer).
- Allow user to alter behavior of `Statement`/ `PreparedStatement.executeBatch()` via `continueBatchOnError` property (defaults to `true`).
- Check for connection closed in more `Connection` methods (`createStatement`, `prepareStatement`, `setTransactionIsolation`, `setAutoCommit`).
- More robust implementation of updatable result sets. Checks that *all* primary keys of the table have been selected.
- `LOAD DATA LOCAL INFILE ...` now works, if your server is configured to allow it. Can be turned off with the `allowLoadLocalInfile` property (see the `README`).
- Substitute `'?'` for unknown character conversions in single-byte character sets instead of `'\0'`.
- `NamedPipeSocketFactory` now works (only intended for Windows), see `README` for instructions.

C.5.4.16. Changes in MySQL Connector/J 3.0.2-dev (08 November 2002)

- Fixed issue with updatable result sets and `PreparedStatements` not working.
- Fixed `ResultSet.setFetchDirection(FETCH_UNKNOWN)`.
- Fixed issue when calling `Statement.setFetchSize()` when using arbitrary values.
- Fixed incorrect conversion in `ResultSet.getLong()`.
- Implemented `ResultSet.updateBlob()`.
- Removed duplicate code from `UpdatableResultSet` (it can be inherited from `ResultSet`, the extra code for each method to handle updatability I thought might someday be necessary has not been needed).

- Fixed `UnsupportedEncodingException` thrown when “forcing” a character encoding via properties.
- Fixed various non-ASCII character encoding issues.
- Added driver property `useHostsInPrivileges`. Defaults to true. Affects whether or not `@hostname` will be used in `DBMD.getColumn/TablePrivileges`.
- All `DBMD` result set columns describing schemas now return `NULL` to be more compliant with the behavior of other JDBC drivers for other database systems (MySQL does not support schemas).
- Added SSL support. See `README` for information on how to use it.
- Properly restore connection properties when autoReconnecting or failing-over, including `autoCommit` state, and isolation level.
- Use `SHOW CREATE TABLE` when possible for determining foreign key information for `DatabaseMetaData`. Also allows cascade options for `DELETE` information to be returned.
- Escape `0x5c` character in strings for the SJIS charset.
- Fixed start position off-by-1 error in `Clob.getSubString()`.
- Implemented `Clob.truncate()`.
- Implemented `Clob.setString()`.
- Implemented `Clob.setAsciiStream()`.
- Implemented `Clob.setCharacterStream()`.
- Added `com.mysql.jdbc.MiniAdmin` class, which allows you to send `shutdown` command to MySQL server. This is intended to be used when “embedding” Java and MySQL server together in an end-user application.
- Added `connectTimeout` parameter that allows users of JDK-1.4 and newer to specify a maxium time to wait to establish a connection.
- Failover and `autoReconnect` work only when the connection is in an `autoCommit(false)` state, in order to stay transaction-safe.
- Added `queriesBeforeRetryMaster` property that specifies how many queries to issue when failed over before attempting to reconnect to the master (defaults to 50).
- Fixed `DBMD.supportsResultSetConcurrency()` so that it returns true for `ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`.
- Fixed `ResultSet.isLast()` for empty result sets (should return `false`).
- `PreparedStatement` now honors stream lengths in setBinary/Ascii/Character Stream() unless you set the connection property `useStreamLengthsInPrepStmts` to `false`.
- Removed some not-needed temporary object creation by smarter use of `Strings` in `EscapeProcessor`, `Connection` and `DatabaseMetaData` classes.

C.5.4.17. Changes in MySQL Connector/J 3.0.1-dev (21 September 2002)

- Fixed `ResultSet.getRow()` off-by-one bug.

- Fixed `RowDataStatic.getAt()` off-by-one bug.
- Added limited `Clob` functionality (`ResultSet.getClob()`, `PreparedStatemtent.setClob()`, `PreparedStatement.setObject(Clob)`.
- Added `socketTimeout` parameter to URL.
- `Connection.isClosed()` no longer “pings” the server.
- `Connection.close()` issues `rollback()` when `getAutoCommit()` is `false`.
- Added `paranoid` parameter, which sanitizes error messages by removing “sensitive” information from them (such as hostnames, ports, or usernames), as well as clearing “sensitive” data structures when possible.
- Fixed `ResultSetMetaData.isSigned()` for `TINYINT` and `BIGINT`.
- Charsets now automatically detected. Optimized code for single-byte character set conversion.
- Implemented `ResultSet.getCharacterStream()`.
- Added `LOCAL TEMPORARY` to table types in `DatabaseMetaData.getTableTypes()`.
- Massive code clean-up to follow Java coding conventions (the time had come).

C.5.4.18. Changes in MySQL Connector/J 3.0.0-dev (31 July 2002)

- **!!! LICENSE CHANGE !!!** The driver is now GPL. If you need non-GPL licenses, please contact me `<mark@mysql.com>`.
- JDBC-3.0 functionality including `Statement/PreparedStatement.getGeneratedKeys()` and `ResultSet.getURL()`.
- Performance enchancements: Driver is now 50–100% faster in most situations, and creates fewer temporary objects.
- Repackaging: New driver name is `com.mysql.jdbc.Driver`, old name still works, though (the driver is now provided by MySQL-AB).
- Better checking for closed connections in `Statement` and `PreparedStatement`.
- Support for streaming (row-by-row) result sets (see `README`) Thanks to Doron.
- Support for large packets (new addition to MySQL-4.0 protocol), see `README` for more information.
- JDBC Compliance: Passes all tests besides stored procedure tests.
- Fix and sort primary key names in `DBMetaData` (SF bugs 582086 and 582086).
- Float types now reported as `java.sql.Types.FLOAT` (SF bug 579573).
- `ResultSet.getTimestamp()` now works for `DATE` types (SF bug 559134).
- `ResultSet.getDate/Time/Timestamp` now recognizes all forms of invalid values that have been set to all zeros by MySQL (SF bug 586058).
- Testsuite now uses Junit (which you can get from http://www.junit.org.
- The driver now only works with JDK-1.2 or newer.

- Added multi-host failover support (see `README`).
- General source-code cleanup.
- Overall speed improvements via controlling transient object creation in `MysqlIO` class when reading packets.
- Performance improvements in string handling and field metadata creation (lazily instantiated) contributed by Alex Twisleton-Wykeham-Fiennes.

C.5.5. Changes in MySQL Connector/J 2.0.x

C.5.5.1. Changes in MySQL Connector/J 2.0.14 (16 May 2002)

- More code cleanup.
- `PreparedStatement` now releases resources on `.close()`. (SF bug 553268)
- Quoted identifiers not used if server version does not support them. Also, if server started with `--ansi` or `--sql-mode=ANSI_QUOTES`, '"' will be used as an identifier quote character, otherwise '`' will be used.
- `ResultSet.getDouble()` now uses code built into JDK to be more precise (but slower).
- `LogicalHandle.isClosed()` calls through to physical connection.
- Added SQL profiling (to `STDERR`). Set `profileSql=true` in your JDBC URL. See `README` for more information.
- Fixed typo for `relaxAutoCommit` parameter.

C.5.5.2. Changes in MySQL Connector/J 2.0.13 (24 April 2002)

- More code cleanup.
- Fixed unicode chars being read incorrectly. (SF bug 541088)
- Faster blob escaping for `PrepStmt`.
- Added `set`/`getPortNumber()` to `DataSource(s)`. (SF bug 548167)
- Added `setURL()` to `MySQLXADataSource`. (SF bug 546019)
- `PreparedStatement.toString()` fixed. (SF bug 534026)
- `ResultSetMetaData.getColumnClassName()` now implemented.
- Rudimentary version of `Statement.getGeneratedKeys()` from JDBC-3.0 now implemented (you need to be using JDK-1.4 for this to work, I believe).
- `DBMetaData.getIndexInfo()` - bad PAGES fixed. (SF BUG 542201)

C.5.5.3. Changes in MySQL Connector/J 2.0.12 (07 April 2002)

- General code cleanup.
- Added `getIdleFor()` method to `Connection` and `MysqlLogicalHandle`.
- Relaxed synchronization in all classes, should fix 520615 and 520393.

- Added `getTable/ColumnPrivileges()` to DBMD (fixes 484502).
- Added new types to `getTypeInfo()`, fixed existing types thanks to Al Davis and Kid Kalanon.
- Added support for `BIT` types (51870) to `PreparedStatement`.
- Fixed `getRow()` bug (527165) in `ResultSet`.
- Fixes for `ResultSet` updatability in `PreparedStatement`.
- Fixed time zone off-by-1-hour bug in `PreparedStatement` (538286, 528785).
- `ResultSet`: Fixed updatability (values being set to `null` if not updated).
- `DataSources` - fixed `setUrl` bug (511614, 525565), wrong datasource class name (532816, 528767).
- Added identifier quoting to all `DatabaseMetaData` methods that need them (should fix 518108).
- Added support for `YEAR` type (533556).
- `ResultSet.insertRow()` should now detect auto_increment fields in most cases and use that value in the new row. This detection will not work in multi-valued keys, however, due to the fact that the MySQL protocol does not return this information.
- `ResultSet.refreshRow()` implemented.
- Fixed `testsuite.Traversal` `afterLast()` bug, thanks to Igor Lastric.

C.5.5.4. Changes in MySQL Connector/J 2.0.11 (27 January 2002)

- Fixed missing `DELETE_RULE` value in `DBMD.getImported/ExportedKeys()` and `getCrossReference()`.
- Full synchronization of `Statement.java`.
- More changes to fix `Unexpected end of input stream` errors when reading `BLOB` values. This should be the last fix.

C.5.5.5. Changes in MySQL Connector/J 2.0.10 (24 January 2002)

- Fixed spurious `Unexpected end of input stream` errors in `MysqlIO` (bug 507456).
- Fixed null-pointer-exceptions when using `MysqlConnectionPoolDataSource` with Websphere 4 (bug 505839).

C.5.5.6. Changes in MySQL Connector/J 2.0.9 (13 January 2002)

- `Ant` build was corrupting included `jar` files, fixed (bug 487669).
- Fixed extra memory allocation in `MysqlIO.readPacket()` (bug 488663).
- Implementation of `DatabaseMetaData.getExported/ImportedKeys()` and `getCrossReference()`.
- Full synchronization on methods modifying instance and class-shared references, driver should be entirely thread-safe now (please let me know if you have problems).
- `DataSource` implementations moved to `org.gjt.mm.mysql.jdbc2.optional` package, and (initial) implementations of `PooledConnectionDataSource` and `XADataSource` are in place (thanks to Todd Wolff for the implementation and testing of `PooledConnectionDataSource` with IBM WebSphere 4).

- Added detection of network connection being closed when reading packets (thanks to Todd Lizambri).
- Fixed quoting error with escape processor (bug 486265).
- Report batch update support through `DatabaseMetaData` (bug 495101).
- Fixed off-by-one-hour error in `PreparedStatement.setTimestamp()` (bug 491577).
- Removed concatenation support from driver (the `||` operator), as older versions of VisualAge seem to be the only thing that use it, and it conflicts with the logical `||` operator. You will need to start `mysqld` with the `--ansi` flag to use the `||` operator as concatenation (bug 491680).
- Fixed casting bug in `PreparedStatement` (bug 488663).

C.5.5.7. Changes in MySQL Connector/J 2.0.8 (25 November 2001)

- Batch updates now supported (thanks to some inspiration from Daniel Rall).
- `XADataSource`/`ConnectionPoolDataSource` code (experimental)
- `PreparedStatement.setAnyNumericType()` now handles positive exponents correctly (adds `+` so MySQL can understand it).
- `DatabaseMetaData.getPrimaryKeys()` and `getBestRowIdentifier()` are now more robust in identifying primary keys (matches regardless of case or abbreviation/full spelling of `Primary Key` in `Key_type` column).

C.5.5.8. Changes in MySQL Connector/J 2.0.7 (24 October 2001)

- `PreparedStatement.setCharacterStream()` now implemented
- Fixed dangling socket problem when in high availability (`autoReconnect=true`) mode, and finalizer for `Connection` will close any dangling sockets on GC.
- Fixed `ResultSetMetaData.getPrecision()` returning one less than actual on newer versions of MySQL.
- `ResultSet.getBlob()` now returns `null` if column value was `null`.
- Character sets read from database if `useUnicode=true` and `characterEncoding` is not set. (thanks to Dmitry Vereshchagin)
- Initial transaction isolation level read from database (if avaialable). (thanks to Dmitry Vereshchagin)
- Fixed `DatabaseMetaData.supportsTransactions()`, and `supportsTransactionIsolationLevel()` and `getTypeInfo()` `SQL_DATETIME_SUB` and `SQL_DATA_TYPE` fields not being readable.
- Fixed `PreparedStatement` generating SQL that would end up with syntax errors for some queries.
- Fixed `ResultSet.isAfterLast()` always returning `false`.
- Fixed time zone issue in `PreparedStatement.setTimestamp()`. (thanks to Erik Olofsson)
- Captialize type names when `captializeTypeNames=true` is passed in URL or properties (for WebObjects. (thanks to Anjo Krank)
- Updatable result sets now correctly handle `NULL` values in fields.

- PreparedStatement.setDouble() now uses full-precision doubles (reverting a fix made earlier to truncate them).
- PreparedStatement.setBoolean() will use 1/0 for values if your MySQL version is 3.21.23 or higher.

C.5.5.9. Changes in MySQL Connector/J 2.0.6 (16 June 2001)

- Fixed `PreparedStatement` parameter checking.
- Fixed case-sensitive column names in `ResultSet.java`.

C.5.5.10. Changes in MySQL Connector/J 2.0.5 (13 June 2001)

- Fixed `ResultSet.getBlob()` `ArrayIndex` out-of-bounds.
- Fixed `ResultSetMetaData.getColumnTypeName` for `TEXT`/`BLOB`.
- Fixed `ArrayIndexOutOfBounds` when sending large `BLOB` queries. (Max size packet was not being set)
- Added `ISOLATION` level support to `Connection.setIsolationLevel()`
- Fixed NPE on `PreparedStatement.executeUpdate()` when all columns have not been set.
- Fixed data parsing of `TIMESTAMP` values with 2-digit years.
- Added `Byte` to `PreparedStatement.setObject()`.
- `ResultSet.getBoolean()` now recognizes `-1` as `true`.
- `ResultSet` has +/-Inf/inf support.
- `ResultSet.insertRow()` works now, even if not all columns are set (they will be set to `NULL`).
- `DataBaseMetaData.getCrossReference()` no longer `ArrayIndexOOB`.
- `getObject()` on `ResultSet` correctly does `TINYINT`->`Byte` and `SMALLINT`->`Short`.

C.5.5.11. Changes in MySQL Connector/J 2.0.3 (03 December 2000)

- Implemented `getBigDecimal()` without scale component for JDBC2.
- Fixed composite key problem with updatable result sets.
- Added detection of -/+INF for doubles.
- Faster ASCII string operations.
- Fixed incorrect detection of `MAX_ALLOWED_PACKET`, so sending large blobs should work now.
- Fixed off-by-one error in `java.sql.Blob` implementation code.
- Added `ultraDevHack` URL parameter, set to `true` to allow (broken) Macromedia UltraDev to use the driver.

C.5.5.12. Changes in MySQL Connector/J 2.0.1 (06 April 2000)

- Fixed `RSMD.isWritable()` returning wrong value. Thanks to Moritz Maass.

- Cleaned up exception handling when driver connects.
- Columns that are of type `TEXT` now return as `Strings` when you use `getObject()`.
- `DatabaseMetaData.getPrimaryKeys()` now works correctly with respect to `key_seq`. Thanks to Brian Slesinsky.
- No escape processing is done on `PreparedStatements` anymore per JDBC spec.
- Fixed many JDBC-2.0 traversal, positioning bugs, especially with respect to empty result sets. Thanks to Ron Smits, Nick Brook, Cessar Garcia and Carlos Martinez.
- Fixed some issues with updatability support in `ResultSet` when using multiple primary keys.

C.5.5.13. Changes in MySQL Connector/J 2.0.0pre5 (21 February 2000)

- Fixed Bad Handshake problem.

C.5.5.14. Changes in MySQL Connector/J 2.0.0pre4 (10 January 2000)

- Fixes to ResultSet for insertRow() - Thanks to Cesar Garcia
- Fix to Driver to recognize JDBC-2.0 by loading a JDBC-2.0 class, instead of relying on JDK version numbers. Thanks to John Baker.
- Fixed ResultSet to return correct row numbers
- Statement.getUpdateCount() now returns rows matched, instead of rows actually updated, which is more SQL-92 like.

10-29-99

- Statement/PreparedStatement.getMoreResults() bug fixed. Thanks to Noel J. Bergman.
- Added Short as a type to PreparedStatement.setObject(). Thanks to Jeff Crowder
- Driver now automagically configures maximum/preferred packet sizes by querying server.
- Autoreconnect code uses fast ping command if server supports it.
- Fixed various bugs with respect to packet sizing when reading from the server and when alloc'ing to write to the server.

C.5.5.15. Changes in MySQL Connector/J 2.0.0pre (17 August 1999)

- Now compiles under JDK-1.2. The driver supports both JDK-1.1 and JDK-1.2 at the same time through a core set of classes. The driver will load the appropriate interface classes at runtime by figuring out which JVM version you are using.
- Fixes for result sets with all nulls in the first row. (Pointed out by Tim Endres)
- Fixes to column numbers in SQLExceptions in ResultSet (Thanks to Blas Rodriguez Somoza)
- The database no longer needs to specified to connect. (Thanks to Christian Motschke)

C.5.6. Changes in MySQL Connector/J 1.2b (04 July 1999)

- Better Documentation (in progress), in doc/mm.doc/book1.html

- DBMD now allows null for a column name pattern (not in spec), which it changes to '%'.
- DBMD now has correct types/lengths for getXXX().
- ResultSet.getDate(), getTime(), and getTimestamp() fixes. (contributed by Alan Wilken)
- EscapeProcessor now handles \{ \} and { or } inside quotes correctly. (thanks to Alik for some ideas on how to fix it)
- Fixes to properties handling in Connection. (contributed by Juho Tikkala)
- ResultSet.getObject() now returns null for NULL columns in the table, rather than bombing out. (thanks to Ben Grosman)
- ResultSet.getObject() now returns Strings for types from MySQL that it doesn't know about. (Suggested by Chris Perdue)
- Removed DataInput/Output streams, not needed, 1/2 number of method calls per IO operation.
- Use default character encoding if one is not specified. This is a work-around for broken JVMs, because according to spec, EVERY JVM must support "ISO8859_1", but they don't.
- Fixed Connection to use the platform character encoding instead of "ISO8859_1" if one isn't explicitly set. This fixes problems people were having loading the character- converter classes that didn't always exist (JVM bug). (thanks to Fritz Elfert for pointing out this problem)
- Changed MysqlIO to re-use packets where possible to reduce memory usage.
- Fixed escape-processor bugs pertaining to {} inside quotes.

C.5.7. Changes in MySQL Connector/J 1.2.x and lower

C.5.7.1. Changes in MySQL Connector/J 1.2a (14 April 1999)

- Fixed character-set support for non-Javasoft JVMs (thanks to many people for pointing it out)
- Fixed ResultSet.getBoolean() to recognize 'y' & 'n' as well as '1' & '0' as boolean flags. (thanks to Tim Pizey)
- Fixed ResultSet.getTimestamp() to give better performance. (thanks to Richard Swift)
- Fixed getByte() for numeric types. (thanks to Ray Bellis)
- Fixed DatabaseMetaData.getTypeInfo() for DATE type. (thanks to Paul Johnston)
- Fixed EscapeProcessor for "fn" calls. (thanks to Piyush Shah at locomotive.org)
- Fixed EscapeProcessor to not do extraneous work if there are no escape codes. (thanks to Ryan Gustafson)
- Fixed Driver to parse URLs of the form "jdbc:mysql://host:port" (thanks to Richard Lobb)

C.5.7.2. Changes in MySQL Connector/J 1.1i (24 March 1999)

- Fixed Timestamps for PreparedStatements
- Fixed null pointer exceptions in RSMD and RS

- Re-compiled with jikes for valid class files (thanks ms!)

C.5.7.3. Changes in MySQL Connector/J 1.1h (08 March 1999)

- Fixed escape processor to deal with unmatched { and } (thanks to Craig Coles)
- Fixed escape processor to create more portable (between DATETIME and TIMESTAMP types) representations so that it will work with BETWEEN clauses. (thanks to Craig Longman)
- MysqlIO.quit() now closes the socket connection. Before, after many failed connections some OS's would run out of file descriptors. (thanks to Michael Brinkman)
- Fixed NullPointerException in Driver.getPropertyInfo. (thanks to Dave Potts)
- Fixes to MysqlDefs to allow all *text fields to be retrieved as Strings. (thanks to Chris at Leverage)
- Fixed setDouble in PreparedStatement for large numbers to avoid sending scientific notation to the database. (thanks to J.S. Ferguson)
- Fixed getScale() and getPrecision() in RSMD. (contrib'd by James Klicman)
- Fixed getObject() when field was DECIMAL or NUMERIC (thanks to Bert Hobbs)
- DBMD.getTables() bombed when passed a null table-name pattern. Fixed. (thanks to Richard Lobb)
- Added check for "client not authorized" errors during connect. (thanks to Hannes Wallnoefer)

C.5.7.4. Changes in MySQL Connector/J 1.1g (19 February 1999)

- Result set rows are now byte arrays. Blobs and Unicode work bidriectonally now. The useUnicode and encoding options are implemented now.
- Fixes to PreparedStatement to send binary set by setXXXStream to be sent untouched to the MySQL server.
- Fixes to getDriverPropertyInfo().

C.5.7.5. Changes in MySQL Connector/J 1.1f (31 December 1998)

- Changed all ResultSet fields to Strings, this should allow Unicode to work, but your JVM must be able to convert between the character sets. This should also make reading data from the server be a bit quicker, because there is now no conversion from StringBuffer to String.
- Changed PreparedStatement.streamToString() to be more efficient (code from Uwe Schaefer).
- URL parsing is more robust (throws SQL exceptions on errors rather than NullPointerExceptions)
- PreparedStatement now can convert Strings to Time/Date values via setObject() (code from Robert Currey).
- IO no longer hangs in Buffer.readInt(), that bug was introduced in 1.1d when changing to all byte-arrays for result sets. (Pointed out by Samo Login)

C.5.7.6. Changes in MySQL Connector/J 1.1b (03 November 1998)

- Fixes to DatabaseMetaData to allow both IBM VA and J-Builder to work. Let me know how it goes. (thanks to Jac Kersing)

- Fix to ResultSet.getBoolean() for NULL strings (thanks to Barry Lagerweij)
- Beginning of code cleanup, and formatting. Getting ready to branch this off to a parallel JDBC-2.0 source tree.
- Added "final" modifier to critical sections in MysqlIO and Buffer to allow compiler to inline methods for speed.

9-29-98

- If object references passed to setXXX() in PreparedStatement are null, setNull() is automatically called for you. (Thanks for the suggestion goes to Erik Ostrom)
- setObject() in PreparedStatement will now attempt to write a serialized representation of the object to the database for objects of Types.OTHER and objects of unknown type.
- Util now has a static method readObject() which given a ResultSet and a column index will re-instantiate an object serialized in the above manner.

C.5.7.7. Changes in MySQL Connector/J 1.1 (02 September 1998)

- Got rid of "ugly hack" in MysqlIO.nextRow(). Rather than catch an exception, Buffer.isLastDataPacket() was fixed.
- Connection.getCatalog() and Connection.setCatalog() should work now.
- Statement.setMaxRows() works, as well as setting by property maxRows. Statement.setMaxRows() overrides maxRows set via properties or url parameters.
- Automatic re-connection is available. Because it has to "ping" the database before each query, it is turned off by default. To use it, pass in "autoReconnect=true" in the connection URL. You may also change the number of reconnect tries, and the initial timeout value via "maxReconnects=n" (default 3) and "initialTimeout=n" (seconds, default 2) parameters. The timeout is an exponential backoff type of timeout; for example, if you have initial timeout of 2 seconds, and maxReconnects of 3, then the driver will timeout 2 seconds, 4 seconds, then 16 seconds between each re-connection attempt.

C.5.7.8. Changes in MySQL Connector/J 1.0 (24 August 1998)

- Fixed handling of blob data in Buffer.java
- Fixed bug with authentication packet being sized too small.
- The JDBC Driver is now under the LPGL

8-14-98

- Fixed Buffer.readLenString() to correctly read data for BLOBS.
- Fixed PreparedStatement.stringToStream to correctly read data for BLOBS.
- Fixed PreparedStatement.setDate() to not add a day. (above fixes thanks to Vincent Partington)
- Added URL parameter parsing (?user=... and so forth).

C.5.7.9. Changes in MySQL Connector/J 0.9d (04 August 1998)

- Big news! New package name. Tim Endres from ICE Engineering is starting a new source tree for GNU GPL'd Java software. He's graciously given me the org.gjt.mm package directory to use, so now the

driver is in the org.gjt.mm.mysql package scheme. I'm "legal" now. Look for more information on Tim's project soon.

- Now using dynamically sized packets to reduce memory usage when sending commands to the DB.
- Small fixes to getTypeInfo() for parameters, and so forth.
- DatabaseMetaData is now fully implemented. Let me know if these drivers work with the various IDEs out there. I've heard that they're working with JBuilder right now.
- Added JavaDoc documentation to the package.
- Package now available in .zip or .tar.gz.

C.5.7.10. Changes in MySQL Connector/J 0.9 (28 July 1998)

- Implemented getTypeInfo(). Connection.rollback() now throws an SQLException per the JDBC spec.
- Added PreparedStatement that supports all JDBC API methods for PreparedStatement including InputStreams. Please check this out and let me know if anything is broken.
- Fixed a bug in ResultSet that would break some queries that only returned 1 row.
- Fixed bugs in DatabaseMetaData.getTables(), DatabaseMetaData.getColumns() and DatabaseMetaData.getCatalogs().
- Added functionality to Statement that allows executeUpdate() to store values for IDs that are automatically generated for AUTO_INCREMENT fields. Basically, after an executeUpdate(), look at the SQLWarnings for warnings like "LAST_INSERTED_ID = 'some number', COMMAND = 'your SQL query'". If you are using AUTO_INCREMENT fields in your tables and are executing a lot of executeUpdate()s on one Statement, be sure to clearWarnings() every so often to save memory.

C.5.7.11. Changes in MySQL Connector/J 0.8 (06 July 1998)

- Split MysqlIO and Buffer to separate classes. Some ClassLoaders gave an IllegalAccess error for some fields in those two classes. Now mm.mysql works in applets and all classloaders. Thanks to Joe Ennis <jce@mail.boone.com> for pointing out the problem and working on a fix with me.

C.5.7.12. Changes in MySQL Connector/J 0.7 (01 July 1998)

- Fixed DatabaseMetadata problems in getColumns() and bug in switch statement in the Field constructor. Thanks to Costin Manolache <costin@tdiinc.com> for pointing these out.

C.5.7.13. Changes in MySQL Connector/J 0.6 (21 May 1998)

- Incorporated efficiency changes from Richard Swift <Richard.Swift@kanatek.ca> in `MysqlIO.java` and `ResultSet.java`:
- We're now 15% faster than gwe's driver.
- Started working on `DatabaseMetaData`.
- The following methods are implemented:
  - `getTables()`
  - `getTableTypes()`

- `getColumns`
- `getCatalogs()`

Apéndice D. Portar a otros sistemas

Tabla de contenidos

This appendix helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See Sección 2.1.1, “Sistemas operativos que MySQL soporta”. 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. On Solaris 2.5 we use Sun PThreads (the native thread support in 2.4 and earlier versions is not good enough), on Linux we use LinuxThreads by Xavier Leroy, `<Xavier.Leroy@inria.fr>`.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See `mit-pthreads/README` and Programming POSIX Threads (http://www.humanfactor.com/pthreads/).

Up to MySQL 4.0.2, the MySQL distribution included a patched version of Chris Provenzano's Pthreads from MIT (see the MIT Pthreads Web page at http://www.mit.edu/afs/sipb/project/pthreads/ and a programming introduction at http://www.mit.edu:8001/people/proven/IAP_2000/). These can be used for some operating systems that do not have POSIX threads. See Sección 2.8.5, “Notas sobre MIT-pthreads”.

It is also possible to use another user level thread package named FSU Pthreads (see http://moss.csc.ncsu.edu/~mueller/pthreads/). This implementation is being used for the SCO port.

See the `thr_lock.c` and `thr_alarm.c` programs in the `mysys` directory for some tests/examples of these problems.

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 SPARCworks, Sun Forte, Irix `cc`, HP-UX `aCC`, IBM AIX `xlC_r`), Intel `ecc/icc` and Compaq `cxx`).

To compile only the client use `./configure --without-server`.

There is currently no support for only compiling the server, nor is it likely to be added unless someone has a good reason for it.

If you want/need to change any `Makefile` or the configure script you also need GNU Automake and Autoconf. See Sección 2.8.3, “Instalar desde el árbol de código fuente de desarrollo”.

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 Sección D.1, “Depurar un servidor MySQL”.

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!

D.1. Depurar un servidor MySQL

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) or with `--safe-mode` which disables a lot of optimization that may cause problems. See Sección A.4.2, “Qué hacer si MySQL sigue fallando (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 Capítulo 5, *Administración de bases de datos*. 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 Sección 2.12, “Notas específicas sobre sistemas operativos”.

D.1.1. Compilación de MySQL para depuración

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` or `egcs`, 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 `safe_mysqld` 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 MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See Sección 1.6.1.1, “Las listas de correo de MySQL”. Please use the `mysqlbug` script for all bug reports or questions regarding the MySQL version you are using!

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

D.1.2. Crear ficheros de traza

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's compiled with support for trace files. (On Windows, the debugging server is named `mysqld-debug` rather than `mysqld` as of MySQL 4.1.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `C:\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`.

Note that the trace file become **very big**! If you want 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 ftp the trace file, together with a full bug report, to ftp://ftp.mysql.com/pub/mysql/upload/ so that a MySQL developer can take a look a this.

The trace file is made with the **DBUG** package by Fred Fish. See Sección D.3, “El paquete DBUG”.

D.1.3. Depurar mysqld con 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. We recommend you to upgrade to gdb 5.1 ASAP as thread debugging works much better with this version!

NTPL 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.

In MySQL 4.0.14 and above you should use the --gdb option to mysqld. This installs an interrupt handler for SIGINT (needed to stop mysqld with ^C to set breakpoints) and disable stack tracing and core file handling.

It's 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 -O thread_cache_size= 'max_connections +1'. In most cases just using -O 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:

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   exit
```

See Sección A.4.2, “Qué hacer si MySQL sigue fallando (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 mail generated with `mysqlbug` and mail this to the general MySQL mailing list. See Sección 1.6.1.1, “Las listas de correo de MySQL”.

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.

D.1.4. Usar 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 Sección 5.10.1, “El registro de errroes (Error Log)”. To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to gcc. See Sección D.1.1, “Compilación de MySQL para depuración”.

If the error file contains something like the following:

```
mysqld got signal 11;
The manual section 'Debugging a MySQL server' tells you how to use a
stack trace and/or the core file to produce a readable backtrace that may
help in finding out why mysqld died
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
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

you can find where mysqld died by doing the following:

1. Copy the preceding numbers to a file, for example mysqld.stack.

2. Make a symbol file for the mysqld server:

```
nm -n libexec/mysqld > /tmp/mysqld.sym
```

   Note that most MySQL binary distributions (except for the "debug" packages, where this information is included inside of the binaries themselves) ship with the above file, named mysqld.sym.gz. In this case, you can simply unpack it by doing:

```
gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
```

3. Execute resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack.

   This prints out where mysqld died. If this doesn't help you find out why mysqld died, you should make a bug report and include the output from the above command with the bug report.

   Note however that in most cases it does not help us to just have a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, we would in most cases need to know the query that killed mysqld and preferable a test case so that we can repeat the problem! See Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

D.1.5. El uso de registros (logs) para encontrar la causa de errores de mysqld

Note that before starting mysqld with --log you should check all your tables with myisamchk. See Capítulo 5, *Administración de bases de datos*.

If mysqld dies or hangs, you should start mysqld with --log. When mysqld dies again, you can examine the end of the log file for the query that killed mysqld.

If you are using --log without a file name, 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 Sección 7.2.1, “Sintaxis de EXPLAIN (Obtener información acerca de un SELECT)”.

You can find the queries that take a long time to execute by starting mysqld with --log-slow-queries. See Sección 5.10.4, “El registro de consultas lentas (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 Capítulo 5, *Administración de bases de datos*), and test the queries in the MySQL log files to see if one doesn't work. 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 Sección 5.3.1, “Opciones del comando `mysqld`”.

It's not a good sign if `mysqld` did die unexpectedly, but in this case one shouldn't investigate the `Checking table...` messages but instead try to find out why `mysqld` died.

D.1.6. Crear un caso de prueba tras haber encontrado una tabla corrupta

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test whether this bug is reproducible by doing the following:

- Take down the MySQL daemon (with `mysqladmin shutdown`).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).
- Check all tables with `myisamchk -s database/*.MYI`. Repair any wrong tables with `myisamchk -r database/table.MYI`.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start `mysqld` with `--log-bin`. See Sección 5.10.3, “El registro binario (Binary Log)”. If you want to find a query that crashes `mysqld`, you should use `--log --log-bin`.
- When you have gotten a crashed table, stop the `mysqld server`.
- Restore the backup.
- Restart the `mysqld` server **without** `--log-bin`
- Re-execute the commands with `mysqlbinlog update-log-file | mysql`. The update log is saved in the MySQL database directory with the name `hostname-bin.#`.
- If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to ftp://ftp.mysql.com/pub/mysql/upload/ and enter it into our bugs system at http://bugs.mysql.com/. If you are a support customer, you can use the MySQL Customer Support Center https://support.mysql.com/ 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.

D.2. Depuración de un cliente MySQL

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 Sección 2.8.2, “Opciones típicas de `configure`”.

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 Sección 1.6.1.3, “Cómo informar de bugs y problemas”.

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.

D.3. El paquete DBUG

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 debugging. See Sección D.1.2, “Crear ficheros de traza”.

This section summaries 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 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 for MySQL 5.0 to get the most updated DBUG manual.

You use the debug package by invoking a program with the `--debug="..."` or the `-#...` option.

Most MySQL programs have a default debug string that is used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/program_name.trace` on Unix and `\program_name.trace` on Windows.

The debug control string is a sequence of colon-separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional '`,`' and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```

The currently recognized flag characters are:

Flag	Description
`d`	Enable output from DBUG_<N> macros for the current state. May be followed by a list of keywords which selects output only for the DBUG macros with that keyword. An empty list of keywords implies output for all macros.
`D`	Delay after each debugger output line. The argument is the number of tenths of seconds to delay, subject to machine capabilities. For example, `-#D,20` specifies a delay of two seconds.
`f`	Limit debugging and/or tracing, and profiling to the list of named functions. Note that a null list disables 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.
g	Enable profiling. Create a file called dbugmon.out containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered.
L	Identify the source file line number for each line of debug or trace output.
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.

Some examples of debug control strings that might appear on a shell command line (the -# is typically used to introduce a control string to an application program) are:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:O,\\mysqld.trace
```

In MySQL, common tags to print (with the d option) are enter, exit, error, warning, info, and loop.

D.4. Comentarios sobre subprocesos RTS

I have tried to use the RTS thread packages with MySQL but stumbled on the following problems:

They use old versions of many POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are currently written. See mysys/my_pthread.c for more info.

At least the following should be changed:

pthread_get_specific should use one argument. sigwait should take two arguments. A lot of functions (at least pthread_cond_wait, pthread_cond_timedwait()) should return the error code on error. Now they return -1 and set errno.

Another problem is that user-level threads use the ALRM signal and this aborts a lot of functions (read, write, open...). MySQL should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread-level alarms I changed `mysys/thr_alarm.c` to wait between alarms with `pthread_cond_timedwait()`, but this aborts with error `EINTR`. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try MySQL with RTS threads I suggest the following:

- Change functions MySQL uses from the thread library to POSIX. This shouldn't take that long.
- Compile all libraries with the `-DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be fixed by changing `my_pthread.h` and `my_pthread.c`.
- Run `thr_alarm`. If it runs without any “warning,” “error,” or aborted messages, you are on the right track. Here is a successful run on Solaris:

```
Main thread: 1
Thread 0 (5) started
Thread: 5  Waiting
process_alarm
Thread 1 (6) started
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 1 (1) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 2 (2) sec
Thread: 6  Simulation of no alarm needed
Thread: 6  Slept for 0 (3) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 4 (4) sec
Thread: 6  Waiting
process_alarm
thread_alarm
Thread: 5  Slept for 10 (10) sec
Thread: 5  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 5 (5) sec
Thread: 6  Waiting
process_alarm
process_alarm

...
thread_alarm
Thread: 5  Slept for 0 (1) sec
end
```

D.5. Diferencias entre paquetes de control de subprocesos

MySQL is very dependent on the thread package used. So when choosing a good platform for MySQL, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library manages all non-thread-safe functions with locks. Read, write and select operations are usually managed with a thread-specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT-pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread-safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT-pthreads and sockets). Thread scheduling isn't always optimal.

- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this method. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left hanging and you must kill them all before restarting. Thread switching is somewhat expensive.

- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, `ps` may show the different threads. If one thread aborts, the whole process aborts. Most system calls are thread-safe and should require very little overhead. Solaris, HP-UX, AIX and OSF/1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really “thread aware.”

Apéndice E. Variables de entorno

Este apéndice lista todas las variables de entorno que se usan directa o indirectamente en MySQL. La mayoría de ellas pueden encontrarse en otros apartados de este manual.

Tenga en cuenta que cualquier opción de la línea de comandos tiene precedencia sobre los valores especificados en los ficheros de opciones y variables de entorno, y los valores en los ficheros de opciones tienen precedencia sobre los valores de las variables de entorno.

En muchos casos, es preferible usar un fichero de opciones en lugar de variables de entorno para modificar el comportamiento de MySQL. Consulte Sección 4.3.2, “Usar ficheros de opciones”.

Variable	Descripción
`CXX`	Nombre del compilador C++ (para ejecutar `configure`).
`CC`	Nombre del compilador C (para ejecutar `configure`).
`CFLAGS`	Flags para el compilador C (para ejecutar `configure`).
`CXXFLAGS`	Flags para el compilador C++ (para ejecutar `configure`).
`DBI_USER`	Nombre de usuario por defecto para Perl DBI.
`DBI_TRACE`	Opciones de traceo para Perl DBI.
`HOME`	La ruta por defecto para el fichero histórico `mysql` es `$HOME/.mysql_history`.
`LD_RUN_PATH`	Usado para especificar dónde está localizado `libmysqlclient.so`.
`MYSQL_DEBUG`	Opciones de traceo de depuración durante la depuración.
`MYSQL_HISTFILE`	La ruta al fichero histórico `mysql` . Si esta variable está inicializada, su valor sobreescribe el valor por defecto de `$HOME/.mysql_history`.
`MYSQL_HOST`	Nombre de equipo usado por defecto por el cliente de línea de comandos `mysql` .
`MYSQL_PS1`	El prompt usado por el cliente de línea de comandos `mysql` .
`MYSQL_PWD`	La contraseña por defecto al conectar a `mysqld`. Tenga en cuenta que su uso es inseguro! Consulte Sección 5.7.6, “Guardar una contraseña de forma segura”.
`MYSQL_TCP_PORT`	Puerto TCP/IP por defecto.
`MYSQL_UNIX_PORT`	Nombre de fichero socket Unix por defecto; usado para conexiones con el `localhost`.
`PATH`	Usado por la shell para encontrar programas MySQL.
`TMPDIR`	Directorio donde se crean los ficheros temporales.
`TZ`	Debe tener el valor de la zona horaria local. Consulte Sección A.4.6, “Problemas con las franjas horarias”.
`UMASK_DIR`	La máscara de creación de directorios del usuario. Tenga en cuenta que se hace una operación Y lógica con `UMASK`!
`UMASK`	La máscara de creación de ficheros del usuario.
`USER`	Nombre de usuario por defecto para Windows y NetWare usado al concetar con `mysqld`.

Apéndice F. Expresiones regulares en MySQL

Las expresiones regulares permiten especificar un patrón para una búsqueda compleja.

MySQL usa la implementación de expresiones regulares de Henry Spencer, que es conforme con POSIX 1003.2. Ver Apéndice B, *Credits*. MySQL usa la versión extendida para soportar operaciones de concordancia de patrones, a través del operador `REGEXP` en sentencias SQL. Ver Sección 3.3.4.7, “Coincidencia de patrones”.

Este apéndice es un resumen, con ejemplos, de caracteres especiales y constructores que se pueden utilizar en MySQL para las operaciones `REGEXP`. No contiene todos los detalles que se pueden encontrar en la página del manual de `regex(7)` de Henry Spencer. Dicho manual se incluye en el código fuente de MySQL, en el archivo `regex.7` dentro del directorio `regex`.

Una expresión regular describe una serie de cadenas. La expresión regular más simple es aquella que no tiene caracteres especiales en su interior. Por ejemplo, la expresión regular `hola` concuerda con `hola` y nada más.

Una expresión regular no trivial usa ciertos constructores especiales, que pueden concordar más de en una cadena. Por ejemplo, la expresión regular `hello|word` concuerda tanto con la cadena `hello` como con la cadena `word`.

Un ejemplo más complicado: la expresión regular `B[an]*s` concuerda con las cadenas `Bananas`, `Baaaaas`, `Bs`, y cualquier otra cadena que comience con una `B`, termine con una`s` y contenga cualquier número de caracteres `a` o `n` entre ellas.

Una expresión regular para el operador `REGEXP` puede utilizar cualquiera de los siguientes caracteres especiales y constructores:

- `^`

  que concuerda con el inicio de la cadena.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';                  -> 0
mysql> SELECT 'fofo' REGEXP '^fo';                     -> 1
```

- `$`

  que concuerda con el final de la cadena.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';                -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';                   -> 0
```

- `.`

  que concuerda con cualquier carácter (incluyendo el retorno de carro y la nueva línea).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';                   -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';               -> 1
```

- `a*`

  que concuerda con cualquier secuencia de cero o más caracteres `a`.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';                    -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';                  -> 1
```

```
mysql> SELECT 'Bn' REGEXP '^Ba*n';                    -> 1
```

- `a+`

  que concuerda con cualquier secuencia de uno o más caracteres `a`.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';                   -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';                    -> 0
```

- `a?`

  que concuerda con cero o un carácter `a`.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';                    -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';                   -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';                  -> 0
```

- `de|abc`

  que concuerda tanto con la secuencia `de` como con `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';                   -> 1
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)*`

  que concuerda con cero o más instancias de la secuencia `abc`.

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';                  -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';                 -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';                -> 1
```

- `{1}`, `{2,3}`

  La notación `{n}` o `{m,n}` provee una manera más general de escribir expresiones regulares que concuerden con varias instancias del átomo anterior (o “pieza”) del patrón. `m` y `n` son enteros.

  - `a*`

    Puede escribirse como `a{0,}`.

  - `a+`

    Puede escribirse como `a{1,}`.

  - `a?`

    Puede escribirse como `a{0,1}`.

  Para ser más preciso, `a{n}` concuerda exactamente con `n` instancias de `a`. `a{n,}` concuerda con `n` o más instancias de `a`. `a{m,n}` concuerda con entre `m` y `n` (inclusive) instancias de `a`.

  `m` y `n` deben estar en el rango desde `0` hasta `RE_DUP_MAX` (255 por defecto). Si se especifican tanto `m` como `n`, `m` debe ser menor o igual que `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]`

  Concuerda con cualquier caracter que sea (o no sea, si se utiliza ^) uno de los siguientes: `a`, `b`, `c`, `d` o `X`. Un carácter `-` entre dos caracteres distintos forma un rango que concuerda con cualquier carácter entre el primero y el segundo. Por ejemplo, `[0-9]` concuerda con cualquier dígito decimal. Para incluir un carácter `]` literal, debe estar precedido inmediatamente por un corchete abierto `[`. Para incluir un carácter `-` literal , debe estar al principio o al final. Cualquier carácter que no tenga un significado especial, concuerda sólo consigo mismo si está dentro de un par de `[]`.

```
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.]`

  Dentro de una expresión de corchetes (escrita usando `[` y `]`), concuerda con la sencuencia de caracteres expresada. `characters` es cualquier carácter simple o su nombre de carácter como `newline`. Se puede encontrar la lista completa de nombres de caracteres en el archivo`regexp/cname.h`.

```
mysql> SELECT '~' REGEXP '[[.~.]]';                     -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';                 -> 1
```

- `[=character_class=]`

  Dentro de una expresión de corchetes (escrita usando `[` y `]`), `[=character_class=]` representa una clase de equivalencia. Concuerda con cualquier carácter con el mismo valor de comparación. Por ejemplo, si `o` y `(+)` son miembros de una clase equivalente, entonces `[[=o=]]`, `[[=(+)=]]`, y `[o(+)]` son todos sinónimos. Una clase equivalente no puede utilizarse como el final de un rango.

- `[:character_class:]`

  Dentro de una expresión de corchetes (escrita usando `[` y `]`), `[:character_class:]` representa una clase de caracteres que concuerda con todos los caracteres pertenecientes a esa clase. Los nombres de clases estándar son:

`alnum`	Caracteres alfanumericos
`alpha`	Caracteres Alfabéticos
`blank`	Caracteres espacios en blanco
`cntrl`	Caracteres de Control
`digit`	Caracteres Dígitos
`graph`	Caracteres gráficos
`lower`	Caracteres alfabéticos en minúsculas
`print`	Caracteres gráficos o espacios

`punct`	Caracteres de puntación
`space`	Espacio, tabulador, nueva línea y retorno de carro
`upper`	Caracteres alfabéticos en mayúsculas
`xdigit`	Caracteres dígitos en hexadecimal

Estos nombres representan las clases de caracteres definidas en `ctype(3)`. Una localización particular pudiera proveer otros nombres de clases. Una clase de caracteres no se puede utilizar como final de un rango.

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1
mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- `[[:<:]]`, `[[:>:]]`

  Estos marcadores representan limitadores de palabras. Concuerdan con el inicio y el final de palabras, respectivamente. Una palabra es una secuencia de caracteres de palabra que no son precedidos ni seguidos por caracteres de palabra. Un carácter de palabra es un carácter alfanumerico en la clase `alnum` o un guión bajo (_).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

Para utilizar en una expresión regular un carácter especial literalmente, precédalo con dos barras invertidas (\). El parser de MySQL interpreta las barras invertidas, y la biblioteca de expresiones regulares interpreta el resto. Por ejemplo, para concordar con la cadena `1+2`, que contiene el carácter especial `+`, sólo la última de las siguientes expresiones regulares es correcta:

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

Apéndice G. Límites en MySQL

Tabla de contenidos

This appendix lists current limits in MySQL.

G.1. Límites de los joins

En MySQL 5.0, el numero máximo de tablas que pueden ser referenciadas en un único join es de 61. Esto también se aplica al número de tablas que pueden ser referenciadas en la definición de una Vista

Apéndice H. Restricciones en características de MySQL

Tabla de contenidos

Se expone a continuación las restricciones en el uso de algunas características de MySQL, como las subconsultas o las vistas.

H.1. Restricciones en procedimientos almacenados y disparadores

Algunas de las restricciones expuestas aquí se refieren a todas las rutinas almacenadas; esto es, procedimientos almacenados y funciones almacenadas. Algunas de las restricciones sólo se refieren a funciones almacenadas, y no a los procedimientos almacenados.

Todas las restricciones para las funciones almacenadas se refieren también a los disparadores(triggers).

Las rutinas almacenadas no pueden contener sentencias SQL arbitrarias. Las siguientes sentencias no están permitidas dentro de una rutina almacenada:

- `CHECK TABLES`
- `LOCK TABLES`, `UNLOCK TABLES`
- `FLUSH`
- `LOAD DATA`, `LOAD TABLE`
- Sentencias SQL preparadas(`PREPARE`, `EXECUTE`, `DEALLOCATE`). Implicación: No puedo usar SQL dinámico dentro de una rutina almacenada(donde construya dinámicamente sentencias como cadenas de caracteres y después ejecutarlas).

Además, en funciones almacenadas (pero no para procedimientos almacenados), no están permitidas las siguientes sentencias:

- Sentencias que hacen commits o rollbacks explícitos o implícitos.
- Sentencias que devuelvan un resultado. Esto incluye sentencias `SELECT` que no tienen una cláusula `INTO` y la sentencia `SHOW`. Una función puede procesar un resultado tanto con `SELECT … INTO` como con el uso de un cursor y de la sentencia `FETCH`.

El uso de una rutina almacenada puede causar problemas de replicación. Este asunto se expone con profundidad en Sección 19.3, “Registro binario de procedimientos almacenados y disparadores”.

`INFORMATION_SCHEMA` todavía no tiene una tabla `PARAMETERS`, así que aplicaciones que, en tiempo de ejecución, necesiten adquirir información de los parámetros de la rutina, deben usar técnicas como tratar la salida de la sentencia `SHOW CREATE`.

No existen facilidades para el depurado de rutinas almacenadas.

Las rutinas almacenadas usan cursores materializados, no cursores nativos. (El resultado se genera y guarda en el lado del servidor, y después se devuelve línea por línea a medida que el cliente lo extrae.)

La sentencia `CALL` no se puede preparar. Esto es cierto tanto para las sentencias preparadas del lado del servidor como para las sentencias SQL preparadas.

H.2. Restricciones en cursores del lado del servidor

Los cursores del lado del servidor están implementados a partir de MySQL 5.0.2 a través de la función de la API de C `mysql_stmt_attr_set()`. Un cursor del lado del servidor permite a un resultado ser generado del lado del servidor, pero no transmitido al cliente excepto para aquellos registros que el cliente solicite. Por ejemplo, si un cliente ejecuta una consulta pero de ésta sólo le interesa el primer registro, los registros sobrantes no son transferidos.

Los cursores son de sólo lectura; no puede usar un cursor para actualizar registros.

`UPDATE WHERE CURRENT OF` y `DELETE WHERE CURRENT OF` no están implementadas, porque los cursores actualizables no están soportados.

Los cursores son no-mantenibles (no se mantienen abiertos después de una ejecución (commit)). Cursors are non-holdable (not held open after a commit).

Los cursores son insensibles.

Los cursores no pueden navegarse.

Los cursores no son nombrados. El proceso de la sentencia actua como el ID del cursor.

Puede tener sólo un cursor por sentencia preparada. Si necesita varios cursores, debe preparar varias sentencias.

No se puede utilizar un cursor para una sentencia que genera un resultado si la sentencia no es soportada en modo preparado. Esto incluye sentencias como `CHECK TABLES`, `HANDLER READ` y `SHOW BINLOG EVENTS`.

H.3. Restricciones en subconsultas

El siguiente bug es conocido y será reparado: Si se compara un valor `NULL` con una subconsulta usando `ALL`, `ANY`, o `SOME`, y la sentencia devuelve un resultado vacío, la comparación podría dar un resultado no estándar de `NULL` en vez de un `TRUE` o `FALSE`.

La sentencia de afuera de la subconsulta puede ser cualquiera de las siguientes: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, o `DO`.

Las operaciones de comparación de registros son soportadas sólo parcialmente:

- En *`expr IN (subconsulta)`*, *`expr`* puede ser una *n*-tupla (especificada vía sintaxis del constructor de registros) y la subconsulta puede devolver registros de *n*-tuplas.
- En *`expr op {ALL|ANY|SOME} (subconsulta)`*, *`expr`* debe ser un valor escalar y la subconsulta debe ser de una sola columna; no puede devolver registros con múltiples columnas.

En otras palabras, para una subconsulta que devuelve registros de *n*-tuplas, esto está soportado:

```
(val_1, ..., val_n) IN (subconsulta)
```

Pero esto no está soportado:

```
(val_1, ..., val_n) op {ALL|ANY|SOME} (subconsulta)
```

La razón por la que se soporta la comparación entre registros con `IN` pero no con los otros es que `IN` fue implementado reescribiéndolo como una secuencia de comparaciones `=` y operaciones `AND`. Esto mismo no puede realizarse con `ALL`, `ANY`, o `SOME`.

Los constructores de registros no están bien optimizados. Las siguientes dos expresiones son equivalentes, pero sólo la segunda puede ser optimizada:

```
(col1, col2, ...) = (val1, val2, ...)
col1 = val1 AND col2 = val2 AND ...
```

La optimización de subconsultas para los `IN` no es tan efectiva como para los `=`.

Un caso típico del pobre rendimiento del `IN` es cuando una subconsulta devuelve un número pequeño de registros, pero la consulta de afuera regresa un número grande de registros para ser comparados con el resultado de la subconsulta.

Las subconsultas en la cláusula `FROM` no pueden ser subconsultas correlacionadas. Éstas son materializadas (ejecutadas para producir un resultado) antes de que se evalúe la consulta exterior, así que no pueden ser evaluadas por registro de la consulta exterior.

En general, no puede modificar una tabla y seleccionar de la misma en una subconsulta. Por ejemplo, esta limitación se aplica a sentencias del siguiente tipo:

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

Excepción: La prohibición precedente no se aplica si se usa una subconsulta para la tabla modificada en la cláusula `FROM`. Ejemplo:

```
UPDATE t ... WHERE col = (SELECT (SELECT ... FROM t...) AS _t ...);
```

Aquí la prohibición no se aplica porque una subconsulta en la cláusula `FROM` se materializa como una tabla temporal, así que los registros relevantes en `t` ya han sido seleccionados cuando la actualización de `t` ha tenido lugar.

El optimizador es más maduro para joins que para subconsultas, así que en la mayor parte de los casos las sentencias que usan subconsultas pueden ser ejecutadas más eficientemente si se reescriben como joins.

Una excepción a esta norma es el caso en que una subconsulta `IN` puede ser reescrita como una join `SELECT DISTINCT`. Ejemplo:

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condición);
```

La sentencia puede ser reescrita como sigue:

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condición;
```

Pero en este caso el join requiere una operación `DISTINCT` extra y no es más eficiente que la subconsulta.

Futura optimización posible: MySQL no reescribe el orden del join para la evaluación de subconsultas. En algunos casos, una subconsulta puede ser ejecutada más eficientemente si MySQL la reescribe como un

join. Esto daría al optimizador mayor posibilidad de elegir entre varios planes de ejecución. Por ejemplo, podría decidir si leer primero una tabla o la otra.

Ejemplo:

```
SELECT a FROM outer_table AS ot
WHERE a IN (SELECT a FROM inner_table AS it WHERE ot.b = it.b);
```

Para esra consulta, MySQL siempre busca `outer_table` primero y después ejecuta la subconsulta en `inner_table` para cada registro. Si `outer_table` tiene muchos registros e `inner_table` tiene pocos, la consulta probablemente no será tan rápida como pudiera ser.

La consulta anterior podría reescribirse así:

```
SELECT a FROM outer_table AS ot, inner_table AS it
WHERE ot.a = it.a AND ot.b = it.b;
```

En este caso podemos explorar la tabla pequeña (`inner_table`) y buscar registros en `outer_table`, lo cual sería más rápido si existiera un índice en `(ot.a,ot.b)`.

Futura optimización posible: Una subconsulta correlacionada se evalúa por cada registro de la consulta externa. Una mejor solución sería que no se evaluara la subconsulta nuevamente si el valor del registro exterior fuese igual al de la línea anterior. Se usaría en este caso el resultado previo.

Futura optimización posible: Una subconsulta en la cláusula `FROM` se evalúa materializando el resultado dentro de una tabla temporal, y esta tabla no usa índices. Esto no permite el uso de índices en comparación con otras tablas en la consulta, aunque pudiera ser útil.

Futura optimización posible: Si una subconsulta en la cláusula `FROM` se asemeja a una vista a la que se puede aplicar el algoritmo `MERGE`, reescribir la consulta, aplicar el algoritmo `MERGE`, para que se puedan utilizar los índices. La siguiente sentencia contiene una subconsulta de este tipo:

```
SELECT * FROM (SELECT * FROM t1 WHERE t1.t1_col) AS _t1, t2 WHERE t2.t2_col;
```

La sentencia puede ser reescrita con un join como éste:

```
SELECT * FROM t1, t2 WHERE t1.t1_col AND t2.t2_col;
```

Este tipo de reescritura comportaría dos beneficios

1. Evitar el uso de una tabla temporal, en la que no se puede utilizar índices. En la consulta reescrita, el optimizador puede usar índices en `t1`.
2. Da al optimizador más libertad al elegir entre diferentes planes de ejecución. Por ejemplo, reescribiendo la consulta como un join permite al optimizador usar `t1` o `t2` primero.

Futura optimización posible: Para los `IN`, `= ANY`, `<> ANY`, `= ALL`, y `<> ALL` con subconsultas no correlacionadas, usar una tabla hash en memoria para un resultado o una tabla temporal con un índice para los resultados más grandes. Ejemplo:

```
SELECT a FROM big_table AS bt
WHERE non_key_field IN (SELECT non_key_field FROM table WHERE condicion)
```

En este caso, podríamos crear una tabla temporal:

```
CREATE TABLE t (key (non_key_field))
(SELECT non_key_field FROM table WHERE condicion)
```

Entonces, para cada renglón en `big_table`, hacer un ciclo en `t` basado en `bt.non_key_field`.

H.4. Restricciones en vistas

El procesamiento de vistas no está optimizado:

- No es posible crear un ínidice en una vista.
- Los índices pueden utilizarse para procesar vistas usando un algoritmo de combinación (MERGE). Sin embargo, una vista que se procesa con el algoritmo de tablas temporales (temptable) no es capaz de tomar ventaja de los índices que hacen referencia a las tablas que contiene (aunque los índices pueden ser usados durante la generación de las tablas temporales).

Las subconsultas no pueden utilizarse en la cláusula `FROM` de una vista. Esta limitación será removida en el futuro.

Existe un principio general por el que no se puede modificar una tabla y seleccionar de la misma en una subconsulta. Ver Sección H.3, “Restricciones en subconsultas”.

El mismo principio se aplica también si se hace una selección de una vista que hace una selección de una tabla, si la vista selecciona de la tabla dentro de una subconsulta, y la vista es evaluada usando el algoritmo de combinación (merge). Ejemplo:

```
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;
```

Si la vista se evalúa usando una tabla temporal, se *puede* seleccionar de la tabla en la subconsulta de la vista y modificarla en la consulta exterior. En este caso la vista se almacenará en una tabla temporal y por ello no se está realmente seleccionando de una tabla en una subconsulta y modificándola “al mismo tiempo”. (Esta es otra razón por la que tal vez sea deseable forzar a MySQL para que use el algoritmo de tablas (temptable) temporales especificando las palabras `ALGORITHM = TEMPTABLE` en la definición de la vista.)

Se puede usar `DROP TABLE` o `ALTER TABLE` para eliminar o modificar una tabla utilizada en la definición de una vista (lo cual invalida la vista) y no se obtendrá ninguna alerta de las operaciones de elimiar o modificar. Sin embargo, se obtiene un error más tarde, cuando se utiliza la vista.

Algunas sentencias “congelan” una definición de vista:

- Si una sentencia preparada por `PREPARE` se refiere a una vista, los contenidos de la vista que se ven cada vez que se ejecuta la sentencia, serán los contenidos de la vista en el momento en el que la sentencia fue preparada. Esto es cierto incluso si la definición de la vista se cambia después de preparar la sentencia y antes de que ésta se ejecute. Ejemplo:

  ```
  CREATE VIEW v AS SELECT 1;
  PREPARE s FROM 'SELECT * FROM v';
  ALTER VIEW v AS SELECT 2;
  EXECUTE s;
  ```

  El resultado devuelto por la sentencia `EXECUTE` es 1, y no 2.

- Si una sentencia en una rutina almacenada se refiere a una vista, los contenidos de la vista que ve la sentencia son sus contenidos de la primera ejecución de la sentencia. Esto significa por ejemplo que si se ejecuta una sentencia en un bucle, en todas las iteraciones de la sentencia se verá el mismo contenido de la vista, aunque la definición de la vista cambie durante el bucle. Ejemplo:

```
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();
```

  Cuando se llama al procedimiento p(), el SELECT devuelve siempre 1 dentro del bucle, aunque dentro del mismo cambie la definición de la vista.

Con respecto a las actualizaciones en vistas, el objetivo es que cualquier vista que sea teóricamente actualizable, tiene que serlo en la práctica. Esto incluye vistas que tienen `UNION` en su definición. Actualmente, no todas las vistas teóricamente actualizables se pueden actualizar. La implementación inicial de vistas fue deliberadamente escrita de esta forma para obtener en MySQL vistas utilizables y actualizables lo antes posible. Muchas vistas teóricamente actualizables pueden ser actualizadas actualmente, pero algunas limitaciones siguen existiendo:

- Las vistas actualizables con subconsultas en cualquier lugar que no sea en la cláusula `WHERE`. Algunas vistas que tienen subconsultas en la lista `SELECT` podrían ser actualizables.
- No se puede utilizar `UPDATE` para actualizar más de una tabla incluida en una vista que sea definida como un join.
- No se puede usar una sentencia `DELETE` para actualizar una vista que está definida como un JOIN.

Apéndice I. GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software---to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The ``Program'', below, refers to any such program or work, and a ``work based on the Program'' means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a

portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ``modification''.) Each licensee is addressed as ``you''.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third-party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ``any later version'', you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

    NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ``AS IS'' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ``copyright'' line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items---whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a ``copyright disclaimer'' for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Apéndice J. MySQL FLOSS License Exception

Version 0.3, 10 February 2005

The MySQL AB Exception for Free/Libre and Open Source Software-only Applications Using MySQL Client Libraries (the ``FLOSS Exception'').

Exception Intent

We want specified Free/Libre and Open Source Software (``FLOSS'') applications to be able to use specified GPL-licensed MySQL client libraries (the ``Program'') despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the ``GPL'').

Legal Terms and Conditions

As a special exception to the terms and conditions of version 2.0 of the GPL:

1. You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a ``FLOSS Work'') licensed under one or more of the licenses listed below in section 1, as long as:

   a. You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,

   b. all identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,

      - i

        are distributed subject to one of the FLOSS licenses listed below, and

      - ii

        the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections, and

   c. any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

   If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL or another valid licensing option from MySQL AB.

2. FLOSS License List

License name	Version(s)/Copyright Date
Academic Free License	2.0
Apache Software License	1.0/1.1/2.0
Apple Public Source License	2.0
Artistic license	From Perl 5.8.0

BSD license	"July 22 1999"
Common Public License	1.0
GNU Library or "Lesser" General Public License (LGPL)	2.0/2.1
Jabber Open Source License	1.0
MIT license	-
Mozilla Public License (MPL)	1.0/1.1
Open Software License	2.0
OpenSSL license (with original SSLeay license)	"2003" ("1998")
PHP License	3.0
Python license (CNRI Python License)	-
Python Software Foundation License	2.1.1
Sleepycat License	"1999"
W3C License	"2001"
X11 License	"2001"
Zlib/libpng License	-
Zope Public License	2.0

Due to the many variants of some of the above licenses, we require that any version follow the 2003 version of the Free Software Foundation's Free Software Definition (http://www.gnu.org/philosophy/free-sw.html) or version 1.9 of the Open Source Definition by the Open Source Initiative (http://www.opensource.org/docs/definition.php).

3. Definitions

   a. Terms used, but not defined, herein shall have the meaning provided in the GPL.

   b. Derivative Work means a derivative work under copyright law.

4. Applicability This FLOSS Exception applies to all Programs that contain a notice placed by MySQL AB saying that the Program may be distributed under the terms of this FLOSS Exception. If you create or distribute a work which is a Derivative Work of both the Program and any other work licensed under the GPL, then this FLOSS Exception is not available for that work; thus, you must remove the FLOSS Exception notice from that work and comply with the GPL in all respects, including by retaining all GPL notices. You may choose to redistribute a copy of the Program exclusively under the terms of the GPL by removing the FLOSS Exception notice from that copy of the Program, provided that the copy has never been modified by you or any third party.

Índice

Símbolos

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y