

**Oracle® OLAP**  
Expression Syntax Reference  
12c Release 1 (12.1)  
**E23382-06**

June 2014

Oracle OLAP Expression Syntax Reference, 12c Release 1 (12.1)

E23382-06

Copyright © 2006, 2014, Oracle and/or its affiliates. All rights reserved.

Primary Author: David McDermid

Contributor: The Oracle Database 12c documentation is dedicated to Mark Townsend, who was an inspiration to all who worked on this release.

Contributor: Donna Carver, David Greenfield, Anne Murphy, Martin Roth, Fuad Sheehab

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

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

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

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

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

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

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

---

---

# Contents

<b>Preface</b> .....	vii
<b>1 Basic Elements</b>	
Dimensional Object Names .....	1-1
Dimensional Data Types .....	1-2
Operators.....	1-3
Conditions .....	1-6
Literal Expressions .....	1-10
CASE Expressions .....	1-10
Qualified Data References (QDRs) .....	1-13
<b>2 OLAP Functions</b>	
OLAP Functions in Alphabetical Order .....	2-1
OLAP Functions By Category .....	2-1
AVERAGE_RANK.....	2-4
AVG .....	2-6
COUNT.....	2-9
DENSE_RANK .....	2-12
HIER_ANCESTOR.....	2-14
HIER_CHILD_COUNT.....	2-16
HIER_DEPTH .....	2-18
HIER_LEVEL.....	2-20
HIER_ORDER.....	2-22
HIER_PARENT.....	2-24
HIER_TOP .....	2-26
LAG .....	2-28
LAG_VARIANCE.....	2-30
LAG_VARIANCE_PERCENT .....	2-32
LEAD.....	2-34
LEAD_VARIANCE.....	2-37
LEAD_VARIANCE_PERCENT .....	2-40
MAX .....	2-43
MIN.....	2-46
OLAP_DML_EXPRESSION.....	2-49
RANK .....	2-50

ROW_NUMBER .....	2-52
SHARE .....	2-54
SUM .....	2-56

### 3 Row Functions

Row Functions in Alphabetical Order .....	3-1
Row Functions By Category .....	3-4
ABS .....	3-8
ACOS .....	3-9
ADD_MONTHS .....	3-10
ASCII .....	3-11
ASCIISTR .....	3-12
ASIN .....	3-13
ATAN .....	3-14
ATAN2 .....	3-15
BIN_TO_NUM .....	3-16
BITAND .....	3-17
CAST .....	3-18
CEIL .....	3-19
CHARTOROWID .....	3-20
CHR .....	3-21
COALESCE .....	3-22
CONCAT .....	3-23
COS .....	3-24
COSH .....	3-25
CURRENT_DATE .....	3-26
CURRENT_TIMESTAMP .....	3-27
DBTIMEZONE .....	3-28
DECODE .....	3-29
EXP .....	3-30
EXTRACT (datetime) .....	3-31
FLOOR .....	3-32
FROM_TZ .....	3-33
GREATEST .....	3-34
HEXTORAW .....	3-35
INITCAP .....	3-36
INSTR .....	3-37
LAST_DAY .....	3-38
LEAST .....	3-39
LENGTH .....	3-40
LN .....	3-41
LNNVL .....	3-42
LOCALTIMESTAMP .....	3-43
LOG .....	3-44
LOWER .....	3-45
LPAD .....	3-46
LTRIM .....	3-47

MOD .....	3-48
MONTHS_BETWEEN.....	3-49
NANVL.....	3-50
NEW_TIME .....	3-51
NEXT_DAY.....	3-53
NLS_CHARSET_ID.....	3-54
NLS_CHARSET_NAME.....	3-55
NLS_INITCAP .....	3-56
NLS_LOWER.....	3-57
NLS_UPPER .....	3-58
NLSSORT .....	3-59
NULLIF.....	3-60
NUMTODSINTERVAL .....	3-61
NUMTOYMINTERVAL .....	3-62
NVL.....	3-63
NVL2.....	3-64
ORA_HASH .....	3-65
POWER.....	3-66
RAWTOHEX .....	3-67
REGEXP_COUNT .....	3-68
REGEXP_INSTR.....	3-69
REGEXP_REPLACE.....	3-71
REGEXP_SUBSTR .....	3-73
REMAINDER.....	3-75
REPLACE .....	3-76
ROUND (date) .....	3-77
ROUND (number).....	3-78
ROWIDTOCHAR.....	3-79
ROWIDTONCHAR .....	3-80
RPAD .....	3-81
RTRIM.....	3-82
SESSIONTIMEZONE .....	3-83
SIGN.....	3-84
SIN.....	3-85
SINH .....	3-86
SOUNDEX .....	3-87
SQRT.....	3-88
SUBSTR.....	3-89
SYS_CONTEXT .....	3-90
SYSDATE.....	3-92
SYSTIMESTAMP .....	3-93
TAN.....	3-94
TANH .....	3-95
TO_BINARY_DOUBLE .....	3-96
TO_BINARY_FLOAT .....	3-97
TO_CHAR (character).....	3-98
TO_CHAR (datetime).....	3-99

TO_CHAR (number).....	3-100
TO_DATE .....	3-101
TO_DSINTERVAL.....	3-102
TO_NCHAR (character) .....	3-103
TO_NCHAR (datetime).....	3-104
TO_NCHAR (number) .....	3-105
TO_NUMBER .....	3-106
TO_TIMESTAMP.....	3-107
TO_TIMESTAMP_TZ .....	3-108
TO_YMINTERVAL.....	3-109
TRANSLATE.....	3-110
TRANSLATE (USING) .....	3-111
TRIM.....	3-112
TRUNC (number).....	3-113
TZ_OFFSET .....	3-114
UID.....	3-115
UNISTR.....	3-116
UPPER.....	3-117
USER.....	3-118
VSIZE.....	3-119
WIDTH_BUCKET .....	3-120

## A Reserved Words

Reserved Words.....	A-1
Special Symbols.....	A-4

## Index

---

---

# Preface

The OLAP expression syntax includes analytic functions, arithmetic operators, and single-row functions. The OLAP syntax is an extension of the SQL syntax. If you have used SQL analytic functions or single-row functions, then this syntax is familiar to you.

This preface contains these topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)
- [Backus-Naur Form Syntax](#)

## Audience

This document is intended for anyone who wants to create calculated measures or transform the data stored in relational tables for use in dimensional database objects such as cubes, cube dimensions, and measures.

## Documentation Accessibility

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

### Access to Oracle Support

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

## Related Documents

For more information, see these documents in the Oracle Database 11.2 documentation set:

- *Oracle OLAP User's Guide*  
Explains how SQL applications can extend their analytic processing capabilities and manage summary data by using the OLAP option of Oracle Database.

- *Oracle OLAP Customizing Analytic Workspace Manager*  
Describes how to customize Analytic Workspace Manager with XML documents and Java plug-ins.
- *Oracle OLAP DML Reference*  
Contains a complete description of the OLAP Data Manipulation Language (OLAP DML), which is used to define and manipulate analytic workspace objects.
- *Oracle Database Reference*  
Contains full descriptions of the data dictionary views for cubes, cube dimensions, and other dimensional objects.
- *Oracle Database PL/SQL Packages and Types Reference*  
Contains full descriptions of several PL/SQL packages for managing cubes.
- *Oracle OLAP Java API Developer's Guide*  
Introduces the Oracle OLAP API, a Java application programming interface for Oracle OLAP, which is used for defining, building, and querying dimensional objects in the database.
- *Oracle OLAP Java API Reference*  
Describes the classes and methods in the Oracle OLAP Java API for defining, building, and querying dimensional objects in the database.

## Conventions

These text conventions are used in this document:

Convention	Meaning
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

## Backus-Naur Form Syntax

The syntax in this reference is presented in a simple variation of Backus-Naur Form (BNF) that uses the following symbols and conventions:

Symbol or Convention	Description
[ ]	Brackets enclose optional items.
{ }	Braces enclose a choice of items, only one of which is required.
	A vertical bar separates alternatives within brackets or braces.
...	Ellipses indicate that the preceding syntactic element can be repeated.
delimiters	Delimiters other than brackets, braces, and vertical bars must be entered as shown.



---

<b>Symbol or Convention</b>	<b>Description</b>
<b>boldface</b>	Words appearing in boldface are keywords. They must be typed as shown. (Keywords are case-sensitive in some, but not all, operating systems.) Words that are not in boldface are placeholders for which you must substitute a name or value.

---



---



---

## Basic Elements

This chapter describes the basic building blocks of the OLAP expression syntax. It contains these topics:

- [Dimensional Object Names](#)
- [Dimensional Data Types](#)
- [Operators](#)
- [Conditions](#)
- [Literal Expressions](#)
- [CASE Expressions](#)
- [Qualified Data References \(QDRs\)](#)

### Dimensional Object Names

The naming conventions for dimensional objects follow standard Oracle naming rules. All names are case-insensitive.

#### Syntax

```
owner.{ cube | dimension | table }.{ measure | column | attribute }
```

**Table 1–1** Naming Conventions for Dimensional Objects

Convention	Quoted ID	Unquoted ID
<b>Initial Character</b>	Any character.	Alphabetic character from the database character set.
<b>Other Characters</b>	All characters, punctuation marks, and spaces are permitted. Double quotation marks and nulls (\0) are not permitted.	Alphanumeric characters from the database character set and underscore (_) are permitted.  The dollar sign (\$) and pound sign (#) are permitted but not recommended.  Double quotation marks and nulls (\0) are not permitted.
<b>Reserved Words</b>	Permitted but not recommended.	Not permitted.

#### Examples

GLOBAL.UNITS\_CUBE.SALES identifies the SALES measure in the Units Cube.

TIME.DIM\_KEY and TIME.LEVEL\_NAME identify columns in the Time views.

TIME.CALENDAR identifies the CALENDAR hierarchy in the Time dimension.

TIME.CALENDAR.CALENDAR\_YEAR identifies the CALENDAR\_YEAR level of the CALENDAR hierarchy in the Time dimension.

GLOBAL.UNITS\_FACT.MONTH\_ID identifies a foreign key column in the UNITS\_FACT table.

TIME\_DIM.CALENDAR\_YEAR\_DSC identifies a column in the TIME\_DIM table.

## Dimensional Data Types

Table 1–2 describes the data types that can be used for cubes and measures.

**Table 1–2 Dimensional Data Types**

Data Type	Description
BINARY_DOUBLE	A 64-bit floating number. A BINARY_DOUBLE value requires 9 bytes.
BINARY_FLOAT	A 32-bit floating number. A BINARY_FLOAT value requires 5 bytes.
CHAR ( <i>size</i> [BYTE CHAR])	A fixed length character string with a length of <i>size</i> characters or bytes. The size can range from 1 to 2000.
DATE	A valid date in the range from January 1, 4712 BC to December 31, 9999 CE. It contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It does not have fractional seconds or a time zone. The default format is determined explicitly by the NLS_DATE_FORMAT parameter and implicitly by the NLS_TERRITORY parameter. A DATE value requires 7 bytes.
DECIMAL ( <i>p</i> , <i>s</i> )	A decimal number with precision <i>p</i> and scale <i>s</i> represented as a NUMBER data type.
FLOAT [( <i>p</i> )]	A subtype of NUMBER with precision <i>p</i> . A FLOAT is represented internally as NUMBER. The precision can range from 1 to 126 binary digits. A FLOAT value requires from 1 to 22 bytes.
INTEGER	A whole number represented as a NUMBER data type with a scale of 0.
INTERVAL DAY[( <i>day_precision</i> )] TO SECOND[( <i>second_precision</i> )]	A period of time in days, hours, minutes, and seconds. The day precision is the maximum number of digits in the DAY datetime field. The default is 2. The second precision is the number of digits in the fractional part of the SECOND field. The default value is 6. Both day and second precision can have a value from 0 to 9. An INTERVAL DAY TO SECOND value requires 11 bytes.
INTERVAL YEAR[( <i>precision</i> )] TO MONTH	A period of time in years and months. The precision is the number of digits in the YEAR datetime field, which can have a value of 0 to 9. The default precision is 2 digits. An INTERVAL YEAR TO MONTH value requires 5 bytes.
NCHAR[( <i>size</i> )]	A fixed length character string with a length of <i>size</i> characters. The size can range from 1 character to 2000 bytes. The maximum number of characters depends on the national character set, which can require up to four bytes per character.

**Table 1–2 (Cont.) Dimensional Data Types**

<b>Data Type</b>	<b>Description</b>
NUMBER [( <i>p</i> [, <i>s</i> ))]	A decimal number with precision <i>p</i> and scale <i>s</i> . The precision can range from 1 to 38. The scale can range from -84 to 127. A NUMBER value requires from 1 to 22 bytes.
NVARCHAR2( <i>size</i> )	A variable length Unicode character string with a maximum length of <i>size</i> characters. The size can range from 1 character to 32,767 bytes. The maximum number of characters depends on the national character set, which can require up to four bytes per character.
TIMESTAMP[( <i>precision</i> )]	A valid date that contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It does not have a time zone. The precision is the number of digits in the fractional part of the SECOND field, which can have a value of 0 to 9. The default precision is 6 digits. The default format is determined explicitly by the NLS_DATE_FORMAT parameter and implicitly by the NLS_TERRITORY parameter. A TIMESTAMP value requires from 7 to 11 bytes depending on the precision.
TIMESTAMP [( <i>precision</i> )] WITH LOCAL TIME ZONE	A valid date with the same description as TIMESTAMP WITH TIME ZONE with these exceptions: <ul style="list-style-type: none"> <li>■ The data is stored in the database with the database time zone.</li> <li>■ The data is converted to the session time zone when it is retrieved.</li> <li>■ A TIMESTAMP WITH LOCAL TIME ZONE value requires from 7 to 11 bytes depending on the precision.</li> </ul>
TIMESTAMP[( <i>precision</i> )] WITH TIME ZONE	A valid date that contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, and TIMEZONE_MINUTE. The precision is the number of digits in the fractional part of the SECOND field, which can have a value of 0 to 9. The default precision is 6 digits. The default format is determined explicitly by the NLS_DATE_FORMAT parameter and implicitly by the NLS_TERRITORY parameter. A TIMESTAMP WITH TIMEZONE value requires 13 bytes.
VARCHAR2( <i>size</i> [BYTE CHAR])	A variable length character string with a maximum length of <i>size</i> characters or bytes. The size can range from 1 to 32,767 bytes.

## Operators

An operator manipulates data items and returns a result. Operators manipulate individual data items called operands or arguments. They are represented by special characters or by keywords. Syntactically, an operator appears before an operand, after an operand, or between two operands.

The OLAP Expression Syntax has these types of operators:

- [Unary Arithmetic Operators](#)
- [Binary Arithmetic Operators](#)
- [Concatenation Operator](#)

For conditional operators, go to "[Conditions](#)" on page 1-6.

## Unary Arithmetic Operators

A unary operator operates on only one operand.

**Table 1–3 Unary Operators**

Operator	Description
+	Positive value
-	Negative value

### Syntax

operator operand

### Example

-5 is a negative number.

## Binary Arithmetic Operators

A binary operator operates on two operands.

**Table 1–4 Binary Operators**

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

### Syntax

operand operator operand

### Examples

Here are two simple examples using numeric literals for the operands.

$7 * 2$  is 14.

$(8/2) + 1$  is 5.

This example multiplies the values of the Sales measure by a numeric literal to create a calculated measure named Sales Budget.

`UNITS_CUBE.SALES * 1.06`

**Table 1–5 Multiplication Operator Example**

Product	Level	Sales	Sales Budget
Hardware	CLASS	124191336	131642816
Desktop PCs	FAMILY	74556528	79029919
Monitors	FAMILY	3972142	4210470
Memory	FAMILY	5619219	5956372
Modems/Fax	FAMILY	5575726	5910269

**Table 1–5 (Cont.) Multiplication Operator Example**

Product	Level	Sales	Sales Budget
CD/DVD	FAMILY	16129497	17097267
Portable PCs	FAMILY	18338225	19438518

The next example creates a calculated measure named Profit by subtracting Cost from Sales.

```
UNITS_CUBE.SALES - UNITS_CUBE.COST
```

Product	Level	Sales	Cost	Profit
Hardware	CLASS	124191336	116058248	8133088
Desktop PCs	FAMILY	74556528	71937312	2619215
Monitors	FAMILY	3972142	3546195	425947
Memory	FAMILY	5619219	4962527	656692
Modems/Fax	FAMILY	5575726	5162879	412847
CD/DVD	FAMILY	16129497	12510832	3618664
Portable PCs	FAMILY	18338225	17938502	399723

## Concatenation Operator

The concatenation operator (||) combines text expressions.

### Syntax

```
operand || operand
```

### Example

'The date today is: ' || sysdate generates a text string such as The date today is: 23-AUG-06.

The next example concatenates the level name and dimension keys of the Product dimension to create an identifier.

```
PRODUCT.LEVEL_NAME || ' ' || PRODUCT.DIM_KEY
```

Level	Dim Key	Identifier
CLASS	HRD	CLASS HRD
FAMILY	DTPC	FAMILY DTPC
FAMILY	MON	FAMILY MON
FAMILY	MEM	FAMILY MEM
FAMILY	MOD	FAMILY MOD
FAMILY	DISK	FAMILY DISK
FAMILY	LTPC	FAMILY LTPC

## Conditions

A condition specifies a combination of one or more expressions and logical (Boolean) operators. The OLAP Expression Syntax has these types of conditions:

- [Simple Comparison Conditions](#)
- [Group Comparison Conditions](#)
- [Range Conditions](#)
- [Multiple Conditions](#)
- [Negation Conditions](#)
- [Special Conditions](#)
- [Pattern-Matching Conditions](#)

### Return Value

NUMBER (0=FALSE, 1=TRUE)

## Simple Comparison Conditions

Comparison conditions compare one expression with another.

You can use these comparison operators:

**Table 1–6 Simple Comparison Operators**

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to
!= ^= <>	Not equal to

### Syntax

expr { > | >= | < | <= | = | != | ^= | <> } expr

### Arguments

expr can be any expression.

### Examples

5 > 3 is true, 4 != 5 is true, 6 >= 9 is false.

## Group Comparison Conditions

A group comparison condition specifies a comparison with any or all members in a list or subquery.

You can use these comparison operators:



**Table 1–7 Group Comparison Operators**

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to
!= ^= <>	Not equal to
ANY SOME	Returns true if one or more values in the list match, or false if no values match.
ALL	Returns true if all values in the list match, or false if one or more values do not match.

**Syntax**

expr

```
{ = | != | ^= | <> | > | < | >= | <= }
{ ANY | SOME | ALL }
({ expression_list | subquery })
```

**Examples**

5 <= ALL (5, 10, 15) is true, 5 <> ANY (5, 10, 15) is true.

**Hierarchical Relation Conditions**

Hierarchical relation conditions specify the comparison of the relationship of a hierarchy member to itself or to another member of the hierarchy.

**Syntax**

```
expr1 IS [ NOT ]
{ PARENT
| CHILD
| ANCESTOR
| ROOT_ANCESTOR
| DESCENDANT
| LEAF_DESCENDANT
| RELATIVE
}
[ OR SELF ] OF expr2 WITHIN hierarchy
```

**Arguments**

expr1 is any expression, including a literal or a column, that resolves to a dimension member.

PARENT compares *expr1* as the parent of *expr2*.

CHILD compares *expr1* as a child of *expr2*.

ANCESTOR compares *expr1* as an ancestor of *expr2*.

ROOT\_ANCESTOR compares *expr1* as the highest-level ancestor of *expr2*.

DESCENDANT compares *expr1* as a descendant of *expr2*.

LEAF\_DESCENDANT compares *expr1* as a descendant that has no children of *expr2*.

RELATIVE compares *expr1* as a dimension member that has a parent in common with *expr2*.

OR SELF compares *expr1* as the same dimension member as *expr2*.

*expr2* is any expression, including a literal or a column, that resolves to a dimension member.

*hierarchy* is the hierarchy to consider when determining the relationship between *expr1* and *expr2*, expressed in the form *dimension\_name.hierarchy\_name*, as in PRODUCT.PRIMARY, or *owner.dimension\_name.hierarchy\_name*, as in GLOBAL.PRODUCT.PRIMARY.

### Examples

```
TIME.DIM_KEY IS CHILD OR SELF OF 'FY2008' WITHIN TIME.FISCAL
'FY2008' IS ROOT_ANCESTOR OR SELF OF TIME.DIM_KEY WITHIN TIME.FISCAL
'MEMORY' IS NOT ANCESTOR OF PRODUCT.DIM_KEY WITHIN GLOBAL.PRODUCT.PRIMARY
```

## Range Conditions

The BETWEEN operator tests whether a value is in a specific range of values. It returns true if the value being tested is greater than or equal to a low value and less than or equal to a high value.

### Syntax

```
expr [ NOT ] BETWEEN expr AND expr
```

### Example

7 NOT BETWEEN 10 AND 15 is true.

## Multiple Conditions

Conjunctions compare a single expression with two conditions.

**Table 1–8 Conjunctions**

Operator	Description
AND	Returns true if both component conditions are true. Returns false if either is false.
OR	Returns true if either component condition is true. Returns false if both are false.

### Syntax

```
expr operator condition1 { AND | OR } condition2
```

### Example

5 < 7 AND 5 > 3 is true; 5 < 3 OR 10 < 15 is true.

## Negation Conditions

The NOT operator reverses the meaning of a condition. It returns true if the condition is false. It returns false if the condition is true.

## Syntax

```
NOT { BETWEEN | IN | LIKE | NULL }
```

## Example

5 IS NOT NULL is true; 5 NOT IN (5, 10, 15) is false.

## Special Conditions

The IS operator tests for special conditions, such as nulls, infinity and values that are not numbers.

**Table 1–9** *Special Conditions Operators*

Operator	Description
IS INFINITE	Returns true if the expression is infinite, or false otherwise. For mappings only.
IS NAN	Returns true if the expression is not a number, or false otherwise. For mappings only.
IS NULL	Returns true if the expression is null, or false otherwise.

## Syntax

```
expr IS [ NOT ] NULL
```

## Example

13 IS NOT NULL is true.

## Pattern-Matching Conditions

The pattern-matching conditions compare character data.

### LIKE Operators

The LIKE operators specify a test involving pattern matching. Whereas the equality operator (=) exactly matches one character value to another, the LIKE operators can match patterns defined by special pattern-matching ("wildcard") characters.

You can choose from these LIKE operators:

**Table 1–10** *LIKE Pattern-Matching Operators*

Operator	Description
LIKE	Uses characters in the input character set.
LIKEC	Uses Unicode complete characters. It treats a Unicode supplementary character as two characters.
LIKE2	Uses UCS2 code points. It treats a Unicode supplementary character as one character.
LIKE4	Uses UCS4 code points. It treats a composite character as one character.

## Syntax

```
char1 [ NOT ] ( LIKE | LIKEC | LIKE2 | LIKE4 )
char2 [ ESCAPE esc_char ]
```

## Arguments

`char1` is a text expression for the search value.

`char2` is a text expression for the pattern. The pattern can contain these wildcard characters:

- An underscore (`_`) matches exactly one character (as opposed to one byte in a multibyte character set) in the value.
- A percent sign (`%`) can match zero or more characters (as opposed to bytes in a multibyte character set) in the value. A `'%'` cannot match a null.

`esc_char` is a text expression, usually a literal, that is one character long. This escape character identifies an underscore or a percent sign in the pattern as literal characters instead of wildcard characters. You can also search for the escape character itself by repeating it. For example, if `@` is the escape character, then you can use `@%` to search for `%` and `@@` to search for `@`.

## Examples

```
'Ducks' LIKE 'Duck_' and 'Ducky' LIKE 'Duck_' are true.
```

```
'Duckling' LIKE 'Duck_' is false.
```

```
'Duckling' LIKE 'Duck%' is true.
```

## Literal Expressions

The OLAP Expression Syntax has three types of literal expressions: strings, numbers, and null. Other data types must be created using conversion functions such as `TO_DATE`.

The terms **text literal**, **character literal**, and **string** are used interchangeably. They are always enclosed in single quotes to distinguish them from object names.

## Examples

```
'A Literal Text String'
```

```
'A Literal Text String with ''Quotes '''
```

```
'A Literal Text String  
That Crosses Into a Second Line'
```

```
2
```

```
2.4
```

```
+1
```

```
-1
```

```
NULL
```

## CASE Expressions

CASE expressions let you use `IF... THEN... ELSE` logic in expressions.

In a simple case expression, CASE searches for the first `WHEN... THEN` pair for which `expr` equals `comparison_expr`, then it returns `return_expr`. If none of the `WHEN... THEN` pairs

meet this condition, and an ELSE clause exists, then CASE returns `else_expr`. Otherwise, CASE returns NULL.

In a searched CASE expression, CASE searches from left to right until it finds an occurrence of condition that is true, and then returns `return_expr`. If no condition is found to be true, and an ELSE clause exists, CASE returns `else_expr`. Otherwise, CASE returns NULL.

## Return Value

Same as the `else_expression` argument

## Syntax

```
CASE { simple_case_expression
      | searched_case_expression
      }
      [ ELSE else_expression ]
END
```

```
simple_case_expression ::=
  expr WHEN comparison_expr
      THEN return_expr
  [ WHEN comparison_expr
      THEN return_expr ]...
```

```
searched_case_expression ::=
  WHEN condition THEN return_expr
  [ WHEN condition THEN return_expr ]...
```

## Arguments

`expr` is the base expression being tested.

`comparison_expr` is the expression against which `expr` is being tested. It must be the same basic data type (numeric or text) as `expr`.

`condition` is a conditional expression.

`return_expr` is the value returned when a match is found or the condition is true.

## Examples

This statement returns `Single Item` or `Value Pack` depending on whether the `PACKAGE` attribute of the `PRODUCT` dimension is null or has a value:

```
CASE PRODUCT.PACKAGE WHEN NULL THEN 'Single Item' ELSE 'Value Pack'END
```

Product	Package	Category
1.44MB External 3.5" Diskette	Executive	Value Pack
1GB USB Drive	--	Single Item
512MB USB Drive	--	Single Item
56Kbps V.90 Type II Modem	Executive	Value Pack
56Kbps V.92 Type II Fax/Modem	Laptop Value Pack	Value Pack
Deluxe Mouse	Executive	Value Pack

Product	Package	Category
Envoy Ambassador	--	Single Item
Envoy Executive	Executive	Value Pack
Envoy External Keyboard	Executive	Value Pack
Envoy Standard Laptop	Value Pack	Value Pack
External - DVD-RW - 8X	Executive	Value Pack
External 101-key keyboard	Multimedia	Value Pack
External 48X CD-ROM	--	Single Item
Internal - DVD-RW - 6X	Multimedia	Value Pack

The next statement increases the unit price by 20%, truncated to the nearest dollar, if the difference between price and cost is less than 10%. Otherwise, it returns the current unit price.

```

CASE
  WHEN PRICE_CUBE.UNIT_PRICE < PRICE_CUBE.UNIT_COST * 1.1
  THEN TRUNC (PRICE_CUBE.UNIT_COST * 1.2) ELSE PRICE_CUBE.UNIT_PRICE
END

```

Product	Cost	Price
1GB USB Drive	483.55	546.83
512MB USB Drive	234.69	275.91
56Kbps V.90 Type II Modem	135.72	158.58
56Kbps V.92 Type II Fax/Modem	95.01	111.08
Envoy Ambassador	2686.01	2850.88
Envoy Executive	2799.80	2943.96
Envoy Standard	1933.82	1921.62
External - DVD-RW - 8X	263.83	300.34
External 48X CD-ROM	223.11	254.15
Internal - DVD-RW - 6X	134.46	160.18
Internal 48X CD-ROM	108.32	127.54
Internal 48X CD-ROM USB	46.00	68.54
Monitor- 17"Super VGA	228.53	269.70
Monitor- 19"Super VGA	445.04	504.84
Sentinel Financial	1685.72	1764.14
Sentinel Multimedia	1849.17	1932.54
Sentinel Standard	1572.98	1610.53

The next example creates a Sales Budget calculated measure by multiplying Sales from the previous year by 1.06 for a 6% increase. The detail levels of all dimensions are excluded from the calculation. The Budget is projected only using data from 2006 or later.

```

CASE

```

```

WHEN TIME.END_DATE >= TO_DATE('01-JAN-2006')
AND TIME.LEVEL_NAME IN ('CALENDAR_YEAR', 'CALENDAR_QUARTER')
AND PRODUCT.LEVEL_NAME != 'ITEM'
AND CUSTOMER.LEVEL_NAME IN ('TOTAL', 'REGION', 'WAREHOUSE')
THEN TRUNC(LAG(UNITS_CUBE.SALES, 1) OVER HIERARCHY
(TIME.CALENDAR BY ANCESTOR AT LEVEL TIME.CALENDAR.CALENDAR_YEAR
POSITION FROM BEGINNING) * 1.06)
ELSE NULL
END

```

Product	Time	Sales
Hardware	Q1.05	28172590
Hardware	Q2.05	34520379
Hardware	Q3.05	29466573
Hardware	Q4.05	32031795
Hardware	Q1.06	32711891
Hardware	Q2.06	33637473
Hardware	Q3.06	29227635
Hardware	Q4.06	31319881
Hardware	Q1.07	--
Hardware	Q2.07	--
Hardware	Q3.07	--
Hardware	Q4.07	--

## Qualified Data References (QDRs)

Qualified data references (QDRs) limit a dimensional object to a single member in one or more dimensions for the duration of a query.

### Syntax

```
expression [ qualifier [ , qualifier ] ... ]
```

*qualifier*::=

```
dimension_id = member_expression
```

**Note:** The outside square brackets shown in bold are part of the syntax. In this case, they do not indicate an optional argument.

### Arguments

*expression* is a dimensional expression, typically the name of a measure.

*dimension\_id* is a cube dimension of *expression*.

*member\_expression* resolves to a single member of *dimension\_id*.

### Examples

`global.sales[global.time = 'CY2007']` returns Sales values for the year 2007.

`sales[customer = 'US', time = 'CY2007']` returns Sales values only for the United States in calendar year 2007.



---

---

## OLAP Functions

The OLAP functions extend the syntax of the SQL analytic functions. This syntax is familiar to SQL developers and DBAs, so you can adopt it more easily than proprietary OLAP languages and APIs. Using the OLAP functions, you can create all standard calculated measures, including rank, share, prior and future periods, period-to-date, parallel period, moving aggregates, and cumulative aggregates.

This chapter describes the OLAP functions. It contains these topics:

- [OLAP Functions in Alphabetical Order](#)
- [OLAP Functions By Category](#)

### OLAP Functions in Alphabetical Order

AVERAGE\_RANK  
AVG  
COUNT  
DENSE\_RANK  
HIER\_ANCESTOR  
HIER\_CHILD\_COUNT  
HIER\_DEPTH  
HIER\_LEVEL  
HIER\_ORDER  
HIER\_PARENT  
HIER\_TOP  
LAG  
LAG\_VARIANCE  
LAG\_VARIANCE\_PERCENT  
LEAD  
LEAD\_VARIANCE  
LEAD\_VARIANCE\_PERCENT  
MAX  
MIN  
OLAP\_DML\_EXPRESSION  
RANK  
ROW\_NUMBER  
SHARE  
SUM

### OLAP Functions By Category

The OLAP functions are grouped into these categories:

- [Aggregate Functions](#)
- [Analytic Functions](#)
- [Hierarchical Functions](#)
- [Lag Functions](#)
- [OLAP DML Functions](#)
- [Rank Functions](#)
- [Share Functions](#)
- [Window Functions](#)

## Aggregate Functions

[AVERAGE\\_RANK](#)  
[AVG](#)  
[COUNT](#)  
[DENSE\\_RANK](#)  
[MAX](#)  
[MIN](#)  
[RANK](#)  
[SUM](#)

## Analytic Functions

[AVERAGE\\_RANK](#)  
[AVG](#)  
[COUNT](#)  
[DENSE\\_RANK](#)  
[LAG](#)  
[LAG\\_VARIANCE](#)  
[LEAD\\_VARIANCE\\_PERCENT](#)  
[MAX](#)  
[MIN](#)  
[RANK](#)  
[ROW\\_NUMBER](#)  
[SUM](#)

## Hierarchical Functions

[HIER\\_ANCESTOR](#)  
[HIER\\_CHILD\\_COUNT](#)  
[HIER\\_DEPTH](#)  
[HIER\\_LEVEL](#)  
[HIER\\_ORDER](#)  
[HIER\\_PARENT](#)  
[HIER\\_TOP](#)

## Lag Functions

[LAG](#)  
[LAG\\_VARIANCE](#)  
[LAG\\_VARIANCE\\_PERCENT](#)  
[LEAD](#)  
[LEAD\\_VARIANCE](#)

LEAD\_VARIANCE\_PERCENT

## **OLAP DML Functions**

OLAP\_DML\_EXPRESSION

## **Rank Functions**

AVERAGE\_RANK

DENSE\_RANK

RANK

ROW\_NUMBER

## **Share Functions**

SHARE

## **Window Functions**

AVG

COUNT

MAX

MIN

SUM

## AVERAGE\_RANK

AVERAGE\_RANK orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members.

AVERAGE\_RANK assigns the same average rank to identical values. For example, AVERAGE\_RANK may return 1, 2, 3.5, 3.5, 5 for a series of five dimension members.

### Return Value

NUMBER

### Syntax

```
AVERAGE_RANK ( ) OVER (rank_clause)
```

*rank\_clause*::=

```
{ DIMENSION dimension_id | HIERARCHY hierarchy_id }
  ORDER BY order_by_clause [, order_by_clause]...
  [ WITHIN { LEVEL
            | PARENT
            | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                          | HIERARCHY LEVEL hier_level_id
                          }
          }
  ]
```

*order\_by\_clause*::=

```
expression [ASC | DESC] [NULLS {FIRST | LAST}]
```

### Arguments

#### ***dimension\_id***

The dimension over which the values are calculated using the default hierarchy.

#### ***hierarchy\_id***

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

#### **ORDER BY**

Provides the basis for the ranking. You can provide additional ORDER BY clauses to break any ties in the order.

#### ***expression***

Provides the values to use as the basis for the rankings.

#### **ASC | DESC**

Sorts the ranking from smallest to largest (ascending) or from largest to smallest (descending).

#### **NULLS {FIRST | LAST}**

Determines whether members with null values are listed first or last.

**WITHIN**

Selects a set of related dimension members to be ranked.

**LEVEL** ranks all members at the same level.

**PARENT** ranks members at the same level with the same parent.

**ANCESTOR** ranks all members at the same level and with the same ancestor at a specified level.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**Example**

This example ranks time periods within a calendar year by Unit Cost. Notice that no month is ranked 7, because two months (JAN-02 and JUL-02) have the same value and the same rank (6.5).

```
AVERAGE_RANK() OVER (HIERARCHY TIME.CALENDAR ORDER BY PRICE_CUBE.UNIT_COST
DESC NULLS LAST WITHIN ANCESTOR AT DIMENSION LEVEL TIME.CALENDAR_YEAR)
```

<b>Product</b>	<b>Time</b>	<b>Cost</b>	<b>Average Rank</b>
Deluxe Mouse	MAR-02	24.05	1
Deluxe Mouse	APR-02	23.95	2
Deluxe Mouse	FEB-02	23.94	3
Deluxe Mouse	AUG-02	23.88	4
Deluxe Mouse	MAY-02	23.84	5
Deluxe Mouse	JAN-02	23.73	6.5
Deluxe Mouse	JUL-02	23.73	6.5
Deluxe Mouse	JUN-02	23.72	8
Deluxe Mouse	SEP-02	23.71	9
Deluxe Mouse	NOV-02	23.65	10
Deluxe Mouse	DEC-02	23.62	11
Deluxe Mouse	OCT-02	23.37	12

**Related Topics**

[DENSE\\_RANK](#), [RANK](#), [ROW\\_NUMBER](#)

## AVG

AVG returns the average of a selection of values calculated over a Time dimension. Use this function to create cumulative averages and moving averages.

The GREGORIAN relations superimpose the Gregorian calendar on the Time dimension. These relations can be useful for calculations on fiscal and nonstandard hierarchies.

### Return Value

NUMBER

### Syntax

**AVG** (*value\_expr*) **OVER** (*window\_clause*)

*window\_clause*::=

```
[ { DIMENSION dimension_id | HIERARCHY hierarchy_id } ]
  BETWEEN preceding_boundary | following_boundary
    [ WITHIN { LEVEL
      | PARENT
      | GREGORIAN { YEAR | QUARTER | MONTH | WEEK }
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                      | HIERARCHY LEVEL hier_level_id
                      }
      }
    ]
```

*preceding\_boundary*::=

```
{ UNBOUNDED PRECEDING | expression PRECEDING } AND
  { CURRENT MEMBER
  | expression { PRECEDING | FOLLOWING }
  | UNBOUNDED FOLLOWING
  }
```

*following\_boundary*::=

```
{ CURRENT MEMBER | expression FOLLOWING } AND
  { expression FOLLOWING
  | UNBOUNDED FOLLOWING
  }
```

### Arguments

#### *value\_expr*

A dimensional expression whose values you want to calculate.

#### *dimension\_id*

The Time dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

***boundaries***

The *preceding\_boundary* and *following\_boundary* identify a range of time periods within the group identified by the dimension or hierarchy.

UNBOUNDED starts with the first period or ends with the last period of the group.

CURRENT MEMBER starts or ends the calculation at the current time period.

***expression***

A numeric value identifying a period at a particular distance from the current time period that starts or ends the range.

**WITHIN subclause**

Identifies the range of time periods used in the calculation. Following are descriptions of the keywords.

Range	Description
LEVEL	Calculates all time periods at the same level. (Default)
PARENT	Calculates time periods at the same level with the same parent.
GREGORIAN YEAR	Calculates time periods within the same Gregorian year.
GREGORIAN QUARTER	Calculates time periods within the same Gregorian quarter.
GREGORIAN MONTH	Calculates time periods within the same Gregorian month.
GREGORIAN WEEK	Calculates time periods within the same Gregorian week.
ANCESTOR	Includes time periods at the same level and with the same ancestor at a specified level.

**Example**

This example calculates a cumulative average within each parent. The selection of data shows the cumulative averages for quarters within the 2005 and 2006 calendar years.

```
AVG (GLOBAL.UNITS_CUBE.UNITS) OVER (HIERARCHY GLOBAL.TIME.CALENDAR BETWEEN UNBOUNDED PRECEDING AND CURRENT MEMBER WITHIN PARENT)
```

TIME	PARENT	UNITS	AVERAGE
Q1.05	CY2005	143607	143607
Q2.05	CY2005	138096	140852
Q3.05	CY2005	138953	140219
Q4.05	CY2005	145062	141430
Q1.06	CY2006	146819	146819
Q2.06	CY2006	145233	146026
Q3.06	CY2006	143572	145208

<b>TIME</b>	<b>PARENT</b>	<b>UNITS</b>	<b>AVERAGE</b>
Q4.06	CY2006	149305	146232

**Related Topics**

[COUNT](#), [MAX](#), [MIN](#), [SUM](#)



## COUNT

COUNT tallies the number of data values identified by a selection of members in a Time dimension.

The GREGORIAN relations superimpose the Gregorian calendar on the Time dimension. These relations can be useful for calculations on fiscal and nonstandard hierarchies.

### Return Value

NUMBER

### Syntax

**COUNT** (*value\_expr*) **OVER** (*window\_clause*)

*window\_clause*::=

```
{ DIMENSION dimension_id | HIERARCHY hierarchy_id }
BETWEEN preceding_boundary AND following_boundary
  [ WITHIN { LEVEL
              | PARENT
              | GREGORIAN { YEAR | QUARTER | MONTH | WEEK }
              | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                               | HIERARCHY LEVEL hier_level_id
                             }
            }
  ]
```

*preceding\_boundary*::=

```
{ UNBOUNDED PRECEDING | expression PRECEDING } AND
  { CURRENT MEMBER
    | expression { PRECEDING | FOLLOWING }
    | UNBOUNDED FOLLOWING
  }
```

*following\_boundary*::=

```
{ CURRENT MEMBER | expression FOLLOWING } AND
  { expression FOLLOWING
    | UNBOUNDED FOLLOWING
  }
```

### Arguments

#### *value\_expr*

A dimensional expression whose values you want to calculate.

#### *dimension\_id*

The Time dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

***boundaries***

The *preceding\_boundary* and *following\_boundary* identify a range of time periods within the group identified by the dimension or hierarchy.

UNBOUNDED starts with the first period or ends with the last period of the group.

CURRENT MEMBER starts or ends the calculation at the current time period.

***expression***

A numeric value identifying a period at a particular distance from the current time period that starts or ends the range.

**WITHIN subclause**

Identifies the range of time periods used in the calculation. Following are descriptions of the keywords.

Range	Description
LEVEL	Calculates all time periods at the same level. (Default)
PARENT	Calculates time periods at the same level with the same parent.
GREGORIAN YEAR	Calculates time periods within the same Gregorian year.
GREGORIAN QUARTER	Calculates time periods within the same Gregorian quarter.
GREGORIAN MONTH	Calculates time periods within the same Gregorian month.
GREGORIAN WEEK	Calculates time periods within the same Gregorian week.
ANCESTOR	Includes time periods at the same level and with the same ancestor at a specified level.

**Example**

This example tallies the number of time periods at the same level and the same year up to and including the current time period. The selected data displays the number of each month in the year.

```
COUNT (GLOBAL.UNITS_CUBE.UNITS) OVER (HIERARCHY GLOBAL.TIME.CALENDAR
BETWEEN UNBOUNDED PRECEDING AND CURRENT MEMBER WITHIN ANCESTOR AT
DIMENSION LEVEL GLOBAL.TIME.CALENDAR_YEAR)
```

TIME	UNITS	COUNT
JAN-06	47776	1
FEB-06	47695	2
MAR-06	51348	3
APR-06	47005	4
MAY-06	52809	5
JUN-06	45419	6

---

<b>TIME</b>	<b>UNITS</b>	<b>COUNT</b>
JUL-06	48388	7
AUG-06	48830	8
SEP-06	46354	9
OCT-06	47411	10
NOV-06	46842	11
DEC-06	55052	12

**Related Topics**[AVG, MAX, MIN, SUM](#)

## DENSE\_RANK

DENSE\_RANK orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members.

DENSE\_RANK assigns the same minimum rank to identical values, and returns the results in a sequential list. The result may be fewer ranks than values in the series. For example, DENSE\_RANK may return 1, 2, 3, 3, 4 for a series of five dimension members.

### Return Value

NUMBER

### Syntax

```
DENSE_RANK ( ) OVER (rank_clause)
```

*rank\_clause*::=

```
{ DIMENSION dimension_id | HIERARCHY hierarchy_id }
  ORDER BY order_by_clause [, order_by_clause]...
  [ WITHIN { LEVEL
            | PARENT
            | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                          | HIERARCHY LEVEL hier_level_id
                          }
          }
  ]
```

*order\_by\_clause*::=

```
expression [ASC | DESC] [NULLS {FIRST | LAST}]
```

### Arguments

#### ***dimension\_id***

The dimension over which the values are calculated using the default hierarchy.

#### ***hierarchy\_id***

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

#### **ORDER BY**

Provides the basis for the ranking. You can provide additional ORDER BY clauses to break any ties in the order.

#### **expression**

Provides the values to use as the basis for the rankings.

#### **ASC | DESC**

Sorts the ranking from smallest to largest (ascending) or from largest to smallest (descending).

**NULLS {FIRST | LAST}**

Determines whether members with null values are listed first or last.

**WITHIN**

Selects a set of related dimension members to be ranked.

**LEVEL** ranks all members at the same level.

**PARENT** ranks members at the same level with the same parent.

**ANCESTOR** ranks all members at the same level and with the same ancestor at a specified level.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**Example**

This example ranks time periods within a calendar year by Unit Cost, using the default Calendar hierarchy. Notice that although two months (JAN-02 and JUL-02) have the same value and the same rank (6), the ranking continues at 7 for JUN-02.

```
DENSE_RANK() OVER (DIMENSION "TIME" ORDER BY PRICE_CUBE.UNIT_COST DESC
NULLS LAST WITHIN ANCESTOR AT DIMENSION LEVEL TIME.CALENDAR_YEAR)
```

Product	Time	Cost	Dense Rank
Deluxe Mouse	MAR-02	24.05	1
Deluxe Mouse	APR-02	23.95	2
Deluxe Mouse	FEB-02	23.94	3
Deluxe Mouse	AUG-02	23.88	4
Deluxe Mouse	MAY-02	23.84	5
Deluxe Mouse	JAN-02	23.73	6
Deluxe Mouse	JUL-02	23.73	6
Deluxe Mouse	JUN-02	23.72	7
Deluxe Mouse	SEP-02	23.71	8
Deluxe Mouse	NOV-02	23.65	9
Deluxe Mouse	DEC-02	23.62	10
Deluxe Mouse	OCT-02	23.37	11

**Related Topics**

[AVERAGE\\_RANK](#), [RANK](#), [ROW\\_NUMBER](#)

---

## HIER\_ANCESTOR

HIER\_ANCESTOR returns the ancestor at a particular level of a hierarchy for either all members in the hierarchy or a particular member. The hierarchy must be level-based.

### Return Value

VARCHAR2

### Syntax

```
HIER_ANCESTOR(
  [member_expression] [WITHIN]
  {DIMENSION dimension_id | HIERARCHY hierarchy_id}
  {DIMENSION LEVEL dim_level_id | HIERARCHY LEVEL hier_level_id} )
```

### Arguments

#### *member\_expression*

Identifies a dimension member within the hierarchy whose ancestor is returned. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

#### *dim\_level\_id*

The level of the ancestor in *dimension\_id*.

#### *hier\_level\_id*

The level of the ancestor in *hierarchy\_id*.

### Example

This example returns the ancestor at the Calendar Quarter level for members of the default Calendar hierarchy of the Time dimension.

```
HIER_ANCESTOR(DIMENSION "TIME" DIMENSION LEVEL TIME.CALENDAR_QUARTER)
```

Time	Ancestor
2006	--
Q1.06	CY2006.Q1
Q2.06	CY2006.Q2
Q3.06	CY2006.Q3
Q4.06	CY2006.Q4
JAN-06	CY2006.Q1

---

<b>Time</b>	<b>Ancestor</b>
FEB-06	CY2006.Q1
MAR-06	CY2006.Q1
APR-06	CY2006.Q2
MAY-06	CY2006.Q2
JUN-06	CY2006.Q2
JUL-06	CY2006.Q3
AUG-06	CY2006.Q3
SEP-06	CY2006.Q3
OCT-06	CY2006.Q4
NOV-06	CY2006.Q4
DEC-06	CY2006.Q4

---

The next example returns GOV as the ancestor of the US Department of Labor at the Customer Market Segment level in the Market hierarchy of the Customer dimension.

```
HIER_ANCESTOR('US DPT LBR' WITHIN HIERARCHY CUSTOMER.MARKET DIMENSION  
LEVEL CUSTOMER.MARKET_SEGMENT)
```

## HIER\_CHILD\_COUNT

HIER\_CHILD\_COUNT returns the number of children of either all dimension members in a hierarchy or a particular member. The hierarchy can be either level-based or value-based.

### Return Value

NUMBER

### Syntax

```
HIER_CHILD_COUNT (
    [member_expression] [WITHIN]
    {DIMENSION dimension_id | HIERARCHY hierarchy_id} )
```

### Arguments

#### *member\_expression*

Identifies a single dimension member within the hierarchy used for the calculation. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

### Example

This example returns the number of children for each member of the default hierarchy of the Time dimension.

```
HIER_CHILD_COUNT(DIMENSION "TIME")
```

Time	Children
2006	4
Q1.06	3
Q2.06	3
Q3.06	3
Q4.06	3
JAN-06	0
FEB-06	0
MAR-06	0
APR-06	0
MAY-06	0



---

<b>Time</b>	<b>Children</b>
JUN-06	0
JUL-06	0
AUG-06	0
SEP-06	0
OCT-06	0
NOV-06	0
DEC-06	0

---

The next example returns 8 as the number of children for Government within the Market hierarchy of the Customer dimension.

```
HIER_CHILD_COUNT('GOV' WITHIN HIERARCHY CUSTOMER.MARKET)
```

## HIER\_DEPTH

HIER\_DEPTH returns a number representing the level depth of either all members of a hierarchy or a particular member, where 0 is the top level. The hierarchy can be either level-based or value-based.

### Return Value

NUMBER

### Syntax

```
HIER_DEPTH (
  [member_expression] [WITHIN]
  {DIMENSION dimension_id | HIERARCHY hierarchy_id} )
```

### Arguments

#### *member\_expression*

Identifies a single dimension member within the hierarchy used for the calculation. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

### Example

This example returns the depth of each member in the default hierarchy of the Time dimension.

```
HIER_DEPTH(DIMENSION "TIME")
```

Time	Depth
2006	1
Q1.06	2
Q2.06	2
Q3.06	2
Q4.06	2
JAN-06	3
FEB-06	3
MAR-06	3
APR-06	3
MAY-06	3

---

<b>Time</b>	<b>Depth</b>
JUN-06	3
JUL-06	3
AUG-06	3
SEP-06	3
OCT-06	3
NOV-06	3
DEC-06	3

---

The next example returns 2 as the depth of Italy in the default Customer hierarchy.

```
HIER_DEPTH('ITA' WITHIN DIMENSION CUSTOMER)
```

---

## HIER\_LEVEL

HIER\_LEVEL returns the level of either all members of a hierarchy or a particular member. The hierarchy must be level-based.

### Return Value

VARCHAR2

### Syntax

```
HIER_LEVEL (
  [member_expression] [WITHIN]
  {DIMENSION dimension_id | HIERARCHY hierarchy_id} )
```

### Arguments

#### *member\_expression*

Identifies a single dimension member within the hierarchy used for the calculation. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

### Example

This example returns the level of each member of the default hierarchy of the Time dimension.

```
HIER_LEVEL (DIMENSION "TIME")
```

Time	Level
2006	CALENDAR_YEAR
Q1.06	CALENDAR_QUARTER
Q2.06	CALENDAR_QUARTER
Q3.06	CALENDAR_QUARTER
Q4.06	CALENDAR_QUARTER
JAN-06	MONTH
FEB-06	MONTH
MAR-06	MONTH
APR-06	MONTH
MAY-06	MONTH
JUN-06	MONTH

---

<b>Time</b>	<b>Level</b>
JUL-06	MONTH
AUG-06	MONTH
SEP-06	MONTH
OCT-06	MONTH
NOV-06	MONTH
DEC-06	MONTH

---

The next example returns ACCOUNT as the level of Business World in the Market hierarchy of the Customer dimension.

```
HIER_LEVEL('BUSN WRLD' WITHIN HIERARCHY CUSTOMER.MARKET)
```

---

## HIER\_ORDER

HIER\_ORDER sorts the members of a dimension with children immediately after their parents, and returns a sequential number for each member.

### Return Value

NUMBER

### Syntax

```
HIER_ORDER (
  [member_expression] [WITHIN]
  {DIMENSION dimension_id | HIERARCHY hierarchy_id} )
```

### Arguments

#### *member\_expression*

Identifies a single dimension member within the hierarchy used for the calculation. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

### Example

This example orders the values of the Time dimension:

```
HIER_ORDER (DIMENSION "TIME")
```

Time	Order
2006	138
Q1.06	139
JAN-06	140
FEB-06	141
MAR-06	142
Q2.06	143
APR-06	144
MAY-06	145
JUN-06	146
Q3.06	147
JUL-06	148
AUG-06	149

---

<b>Time</b>	<b>Order</b>
SEP-06	150
Q4.06	151
OCT-06	152
NOV-06	153
DEC-06	154

---

The next example returns 78 as the order number of Business World in the Market hierarchy of the Customer dimension.

```
HIER_ORDER('BUSN WRLD' WITHIN HIERARCHY CUSTOMER.MARKET)
```

## HIER\_PARENT

HIER\_PARENT returns the parent of either all dimension members in a hierarchy or a particular member. The hierarchy can be either level-based or value-based.

### Return Value

VARCHAR2

### Syntax

```
HIER_PARENT (
  [member_expression] [WITHIN]
  {DIMENSION dimension_id | HIERARCHY hierarchy_id} )
```

### Arguments

#### *member\_expression*

Identifies a single dimension member within the hierarchy used for the calculation. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

### Example

This example returns the parents of all members of the default hierarchy of the Time dimension.

```
HIER_PARENT(DIMENSION GLOBAL.TIME)
```

<b>Time</b>	<b>Parent</b>
2006	TOTAL
Q1.06	CY2006
Q2.06	CY2006
Q3.06	CY2006
Q4.06	CY2006
JAN-06	CY2006.Q1
FEB-06	CY2006.Q1
MAR-06	CY2006.Q1
APR-06	CY2006.Q2
MAY-06	CY2006.Q2
JUN-06	CY2006.Q2



<b>Time</b>	<b>Parent</b>
JUL-06	CY2006.Q3
AUG-06	CY2006.Q3
SEP-06	CY2006.Q3
OCT-06	CY2006.Q4
NOV-06	CY2006.Q4
DEC-06	CY2006.Q4

The next example returns EMEA as the parent of Italy within the default hierarchy of the Customer dimension.

```
HIER_PARENT('ITA' WITHIN DIMENSION CUSTOMER)
```

## HIER\_TOP

HIER\_TOP returns the topmost ancestor of either all members of a hierarchy or a particular member. The hierarchy can be either level-based or value-based.

### Return Value

VARCHAR2

### Syntax

```
HIER_TOP (
  [member_expression] [WITHIN]
  {DIMENSION dimension_id | HIERARCHY hierarchy_id} )
```

### Arguments

#### *member\_expression*

Identifies a single dimension member within the hierarchy used for the calculation. If this optional argument is specified, then the result does not vary across dimension members.

#### *dimension\_id*

The dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

### Example

This example returns the top member of the default hierarchy of the Time dimension.

```
HIER_TOP(DIMENSION "TIME")
```

<b>Time</b>	<b>Top</b>
2006	TOTAL
Q1.06	TOTAL
Q2.06	TOTAL
Q3.06	TOTAL
Q4.06	TOTAL
JAN-06	TOTAL
FEB-06	TOTAL
MAR-06	TOTAL
APR-06	TOTAL
MAY-06	TOTAL
JUN-06	TOTAL
JUL-06	TOTAL

<b>Time</b>	<b>Top</b>
AUG-06	TOTAL
SEP-06	TOTAL
OCT-06	TOTAL
NOV-06	TOTAL
DEC-06	TOTAL

The next example returns TOTAL, which is the top member for Europe in the default hierarchy of the Customer dimension.

```
HIER_TOP('EMEA' WITHIN DIMENSION CUSTOMER)
```

## LAG

LAG returns the value from an earlier time period.

### Return Value

The same data type as the value expression

### Syntax

```
LAG (lag_args) OVER (lag_clause)
```

*lag\_args*::=

```
expression, offset [, {default_expression | CLOSEST} ]
```

*lag\_clause*::=

```
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]
[ [BY] { LEVEL
        | PARENT
        | GREGORIAN {YEAR | QUARTER | MONTH | WEEK | DAY}
        | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                        | HIERARCHY LEVEL hier_level_id
                        }
        }
  [POSITION FROM {BEGINNING | END}]
]
```

### Arguments

#### **expression**

A dimensional expression whose values you want to calculate.

#### **offset**

A numeric expression for the number of periods to count back from the current time period.

#### **default\_expression**

The value returned when *offset* does not identify a valid period. This clause is either an expression of any data type or the **CLOSEST** keyword for the closest match. The closest match is the first member when counting back.

#### **dimension\_id**

The Time dimension over which the lag is calculated.

#### **hierarchy\_id**

The hierarchy over which the lag is calculated. Otherwise, the default hierarchy for *dimension\_id* is used.

#### **dim\_level\_id**

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**BY subclause**

The BY subclause identifies the range of time periods used when counting the offset. Following are descriptions of the keywords:

Offset Unit	Description
LEVEL	The member at the same level <i>offset</i> periods before the current member. (Default)
PARENT	The member at the same level with the same parent <i>offset</i> periods before the current member.
GREGORIAN YEAR	The period at the same level with a start date exactly <i>offset</i> years before the start date of the current period.
GREGORIAN QUARTER	The period at the same level with a start date exactly <i>offset</i> quarters before the start date of the current period.
GREGORIAN MONTH	The period at the same level with a start date exactly <i>offset</i> months before the start date of the current period.
GREGORIAN WEEK	The period at the same level with a start date exactly <i>offset</i> weeks before the start date of the current period.
GREGORIAN DAY	The period at the same level with a start date exactly <i>offset</i> days before the start date of the current period.
ANCESTOR	The period at the same level as the current period and whose ancestor is <i>offset</i> positions before the ancestor of the current period.

**Example**

This example returns the value from the prior year for each period.

```
LAG(UNITS_CUBE.UNITS, 1) OVER (HIERARCHY "TIME".CALENDAR ANCESTOR AT
DIMENSION LEVEL "TIME".CALENDAR_YEAR)
```

Time	Units	Last Year
Q1.05	143607	146529
Q2.05	138096	143070
Q3.05	138953	148292
Q4.05	145062	149528
Q1.06	146819	143607
Q2.06	145233	138096
Q3.06	143572	138953
Q4.06	149305	145062

**Related Topics**

[LAG\\_VARIANCE](#), [LAG\\_VARIANCE\\_PERCENT](#), [LEAD](#)

## LAG\_VARIANCE

LAG\_VARIANCE returns the difference between values for the current time period and an earlier period.

### Return Value

The same data type as the value expression

### Syntax

```
LAG_VARIANCE (lag_args) OVER (lag_clause)
```

*lag\_args*::=

```
expression, offset [, {default_expression | CLOSEST} ]
```

*lag\_clause*::=

```
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]
[ [BY] { LEVEL
      | PARENT
      | GREGORIAN {YEAR | QUARTER | MONTH | WEEK | DAY}
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                    | HIERARCHY LEVEL hier_level_id
                    }
      }
  [POSITION FROM {BEGINNING | END}]
]
```

### Arguments

#### ***expression***

A dimensional expression whose values you want to calculate.

#### ***offset***

A numeric expression for the number of periods to count back from the current time period.

#### ***default\_expression***

The value returned when *offset* does not identify a valid period. This clause is either an expression of any data type or the CLOSEST keyword for the closest match. The closest match is the first member when counting back.

#### ***dimension\_id***

The Time dimension over which the lag is calculated.

#### ***hierarchy\_id***

The hierarchy over which the lag is calculated. Otherwise, the default hierarchy for *dimension\_id* is used.

#### ***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**BY subclause**

The BY subclause identifies the range of time periods used when counting the offset. Following are descriptions of the keywords:

Offset Unit	Description
LEVEL	The member at the same level <i>offset</i> periods before the current member. (Default)
PARENT	The member at the same level with the same parent <i>offset</i> periods before the current member.
GREGORIAN YEAR	The period at the same level with a start date exactly <i>offset</i> years before the start date of the current period.
GREGORIAN QUARTER	The period at the same level with a start date exactly <i>offset</i> quarters before the start date of the current period.
GREGORIAN MONTH	The period at the same level with a start date exactly <i>offset</i> months before the start date of the current period.
GREGORIAN WEEK	The period at the same level with a start date exactly <i>offset</i> weeks before the start date of the current period.
GREGORIAN DAY	The period at the same level with a start date exactly <i>offset</i> days before the start date of the current period.
ANCESTOR	The period at the same level as the current period and whose ancestor is <i>offset</i> positions before the ancestor of the current period.

**Examples**

This example returns the difference in values between the current period and the equivalent period in the prior year.

```
LAG_VARIANCE (GLOBAL.UNITS_CUBE.UNITS, 1) OVER (HIERARCHY
GLOBAL.TIME.CALENDAR ANCESTOR AT DIMENSION LEVEL GLOBAL.TIME.CALENDAR_
YEAR)
```

Time	Units	Last Year	Difference
Q1.05	143607	146529	-2922
Q2.05	138096	143070	-4974
Q3.05	138953	148292	-9339
Q4.05	145062	149528	-4466
Q1.06	146819	143607	3212
Q2.06	145233	138096	7137
Q3.06	143572	138953	4619
Q4.06	149305	145062	4243

**Related Topics**

[LAG](#), [LAG\\_VARIANCE\\_PERCENT](#), [LEAD](#)

## LAG\_VARIANCE\_PERCENT

LAG\_VARIANCE\_PERCENT returns the percent difference between values for the current time period and an earlier period.

### Return Value

NUMBER

### Syntax

```
LAG_VARIANCE_PERCENT (lag_args) OVER (lag_clause)

lag_args::=
expression, offset [, {default_expression | CLOSEST} ]

lag_clause::=
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]
[ [BY] { LEVEL
      | PARENT
      | GREGORIAN {YEAR | QUARTER | MONTH | WEEK | DAY}
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                    | HIERARCHY LEVEL hier_level_id
                  }
      }
  ]
[ POSITION FROM {BEGINNING | END}]
]
```

### Arguments

#### **expression**

A dimensional expression whose values you want to calculate.

#### **offset**

A numeric expression for the number of periods to count back from the current time period.

#### **default\_expression**

The value returned when *offset* does not identify a valid period. This clause is either an expression of any data type or the CLOSEST keyword for the closest match. The closest match is the first member when counting back.

#### **dimension\_id**

The Time dimension over which the lag is calculated.

#### **hierarchy\_id**

The hierarchy over which the lag is calculated. Otherwise, the default hierarchy for *dimension\_id* is used.

#### **dim\_level\_id**

The name of a level of *dimension\_id*.



***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**BY subclause**

The BY subclause identifies the range of time periods used when counting the offset. Following are descriptions of the keywords:

Offset Unit	Description
LEVEL	The member at the same level <i>offset</i> periods before the current member. (Default)
PARENT	The member at the same level with the same parent <i>offset</i> periods before the current member.
GREGORIAN YEAR	The period at the same level with a start date exactly <i>offset</i> years before the start date of the current period.
GREGORIAN QUARTER	The period at the same level with a start date exactly <i>offset</i> quarters before the start date of the current period.
GREGORIAN MONTH	The period at the same level with a start date exactly <i>offset</i> months before the start date of the current period.
GREGORIAN WEEK	The period at the same level with a start date exactly <i>offset</i> weeks before the start date of the current period.
GREGORIAN DAY	The period at the same level with a start date exactly <i>offset</i> days before the start date of the current period.
ANCESTOR	The period at the same level as the current period and whose ancestor is <i>offset</i> positions before the ancestor of the current period.

**Examples**

This example returns the percent difference in value between the current period and the equivalent period in the prior year.

```
LAG_VARIANCE_PERCENT (GLOBAL.UNITS_CUBE.UNITS, 1) OVER (HIERARCHY
GLOBAL.TIME.CALENDAR ANCESTOR AT DIMENSION LEVEL GLOBAL.TIME.CALENDAR_
YEAR)
```

Time	Units	Last Year	Difference	Percent
Q1.05	143607	146529	-2922	-.02
Q2.05	138096	143070	-4974	-.03
Q3.05	138953	148292	-9339	-.06
Q4.05	145062	149528	-4466	-.03
Q1.06	146819	143607	3212	.02
Q2.06	145233	138096	7137	.05
Q3.06	143572	138953	4619	.03
Q4.06	149305	145062	4243	.03

**Related Topics**

[LAG](#), [LAG\\_VARIANCE](#), [LEAD](#)

## LEAD

LEAD returns the value of an expression for a later time period.

### Return Value

The same data type as the value expression

### Syntax

```
LEAD (lead_args) OVER (lead_clause)
```

*lead\_args*::=

```
expression, offset [, {default_expression | CLOSEST} ]
```

*lead\_clause*::=

```
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]  
[ [BY] { LEVEL  
      | PARENT  
      | GREGORIAN {YEAR | QUARTER | MONTH | WEEK | DAY}  
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id  
                    | HIERARCHY LEVEL hier_level_id  
                    }  
      }  
  [POSITION FROM {BEGINNING | END}]]  
]
```

### Arguments

#### *expression*

A dimensional expression whose values you want to calculate.

#### *offset*

A numeric expression for the number of periods to count forward from the current time period.

#### *default\_expression*

The value returned when *offset* does not identify a valid period. This clause is either an expression of any data type or the CLOSEST keyword for the closest match. The closest match is the first member when counting forward.

#### *dimension\_id*

The Time dimension over which the lead is calculated.

#### *hierarchy\_id*

The hierarchy over which the lead is calculated. Otherwise, the default hierarchy for *dimension\_id* is used.

#### *dim\_level\_id*

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**BY subclause**

The BY subclause identifies the range of time periods used when counting the offset. Following are descriptions of the keywords:

Offset Unit	Description
LEVEL	The member at the same level <i>offset</i> periods after the current member. (Default)
PARENT	The member at the same level with the same parent <i>offset</i> periods after the current member.
GREGORIAN YEAR	The period at the same level with a start date exactly <i>offset</i> years after the start date of the current period.
GREGORIAN QUARTER	The period at the same level with a start date exactly <i>offset</i> quarters after the start date of the current period.
GREGORIAN MONTH	The period at the same level with a start date exactly <i>offset</i> months after the start date of the current period.
GREGORIAN WEEK	The period at the same level with a start date exactly <i>offset</i> weeks after the start date of the current period.
GREGORIAN DAY	The period at the same level with a start date exactly <i>offset</i> days after the start date of the current period.
ANCESTOR	The period at the same level as the current period and whose ancestor is <i>offset</i> positions after the ancestor of the current period.

**Examples**

This calculation returns the value of the next time period based on calendar quarter. The sample output from this calculation appears in the Next Qtr column.

```
LEAD (GLOBAL.UNITS_CUBE.UNITS, 1, CLOSEST) OVER (DIMENSION GLOBAL.TIME BY ANCESTOR AT DIMENSION LEVEL GLOBAL.TIME.CALENDAR_QUARTER)
```

Time	Parent	Units	Next Qtr
2005	TOTAL	565718	--
Q1-05	CY2005	143607	138096
Q2-05	CY2005	138096	138953
Q3-05	CY2005	138953	145062
Q4-05	CY2005	145062	146819
Jan-05	CY2005.Q1	50098	40223
Feb-05	CY2005.Q1	43990	45477
Mar-05	CY2005.Q1	49519	52396
Apr-05	CY2005.Q2	40223	45595
May-05	CY2005.Q2	45477	46882
Jun-05	CY2005.Q2	52396	46476
Jul-05	CY2005.Q3	45595	47476

<b>Time</b>	<b>Parent</b>	<b>Units</b>	<b>Next Qtr</b>
Aug-05	CY2005.Q3	46882	47496
Sep-05	CY2005.Q3	46476	50090
Oct-05	CY2005.Q4	47476	47776
Nov-05	CY2005.Q4	47496	47695
Dec-05	CY2005.Q4	50090	51348

**Related Topics**

[LAG](#), [LEAD\\_VARIANCE](#), [LEAD\\_VARIANCE\\_PERCENT](#)

---

## LEAD\_VARIANCE

LEAD\_VARIANCE returns the difference between values for the current time period and the offset period.

### Return Value

The same data type as the value expression

### Syntax

```
LEAD_VARIANCE (lead_args) OVER (lead_clause)
```

*lead\_args*::=

```
expression, offset [, {default_expression | CLOSEST} ]
```

*lead\_clause*::=

```
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]
[ [BY] { LEVEL
      | PARENT
      | GREGORIAN {YEAR | QUARTER | MONTH | WEEK | DAY}
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                    | HIERARCHY LEVEL hier_level_id
                  }
      }
  [POSITION FROM {BEGINNING | END}]
]
```

### Arguments

#### ***expression***

A dimensional expression whose values you want to calculate.

#### ***offset***

A numeric expression for the number of periods to count forward from the current time period.

#### ***default\_expression***

The value returned when *offset* does not identify a valid period. This clause is either an expression of any data type or the CLOSEST keyword for the closest match. The closest match is the first member when counting forward.

#### ***dimension\_id***

The Time dimension over which the lead is calculated.

#### ***hierarchy\_id***

The hierarchy over which the lead is calculated. Otherwise, the default hierarchy for *dimension\_id* is used.

#### ***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**BY subclause**

The BY subclause identifies the range of time periods used when counting the offset. Following are descriptions of the keywords:

Offset Unit	Description
LEVEL	The member at the same level <i>offset</i> periods after the current member. (Default)
PARENT	The member at the same level with the same parent <i>offset</i> periods after the current member.
GREGORIAN YEAR	The period at the same level with a start date exactly <i>offset</i> years after the start date of the current period.
GREGORIAN QUARTER	The period at the same level with a start date exactly <i>offset</i> quarters after the start date of the current period.
GREGORIAN MONTH	The period at the same level with a start date exactly <i>offset</i> months after the start date of the current period.
GREGORIAN WEEK	The period at the same level with a start date exactly <i>offset</i> weeks after the start date of the current period.
GREGORIAN DAY	The period at the same level with a start date exactly <i>offset</i> days after the start date of the current period.
ANCESTOR	The period at the same level as the current period and whose ancestor is <i>offset</i> positions after the ancestor of the current period.

**Examples**

This calculation returns the difference between the current value and the value of the next time period based on calendar quarter. The sample output from this calculation appears in the Difference column.

```
LEAD_VARIANCE (GLOBAL.UNITS_CUBE.UNITS, 1, CLOSEST) OVER (DIMENSION
GLOBAL.TIME BY ANCESTOR AT DIMENSION LEVEL GLOBAL.TIME.CALENDAR_QUARTER)
```

Time	Parent	Units	Next Qtr	Difference
2005	TOTAL	565718	--	--
Q1-05	CY2005	143607	138096	5511
Q2-05	CY2005	138096	138953	-857
Q3-05	CY2005	138953	145062	-6109
Q4-05	CY2005	145062	146819	-1757
Jan-05	CY2005.Q1	50098	40223	9875
Feb-05	CY2005.Q1	43990	45477	-1487
Mar-05	CY2005.Q1	49519	52396	-2877
Apr-05	CY2005.Q2	40223	45595	-5372
May-05	CY2005.Q2	45477	46882	-1405
Jun-05	CY2005.Q2	52396	46476	5920
Jul-05	CY2005.Q3	45595	47476	-1881

---

<b>Time</b>	<b>Parent</b>	<b>Units</b>	<b>Next Qtr</b>	<b>Difference</b>
Aug-05	CY2005.Q3	46882	47496	-614
Sep-05	CY2005.Q3	46476	50090	-3614
Oct-05	CY2005.Q4	47476	47776	-300
Nov-05	CY2005.Q4	47496	47695	-199
Dec-05	CY2005.Q4	50090	51348	-1258

---

**Related Topics**

[LAG](#), [LEAD](#), [LEAD\\_VARIANCE\\_PERCENT](#)

## LEAD\_VARIANCE\_PERCENT

LEAD\_VARIANCE\_PERCENT returns the percent difference between values for the current time period and the offset period.

### Return Value

NUMBER

### Syntax

```
LEAD_VARIANCE_PERCENT (lead_args) OVER (lead_clause)
```

*lead\_args*::=

```
expression, offset [, {default_expression | CLOSEST} ]
```

*lead\_clause*::=

```
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]
[ [BY] { LEVEL
      | PARENT
      | GREGORIAN {YEAR | QUARTER | MONTH | WEEK | DAY}
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                    | HIERARCHY LEVEL hier_level_id
                  }
      }
  [POSITION FROM {BEGINNING | END}]
]
```

### Arguments

#### ***expression***

A dimensional expression whose values you want to calculate.

#### ***offset***

A numeric expression for the number of periods to count forward from the current time period.

#### ***default\_expression***

The value returned when *offset* does not identify a valid period. This clause is either an expression of any data type or the CLOSEST keyword for the closest match. The closest match is the first member when counting forward.

#### ***dimension\_id***

The Time dimension over which the lead is calculated.

#### ***hierarchy\_id***

The hierarchy over which the lead is calculated. Otherwise, the default hierarchy for *dimension\_id* is used.

#### ***dim\_level\_id***

The name of a level of *dimension\_id*.



***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**BY subclause**

The BY subclause identifies the range of time periods used when counting the offset. Following are descriptions of the keywords:

Offset Unit	Description
LEVEL	The member at the same level <i>offset</i> periods after the current member. (Default)
PARENT	The member at the same level with the same parent <i>offset</i> periods after the current member.
GREGORIAN YEAR	The period at the same level with a start date exactly <i>offset</i> years after the start date of the current period.
GREGORIAN QUARTER	The period at the same level with a start date exactly <i>offset</i> quarters after the start date of the current period.
GREGORIAN MONTH	The period at the same level with a start date exactly <i>offset</i> months after the start date of the current period.
GREGORIAN WEEK	The period at the same level with a start date exactly <i>offset</i> weeks after the start date of the current period.
GREGORIAN DAY	The period at the same level with a start date exactly <i>offset</i> days after the start date of the current period.
ANCESTOR	The period at the same level as the current period and whose ancestor is <i>offset</i> positions after the ancestor of the current period.

**Example**

This calculation returns the percent difference between the current value and the value of the next time period based on calendar quarter. The sample output from this calculation appears in the Percent column.

```
LEAD_VARIANCE_PERCENT (GLOBAL.UNITS_CUBE.UNITS, 1, CLOSEST) OVER
(DIMENSION GLOBAL.TIME BY ANCESTOR AT DIMENSION LEVEL
GLOBAL.TIME.CALENDAR_QUARTER)
```

Time	Parent	Units	Next Qtr	Difference	Percent
2005	TOTAL	565718	--	--	--
Q1-05	CY2005	143607	138096	5511	.04
Q2-05	CY2005	138096	138953	-857	-.01
Q3-05	CY2005	138953	145062	-6109	-.04
Q4-05	CY2005	145062	146819	-1757	-.01
Jan-05	CY2005.Q1	50098	40223	9875	.25
Feb-05	CY2005.Q1	43990	45477	-1487	-.03
Mar-05	CY2005.Q1	49519	52396	-2877	-.05
Apr-05	CY2005.Q2	40223	45595	-5372	-.12
May-05	CY2005.Q2	45477	46882	-1405	-.03
Jun-05	CY2005.Q2	52396	46476	5920	.13

<b>Time</b>	<b>Parent</b>	<b>Units</b>	<b>Next Qtr</b>	<b>Difference</b>	<b>Percent</b>
Jul-05	CY2005.Q3	45595	47476	-1881	-.04
Aug-05	CY2005.Q3	46882	47496	-614	-.01
Sep-05	CY2005.Q3	46476	50090	-3614	-.07
Oct-05	CY2005.Q4	47476	47776	-300	-.01
Nov-05	CY2005.Q4	47496	47695	-199	0
Dec-05	CY2005.Q4	50090	51348	-1258	-.02

**Related Topics**

[LAG](#), [LEAD](#), [LEAD\\_VARIANCE](#)

## MAX

MAX returns the largest of a selection of data values calculated over a Time dimension.

The GREGORIAN relations superimpose the Gregorian calendar on the Time dimension. These relations can be useful for calculations on fiscal and nonstandard hierarchies.

### Return Value

NUMBER

### Syntax

**MAX** (*value\_expr*) **OVER** (*window\_clause*)

*window\_clause*::=

```
[ { DIMENSION dimension_id | HIERARCHY hierarchy_id } ]
  BETWEEN preceding_boundary | following_boundary
    [ WITHIN { PARENT
                | LEVEL
                | GREGORIAN { YEAR | QUARTER | MONTH | WEEK }
                | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                                   | HIERARCHY LEVEL hier_level_id
                                   }
            }
    ]
```

*preceding\_boundary*::=

```
{ UNBOUNDED PRECEDING | expression PRECEDING } AND
  { CURRENT MEMBER
    | expression { PRECEDING | FOLLOWING }
    | UNBOUNDED FOLLOWING
  }
```

*following\_boundary*::=

```
{ CURRENT MEMBER | expression FOLLOWING } AND
  { expression FOLLOWING
    | UNBOUNDED FOLLOWING
  }
```

### Arguments

#### *value\_expr*

A dimensional expression whose values you want to calculate.

#### *dimension\_id*

The Time dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

***boundaries***

The *preceding\_boundary* and *following\_boundary* identify a range of time periods within the group identified by the dimension or hierarchy.

UNBOUNDED starts with the first period or ends with the last period of the group.

CURRENT MEMBER starts or ends the calculation at the current time period.

***expression***

A numeric value identifying a period at a particular distance from the current time period that starts or ends the range.

**WITHIN subclause**

Identifies the range of time periods used in the calculation. Following are descriptions of the keywords.

Range	Description
LEVEL	Calculates all time periods at the same level. (Default)
PARENT	Calculates time periods at the same level with the same parent.
GREGORIAN YEAR	Calculates time periods within the same Gregorian year.
GREGORIAN QUARTER	Calculates time periods within the same Gregorian quarter.
GREGORIAN MONTH	Calculates time periods within the same Gregorian month.
GREGORIAN WEEK	Calculates time periods within the same Gregorian week.
ANCESTOR	Includes time periods at the same level and with the same ancestor at a specified level.

**Example**

This example calculates a moving maximum within the calendar year.

```
MAX (GLOBAL.UNITS_CUBE.UNITS) OVER (DIMENSION GLOBAL.TIME BETWEEN UNBOUNDED
PRECEDING AND CURRENT MEMBER WITHIN ANCESTOR AT DIMENSION LEVEL
GLOBAL.TIME.CALENDAR_YEAR)
```

Time	Units	Maximum
JAN-06	47776	47776
FEB-06	47695	47776
MAR-06	51348	51348
APR-06	47005	51348
MAY-06	52809	52809
JUN-06	45419	52809
JUL-06	48388	52809

---

<b>Time</b>	<b>Units</b>	<b>Maximum</b>
AUG-06	48830	52809
SEP-06	46354	52809
OCT-06	47411	52809
NOV-06	46842	52809
DEC-06	55052	55052

---

**Related Topics**[AVG, COUNT, MIN, SUM](#)

## MIN

MIN returns the smallest of a selection of data values calculated over a Time dimension. The GREGORIAN relations superimpose the Gregorian calendar on the Time dimension. These relations can be useful for calculations on fiscal and nonstandard hierarchies.

### Return Value

NUMBER

### Syntax

**MIN** (*value\_expr*) **OVER** (*window\_clause*)

*window\_clause*::=

```
[ { DIMENSION dimension_id | HIERARCHY hierarchy_id } ]
  BETWEEN preceding_boundary | following_boundary
    [ WITHIN { LEVEL
      | PARENT
      | GREGORIAN { YEAR | QUARTER | MONTH | WEEK }
      | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                      | HIERARCHY LEVEL hier_level_id
                      }
      }
    ]
```

*preceding\_boundary*::=

```
{ UNBOUNDED PRECEDING | expression PRECEDING } AND
  { CURRENT MEMBER
  | expression { PRECEDING | FOLLOWING }
  | UNBOUNDED FOLLOWING
  }
```

*following\_boundary*::=

```
{ CURRENT MEMBER | expression FOLLOWING } AND
  { expression FOLLOWING
  | UNBOUNDED FOLLOWING
  }
```

### Arguments

#### *value\_expr*

A dimensional expression whose values you want to calculate.

#### *dimension\_id*

The Time dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

***boundaries***

The *preceding\_boundary* and *following\_boundary* identify a range of time periods within the group identified by the dimension or hierarchy.

UNBOUNDED starts with the first period or ends with the last period of the group.

CURRENT MEMBER starts or ends the calculation at the current time period.

***expression***

A numeric value identifying a period at a particular distance from the current time period that starts or ends the range.

**WITHIN subclause**

Identifies the range of time periods used in the calculation. Following are descriptions of the keywords.

Range	Description
LEVEL	Calculates all time periods at the same level. (Default)
PARENT	Calculates time periods at the same level with the same parent.
GREGORIAN YEAR	Calculates time periods within the same Gregorian year.
GREGORIAN QUARTER	Calculates time periods within the same Gregorian quarter.
GREGORIAN MONTH	Calculates time periods within the same Gregorian month.
GREGORIAN WEEK	Calculates time periods within the same Gregorian week.
ANCESTOR	Includes time periods at the same level and with the same ancestor at a specified level.

**Example**

This example calculates the minimum value between the current member and all subsequent members in the same calendar year. The selection of the data displays the minimum values for the months in 2006.

```
MIN(GLOBAL.UNITS_CUBE.UNITS) OVER (DIMENSION GLOBAL.TIME BETWEEN CURRENT
MEMBER AND UNBOUNDED FOLLOWING WITHIN ANCESTOR AT DIMENSION LEVEL
GLOBAL.TIME.CALENDAR_YEAR)
```

Time	Units	Minimum
JAN-06	47776	45419
FEB-06	47695	45419
MAR-06	51348	45419
APR-06	47005	45419
MAY-06	52809	45419
JUN-06	45419	45419

<b>Time</b>	<b>Units</b>	<b>Minimum</b>
JUL-06	48388	46354
AUG-06	48830	46354
SEP-06	46354	46354
OCT-06	47411	46842
NOV-06	46842	46842
DEC-06	55052	55052

**Related Topics**

[AVG, COUNT, MAX, SUM](#)



---

## OLAP\_DML\_EXPRESSION

OLAP\_DML\_EXPRESSION executes an expression in the OLAP DML language.

### Return Value

The data type specified in the syntax

### Syntax

```
OLAP_DML_EXPRESSION (expression, datatype)
```

### Arguments

#### ***expression***

An expression in the OLAP DML language, such as a call to a function or a program.

#### ***datatype***

The data type of the return value from *expression*.

### Example

In this example, the OLAP\_DML\_EXPRESSION function executes the OLAP DML RANDOM function to generate a calculated measure with random numbers between 1.05 and 1.10.

```
OLAP_DML_EXPRESSION('RANDOM(1.05, 1.10)', NUMBER)
```

<b>Time</b>	<b>Product</b>	<b>Random</b>
2005	Hardware	1.07663806
2005	Software/Other	1.08295738
2006	Hardware	1.08707305
2006	Software/Other	1.09730881

## RANK

RANK orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members.

RANK assigns the same rank to identical values. For example, RANK may return 1, 2, 3, 3, 5 for a series of five dimension members.

### Return Value

NUMBER

### Syntax

```
RANK ( ) OVER (rank_clause)
```

*rank\_clause*::=

```
{ DIMENSION dimension_id | HIERARCHY hierarchy_id }
  ORDER BY order_by_clause [, order_by_clause]...
  [ WITHIN { PARENT
            | LEVEL
            | ANCESTOR AT { DIMENSION LEVEL dim_lvl_id
                          | HIERARCHY LEVEL hier_level_id
                          }
            }
  ]
```

*order\_by\_clause*::=

```
expression [ASC | DESC] [NULLS {FIRST | LAST}]
```

### Arguments

#### ***dimension\_id***

The dimension over which the values are calculated using the default hierarchy.

#### ***hierarchy\_id***

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, then the default hierarchy is used.

#### **ORDER BY**

Provides the basis for the ranking. You can provide additional ORDER BY clauses to break any ties in the order.

#### **expression**

Provides the values to use as the basis for the rankings.

#### **ASC | DESC**

Sorts the ranking from smallest to largest (ascending) or from largest to smallest (descending).

#### **NULLS {FIRST | LAST}**

Determines whether members with null values are listed first or last.

**WITHIN**

Selects a set of related dimension members to be ranked.

`PARENT` ranks members at the same level with the same parent.

`LEVEL` ranks all members at the same level.

`ANCESTOR` ranks all members at the same level and with the same ancestor at a specified level.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**Example**

This example ranks time periods within a calendar year by Unit Cost. Notice that no month is ranked 7, because two months (JAN-02 and JUL-02) have the same value and the same rank (6).

```
RANK() OVER (DIMENSION TIME.CALENDAR ORDER BY PRICE_CUBE.UNIT_COST DESC
NULLS LAST WITHIN ANCESTOR AT DIMENSION LEVEL TIME.CALENDAR_YEAR)
```

Product	Time	Cost	Rank
Deluxe Mouse	MAR-02	24.05	1
Deluxe Mouse	APR-02	23.95	2
Deluxe Mouse	FEB-02	23.94	3
Deluxe Mouse	AUG-02	23.88	4
Deluxe Mouse	MAY-02	23.84	5
Deluxe Mouse	JAN-02	23.73	6
Deluxe Mouse	JUL-02	23.73	6
Deluxe Mouse	JUN-02	23.72	8
Deluxe Mouse	SEP-02	23.71	9
Deluxe Mouse	NOV-02	23.65	10
Deluxe Mouse	DEC-02	23.62	11
Deluxe Mouse	OCT-02	23.37	12

**Related Topics**

[AVERAGE\\_RANK](#), [DENSE\\_RANK](#), [ROW\\_NUMBER](#)

---

## ROW\_NUMBER

ROW\_NUMBER orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members.

ROW\_NUMBER assigns a unique rank to each dimension member; for identical values, the rank is arbitrary. For example, ROW\_NUMBER always returns 1, 2, 3, 4, 5 for a series of five dimension members, even when they have the same value.

### Return Value

NUMBER

### Syntax

```
ROW_NUMBER ( ) OVER (rank_clause)
```

*rank\_clause*::=

```
{ DIMENSION dimension_id | HIERARCHY hierarchy_id }
  ORDER BY order_by_clause [, order_by_clause]...
  [ WITHIN { PARENT
            | LEVEL
            | ANCESTOR AT { DIMENSION LEVEL dim_lvl_id
                          | HIERARCHY LEVEL hier_level_id
                          }
          }
  ]
```

*order\_by\_clause*::=

```
expression [ASC | DESC] [NULLS {FIRST | LAST}]
```

### Arguments

#### ***dimension\_id***

The dimension over which the values are calculated using the default hierarchy.

#### ***hierarchy\_id***

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, then the default hierarchy is used.

#### **ORDER BY**

Provides the basis for the ranking. You can provide additional ORDER BY clauses to break any ties in the order.

#### **expression**

Provides the values to use as the basis for the rankings.

#### **ASC | DESC**

Sorts the ranking from smallest to largest (ascending) or from largest to smallest (descending).

**NULLS {FIRST | LAST}**

Determines whether members with null values are listed first or last.

**WITHIN**

Selects a set of related dimension members to be ranked.

**PARENT** ranks members at the same level with the same parent.

**LEVEL** ranks all members at the same level.

**ANCESTOR** ranks all members at the same level and with the same ancestor at a specified level.

***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

**Example**

This example ranks time periods within a calendar year by Unit Cost. Notice even though two months (JAN-02 and JUL-02) have the same value, they are assigned sequential numbers (6 and 7).

```
ROW_NUMBER() OVER (HIERARCHY TIME.CALENDAR ORDER BY PRICE_CUBE.UNIT_COST
DESC NULLS LAST WITHIN ANCESTOR AT DIMENSION LEVEL TIME.CALENDAR_YEAR)
```

Product	Time	Cost	Row Number
Deluxe Mouse	MAR-02	24.05	1
Deluxe Mouse	APR-02	23.95	2
Deluxe Mouse	FEB-02	23.94	3
Deluxe Mouse	AUG-02	23.88	4
Deluxe Mouse	MAY-02	23.84	5
Deluxe Mouse	JAN-02	23.73	6
Deluxe Mouse	JUL-02	23.73	7
Deluxe Mouse	JUN-02	23.72	8
Deluxe Mouse	SEP-02	23.71	9
Deluxe Mouse	NOV-02	23.65	10
Deluxe Mouse	DEC-02	23.62	11
Deluxe Mouse	OCT-02	23.37	12

**Related Topics**

[AVERAGE\\_RANK](#), [DENSE\\_RANK](#), [RANK](#)

---

## SHARE

SHARE calculates the ratio of an expression's value for the current dimension member to the value for a related member of the same dimension. Arguments to this function identify which related member is used in the ratio.

### Return Value

NUMBER

### Syntax

*share\_expression*::=

**SHARE** (expression share\_clause [share\_clause]... )

*share\_clause*::=

```
OF { DIMENSION dimension_id | HIERARCHY hierarchy_id }
  { PARENT
    | TOP
    | MEMBER 'member_name'
    | DIMENSION LEVEL dim_level_id
    | HIERARCHY LEVEL hier_level_id
  }
```

### Arguments

#### **expression**

A dimensional expression whose values you want to calculate.

#### **dimension\_id**

A dimension of *expression*. The default hierarchy is used in the calculation. If you want to use a different hierarchy, then use the HIERARCHY argument instead.

#### **hierarchy\_id**

A level hierarchy of *expression*.

#### **member\_name**

A member of the specified dimension or hierarchy.

#### **dim\_level\_id**

The name of a level of *dimension\_id*.

#### **hier\_level\_id**

The name of a level of *hierarchy\_id*.

Share is calculated with these formulas:

Keyword	Formula
PARENT	<i>current member/parent</i>
TOP	<i>current member/root ancestor</i>

<b>Keyword</b>	<b>Formula</b>
MEMBER	<i>current member/specified member</i>
DIMENSION LEVEL	<i>current member/ancestor at specified level</i> or null if the current member is above the specified level.

## Example

This example calculates the percent share of the parent member for each product. The results appear in the Share of Parent column.

$(\text{SHARE}(\text{UNITS\_CUBE.SALES OF HIERARCHY PRODUCT.PRIMARY PARENT})) * 100$

The next example calculates the percent share of Total Product for each product. The results appear in the Share of Top column.

$(\text{SHARE}(\text{UNITS\_CUBE.SALES OF HIERARCHY PRODUCT.PRIMARY TOP})) * 100$

<b>Product</b>	<b>Parent</b>	<b>Sales</b>	<b>Share of Parent</b>	<b>Share of Top</b>
Desktop PCs	HRD	74556528	60	54
Portable PCs	HRD	18338225	15	13
CD/DVD	HRD	16129497	13	12
Memory	HRD	5619219	5	4
Modems/Fax	HRD	5575726	4	4
Monitors	HRD	3972142	3	3
Accessories	SFT	6213535	49	5
Operating Systems	SFT	4766857	37	3
Documentation	SFT	1814844	14	1
Hardware	TOTAL	124191336	91	91
Software/Other	TOTAL	12795236	9	9

---

## SUM

SUM returns the total of a selection of values calculated over a Time dimension. You can use the SUM function to create period-to-date calculations.

The GREGORIAN relations superimpose the Gregorian calendar on the Time dimension. These relations can be useful for calculations on fiscal and nonstandard hierarchies.

### Return Value

NUMBER

### Syntax

```
SUM (value_expr) OVER (window_clause)
```

*window\_clause*::=

```
[ { DIMENSION dimension_id | HIERARCHY hierarchy_id } ]
  BETWEEN preceding_boundary | following_boundary
    [WITHIN { PARENT
              | LEVEL
              | GREGORIAN {YEAR | QUARTER | MONTH | WEEK}
              | ANCESTOR AT { DIMENSION LEVEL dim_level_id
                              | HIERARCHY LEVEL hier_level_id
                              }
            }
    ]
```

*preceding\_boundary*::=

```
{UNBOUNDED PRECEDING | expression PRECEDING} AND
 { CURRENT MEMBER
   | expression {PRECEDING | FOLLOWING}
   | UNBOUNDED FOLLOWING
 }
```

*following\_boundary*::=

```
{CURRENT MEMBER | expression FOLLOWING} AND
 { expression FOLLOWING
   | UNBOUNDED FOLLOWING
 }
```

### Arguments

#### *value\_expr*

A dimensional expression whose values you want to calculate.

#### *dimension\_id*

The Time dimension over which the values are calculated using the default hierarchy.

#### *hierarchy\_id*

The hierarchy over which the values are calculated. If *dimension\_id* is used instead, the default hierarchy is used.



***dim\_level\_id***

The name of a level of *dimension\_id*.

***hier\_level\_id***

The name of a level of *hierarchy\_id*.

***boundaries***

The *preceding\_boundary* and *following\_boundary* identify a range of time periods within the group identified by the dimension or hierarchy.

UNBOUNDED starts with the first period or ends with the last period of the group.

CURRENT MEMBER starts or ends the calculation at the current time period.

***expression***

A numeric value identifying a period at a particular distance from the current time period that starts or ends the range.

**WITHIN subclause**

Identifies the range of time periods used in the calculation. Following are descriptions of the keywords.

Range	Description
LEVEL	Calculates all time periods at the same level. (Default)
PARENT	Calculates time periods at the same level with the same parent.
GREGORIAN YEAR	Calculates time periods within the same Gregorian year.
GREGORIAN QUARTER	Calculates time periods within the same Gregorian quarter.
GREGORIAN MONTH	Calculates time periods within the same Gregorian month.
GREGORIAN WEEK	Calculates time periods within the same Gregorian week.
ANCESTOR	Includes time periods at the same level and with the same ancestor at a specified level.

**Example**

This example calculates the sum of two values, for the current and the following time periods, within a level. The results appear in the Sum column.

```
SUM(UNITS_CUBE.SALES) OVER (DIMENSION "TIME" BETWEEN 1 PRECEDING AND
CURRENT MEMBER WITHIN LEVEL)
```

Time	Sales	Sum
Q1.04	146529	289599
Q2.04	143070	291362
Q3.04	148292	297820
Q4.04	149528	293135
Q1.05	143607	281703
Q2.05	138096	277049
Q3.05	138953	284015

---

<b>Time</b>	<b>Sales</b>	<b>Sum</b>
Q4.05	145062	291881

---

The next example calculates Year-to-Date Sales.

```
SUM(UNITS_CUBE.SALES) OVER (HIERARCHY TIME.CALENDAR BETWEEN UNBOUNDED  
PRECEDING AND CURRENT MEMBER WITHIN ANCESTOR AT DIMENSION LEVEL  
TIME.CALENDAR_YEAR)
```

---

<b>Time</b>	<b>Sales</b>	<b>Sales YTD</b>
JAN-05	12093518	12093518
FEB-05	10103162	22196680
MAR-05	9184658	31381338
APR-05	9185964	40567302
MAY-05	11640216	52207519
JUN-05	16816561	69024079
JUL-05	11110903	80134982
AUG-05	9475807	89610789
SEP-05	12030538	101641328
OCT-05	11135032	112776359
NOV-05	11067754	123844113

---

## Related Topics

[AVG](#), [COUNT](#), [MAX](#), [MIN](#)

---

---

## Row Functions

The OLAP row functions extend the syntax of the SQL row functions for use with dimensional objects. If you use the SQL row functions, then this syntax is familiar. You can use these functions on relational data when loading it into cubes and cube dimensions, and with the OLAP functions when creating calculated measures.

This chapter describes the row functions of the OLAP expression syntax. It contains these topics:

- [Row Functions in Alphabetical Order](#)
- [Row Functions By Category](#)

### Row Functions in Alphabetical Order

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#)

#### **A**

[ABS](#)  
[ACOS](#)  
[ADD\\_MONTHS](#)  
[ASCII](#)  
[ASCIISTR](#)  
[ASIN](#)  
[ATAN](#)  
[ATAN2](#)

#### **B**

[BIN\\_TO\\_NUM](#)  
[BITAND](#)

#### **C**

[CAST](#)  
[CEIL](#)  
[CHARTOROWID](#)  
[CHR](#)  
[COALESCE](#)  
[CONCAT](#)  
[COS](#)  
[COSH](#)  
[CURRENT\\_DATE](#)  
[CURRENT\\_TIMESTAMP](#)

**D**

DBTIMEZONE  
DECODE

**E**

EXP  
EXTRACT (datetime)

**F**

FLOOR  
FROM\_TZ

**G**

GREATEST

**H**

HEXTORAW

**I**

INITCAP  
INSTR

**L**

LAST\_DAY  
LEAST  
LENGTH  
LN  
LNNVL  
LOCALTIMESTAMP  
LOG  
LOWER  
LPAD  
LTRIM

**M**

MOD  
MONTHS\_BETWEEN

**N**

NANVL  
NEW\_TIME  
NEXT\_DAY  
NLS\_CHARSET\_ID  
NLS\_CHARSET\_NAME  
NLS\_INITCAP  
NLS\_LOWER  
NLS\_UPPER  
NLSSORT  
NULLIF  
NUMTODSINTERVAL  
NUMTOYMINTERVAL  
NVL

NVL2

## O

ORA\_HASH

## P

POWER

## R

RAWTOHEX

REGEXP\_COUNT

REGEXP\_INSTR

REGEXP\_REPLACE

REGEXP\_SUBSTR

REMAINDER

REPLACE

ROUND (date)

ROUND (number)

ROWIDTOCHAR

ROWIDTONCHAR

RPAD

RTRIM

## S

SESSIONTIMEZONE

SIGN

SIN

SINH

SOUNDEX

SQRT

SUBSTR

SYS\_CONTEXT

SYSDATE

SYSTIMESTAMP

## T

TAN

TANH

TO\_BINARY\_DOUBLE

TO\_BINARY\_FLOAT

TO\_CHAR (character)

TO\_CHAR (datetime)

TO\_CHAR (number)

TO\_DATE

TO\_DSINTERVAL

TO\_NCHAR (character)

TO\_NCHAR (datetime)

TO\_NCHAR (number)

TO\_NUMBER

TO\_TIMESTAMP

TO\_TIMESTAMP\_TZ

TO\_YMINTERVAL

TRANSLATE

TRANSLATE (USING)

TRIM  
TRUNC (number)  
TZ\_OFFSET

## U

UID  
UNISTR  
UPPER  
USER

## V

VSIZE

## W

WIDTH\_BUCKET

## Row Functions By Category

The row functions are grouped into the following categories:

- [Numeric Functions](#)
- [Character Functions That Return Characters](#)
- [NLS Character Functions](#)
- [Character Functions That Return Numbers](#)
- [Datetime Functions](#)
- [General Comparison Functions](#)
- [Conversion Functions](#)
- [Encoding and Decoding Function](#)
- [Null-Related Functions](#)
- [Environment and Identifier Functions](#)

## Numeric Functions

These functions accept numeric input and return numeric values:

ABS  
ACOS  
ASIN  
ATAN  
ATAN2  
BITAND  
CEIL  
COS  
COSH  
EXP  
FLOOR  
LN  
LOG  
MOD  
NANVL

POWER  
REMAINDER  
ROUND (number)  
SIGN  
SIN  
SINH  
SQRT  
TAN  
TANH  
TRUNC (number)  
WIDTH\_BUCKET

## Character Functions That Return Characters

These functions accept character input and return character values:

CHR  
CONCAT  
INITCAP  
LOWER  
LPAD  
LTRIM  
NLS\_CHARSET\_NAME  
NLS\_INITCAP  
NLS\_LOWER  
NLS\_UPPER  
NLSSORT  
REGEXP\_REPLACE  
REGEXP\_SUBSTR  
REPLACE  
RPAD  
RTRIM  
SOUNDEX  
SUBSTR  
TRANSLATE  
TRIM  
UPPER

## NLS Character Functions

These functions return information about a character set:

NLS\_CHARSET\_ID  
NLS\_CHARSET\_NAME

## Character Functions That Return Numbers

These functions accept character input and return numeric values:

ASCII  
INSTR  
LENGTH  
REGEXP\_COUNT  
REGEXP\_INSTR

## Datetime Functions

These functions operate on date, timestamp, or interval values:

ADD\_MONTHS  
CURRENT\_DATE  
CURRENT\_TIMESTAMP  
DBTIMEZONE  
EXTRACT (datetime)  
FROM\_TZ  
LAST\_DAY  
LOCALTIMESTAMP  
MONTHS\_BETWEEN  
NEW\_TIME  
NEXT\_DAY  
NUMTODSINTERVAL  
NUMTOYMINTERVAL  
ROUND (date)  
SESSIONTIMEZONE  
SYSDATE  
SYSTIMESTAMP  
TO\_CHAR (datetime)  
TO\_DSINTERVAL  
TO\_TIMESTAMP  
TO\_TIMESTAMP\_TZ  
TO\_YMINTERVAL  
TZ\_OFFSET

## General Comparison Functions

These functions determine the greatest or least value in a set of values:

GREATEST  
LEAST

## Conversion Functions

These functions change a value from one data type to another:

ASCIISTR  
BIN\_TO\_NUM  
CAST  
CHARTOROWID  
HEXTORAW  
NUMTODSINTERVAL  
NUMTOYMINTERVAL  
RAWTOHEX  
ROWIDTOCHAR  
ROWIDTONCHAR  
TO\_BINARY\_DOUBLE  
TO\_BINARY\_FLOAT  
TO\_CHAR (character)  
TO\_CHAR (datetime)  
TO\_CHAR (number)  
TO\_DATE  
TO\_DSINTERVAL  
TO\_NCHAR (character)



TO\_NCHAR (datetime)  
TO\_NCHAR (number)  
TO\_NUMBER  
TO\_TIMESTAMP  
TO\_TIMESTAMP\_TZ  
TO\_YMINTERVAL  
TRANSLATE (USING)  
UNISTR

## Encoding and Decoding Function

These functions return a numeric value for each input value:

DECODE  
ORA\_HASH  
VSIZE

## Null-Related Functions

These functions facilitate null handling:

COALESCE  
LNNVL  
NANVL  
NULLIF  
NVL  
NVL2

## Environment and Identifier Functions

These functions provide information about the instance and the session:

SYS\_CONTEXT  
UID  
USER

## ABS

ABS returns the absolute value of a numeric expression.

### Return Value

NUMBER

### Syntax

**ABS** (n)

### Arguments

n is any numeric expression.

### Example

ABS (-15) returns the value 15.

## ACOS

ACOS calculates the angle value in radians of a specified cosine.

### Return Value

NUMBER

### Syntax

**ACOS**(n)

### Arguments

n is a numeric expression for the cosine in the range of -1 to 1.

### Example

ACOS (.3) returns the value 1.26610367.

## ADD\_MONTHS

ADD\_MONTHS returns a date that is a specified number of months after a specified date.

When the starting date is the last day of the month or when the returned month has fewer days, then ADD\_MONTHS returns the last day of the month. Otherwise, the returned day is the starting day.

### Return Value

DATE

### Syntax

**ADD\_MONTHS**(date, integer)

### Arguments

date is the starting date.

integer is the number of months to be added to the starting date.

### Example

ADD\_MONTHS('17-JUN-06', 1) returns the value 17-JUL-06.

## ASCII

ASCII returns the decimal representation of the first character of an expression.

### Return Value

NUMBER

### Syntax

**ASCII**(char)

### Arguments

char can be any text expression.

### Example

ASCII('Boston') returns the value 66, which is the ASCII equivalent of the letter B.

## ASCIISTR

ASCIISTR converts a string in any character set to ASCII in the database character set. Non-ASCII characters are represented as \xxxx, where xxxx is a UTF-16 code unit.

### Return Value

VARCHAR2

### Syntax

**ASCIISTR**(char)

### Arguments

char can be any character string.

### Example

ASCIISTR('Skåne') returns the value Sk\00E5ne.

## ASIN

ASIN calculates the angle value in radians of a specified sine.

### Return Value

NUMBER

### Syntax

**ASIN**(n)

### Arguments

n is a numeric expression in the range of -1 to 1 that contains the decimal value of a sine.

### Example

ASIN(.3) returns the value 0.304692654.

## ATAN

ATAN calculates the angle value in radians of a specified tangent.

Use ATAN2 to retrieve a full-range (0 - 2  $\pi$ ) numeric value indicating the arc tangent of a given ratio.

### Return Value

NUMBER

### Syntax

**ATAN**(n)

### Arguments

n is a numeric expression that contains the decimal value of a tangent.

### Example

ATAN (.3) returns the value 0.291456794.



## ATAN2

ATAN2 returns a full-range ( $0 - 2\pi$ ) numeric value of the arc tangent of a given ratio. The function returns values in the range of  $-\pi$  to  $\pi$ , depending on the signs of the arguments.

Use ATAN to calculate the angle value (in radians) of a specified tangent that is not a ratio.

### Return Value

NUMBER

### Syntax

**ATAN2**(n1, n2)

### Arguments

n1 and n2 are numeric expressions for the components of the ratio.

### Example

ATAN2(.3, .2) returns the value 0.982793723.

## BIN\_TO\_NUM

BIN\_TO\_NUM converts a bit vector to its equivalent number.

### Return Value

NUMBER

### Syntax

**BIN\_TO\_NUM**(expr [, expr ]... )

### Arguments

expr is a numeric expression with a value of 0 or 1 for the value of a bit in the bit vector.

### Example

BIN\_TO\_NUM(1, 0, 1, 0) returns the value 10.

## BITAND

BITAND computes an AND operation on the bits of two nonnegative integers, and returns an integer. This function is commonly used with the DECODE function.

An AND operation compares two bit values. If both values are 1, the operator returns 1. If one or both values are 0, the operator returns 0.

### Return Value

NUMBER

### Syntax

**BITAND**(expr1, expr2)

### Arguments

expr1 and expr2 are numeric expressions for nonnegative integers.

### Example

BITAND(7, 29) returns the value 5.

The binary value of 7 is 111 and of 29 is 11101. A bit-by-bit comparison generates the binary value 101, which is decimal 5.

---

## CAST

CAST converts values from one data type to another.

### Return Value

The data type specified by `type_name`.

### Syntax

```
CAST(expr AS type_name)
```

### Arguments

`expr` can be an expression in one of the data types.

`type_name` is one of the data types listed in [Table 1–2, "Dimensional Data Types"](#).

[Table 3–1](#) shows which data types can be cast into which other built-in data types. NUMBER includes NUMBER, DECIMAL, and INTEGER. DATETIME includes DATE, TIMESTAMP, TIMESTAMP WITH TIMEZONE, and TIMESTAMP WITH LOCAL TIMEZONE. INTERVAL includes INTERVAL DAY TO SECOND and INTERVAL YEAR TO MONTH.

**Table 3–1 Compatible Data Types**

From	To BINARY_FLOAT, BINARY_DOUBLE	To CHAR, VARCHAR2	To NUMBER	To DATETIME, INTERVAL	To NCHAR, NVARCHAR2
BINARY_FLOAT, BINARY_DOUBLE	yes	yes	yes	no	yes
CHAR, VARCHAR2	yes	yes	yes	yes	no
NUMBER	yes	yes	yes	no	yes
DATETIME, INTERVAL	no	yes	no	yes	yes
NCHAR, NVARCHAR2	yes	no	yes	no	yes

### Example

`CAST('123.4567' AS NUMBER(10,2))` returns the value 123.46.

## CEIL

CEIL returns the smallest whole number greater than or equal to a specified number.

### Return Value

NUMBER

### Syntax

**CEIL**(n)

### Arguments

n is a numeric expression.

### Examples

CEIL(3.1415927) returns the value 4.

CEIL(-3.4) returns the value -3.00.

## CHARTOROWID

CHARTOROWID converts a value from a text data type to a ROWID data type.

For more information about the ROWID pseudocolumn, refer to the *Oracle Database SQL Language Reference*.

### Return Value

ROWID

### Syntax

**CHARTOROWID**(char)

### Arguments

char is a text expression that forms a valid rowid.

### Example

`chartorowid('AAAN6EAALAAAAAMAAB')` returns the text string AAAN6EAALAAAAAMAAB as a rowid.

---

## CHR

CHR converts an integer to the character with its binary equivalent in either the database character set or the national character set.

For single-byte character sets, if  $n > 256$ , then CHR converts the binary equivalent of  $\text{mod}(n, 256)$ .

For the Unicode national character sets and all multibyte character sets,  $n$  must resolve to one entire code point. Code points are not validated, and the result of specifying invalid code points is indeterminate.

### Return Value

VARCHAR2 | NVARCHAR2

### Syntax

```
CHR(n [ USING NCHAR_CS ])
```

### Arguments

$n$  is a numeric expression.

USING NCHAR\_CS returns a character in the national character set. Otherwise, the return value is in the database character set. The OLAP engine uses the UTF8 national character set, so the return value may be different from the SQL CHR function, which uses the database UTF16 national character set.

### Example

CHR(67), CHR(67 USING NCHAR\_CS), and CHR(323) all return the letter C on an ASCII-based system with the WE8DEC database character set and the UTF8 national character set. CHR(323) is evaluated as CHR(MOD(323, 256)).

## COALESCE

COALESCE returns the first non-null expression in a list of expressions, or NULL when all of the expressions evaluate to null.

### Return Value

Data type of the first argument

### Syntax

```
COALESCE(expr [, expr ]...)
```

### Arguments

expr can be any expression.

### Examples

COALESCE(5, 8, 3) returns the value 5.

COALESCE(NULL, 8, 3) returns the value 8.



## CONCAT

CONCAT joins two expressions as a single character string. The data type of the return value is the same as the expressions, or if they are mixed, the one that results in a lossless conversion.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
CONCAT(char1, char2)
```

### Arguments

char1 and char2 are text expressions.

### Example

CONCAT('The current date is ', 'October 13, 2006') returns the string The current date is October 13, 2006.

## COS

COS calculates the cosine of an angle.

### Return Value

NUMBER

### Syntax

**cos** (n)

### Arguments

n is a numeric expression for an angle represented in radians.

### Example

`COS(180 * 3.1415927/180)` returns the cosine of 180 degrees as the value -1. The expression converts degrees to radians.

## COSH

COSH calculates the cosine of a hyperbolic angle.

### Return Value

NUMBER

### Syntax

**COSH**(n)

### Arguments

n is a numeric expression for a hyperbolic angle.

### Example

COSH(0) returns the value 1.

---

## CURRENT\_DATE

CURRENT\_DATE returns the current date in the session time zone.

### Return Value

DATE

### Syntax

CURRENT\_DATE

### Example

CURRENT\_DATE returns a value such as 12-AUG-08.

## CURRENT\_TIMESTAMP

CURRENT\_TIMESTAMP returns the current date and time in the session time zone. The time zone offset identifies the current local time of the session.

### Return Value

TIMESTAMP WITH TIME ZONE

### Syntax

**CURRENT\_TIMESTAMP** [ (precision) ]

### Arguments

precision specifies the fractional second precision of the returned time value. The default value is 6.

### Examples

CURRENT\_TIMESTAMP returns a value such as 08-AUG-06 11.18.31.082257 AM -08:00.

CURRENT\_TIMESTAMP(2) returns a value such as 08-AUG-06 11.18.31.08 AM -08:00.

## DBTIMEZONE

DBTIMEZONE returns the value of the database time zone as either a time zone offset from Coordinated Universal Time (UTC) or a time zone region name.

To obtain other time zone offsets, use TZ\_OFFSET.

### Return Value

VARCHAR2

### Syntax

**DBTIMEZONE**

### Example

DBTIMEZONE returns the offset -08:00 for Mountain Standard Time.

---

## DECODE

DECODE compares an expression to one or more search strings one by one.

If `expr` is `search`, then DECODE returns the corresponding `result`. If there is no match, then DECODE returns `default`. If you omit `default`, then DECODE returns NULL.

### Return Value

Data type of the first `result` argument

### Syntax

```
DECODE(expr, search, result
        [, search, result ]...
        [, default ]
      )
```

### Arguments

`expr` is an expression that is compared to one or more search strings.

`search` is a string that is searched for a match to `expr`.

`result` is the return value when `expr` matches the corresponding search string.

`default` is the return value when `expr` does not match any of the search strings. If `default` is omitted, then DECODE returns NULL.

The arguments can be any numeric or character type. Two nulls are equivalent. If `expr` is null, then DECODE returns the result of the first search that is also null.

The maximum number of components, including `expr`, searches, results, and `default`, is 255.

### Example

```
DECODE(sysdate, '21-JUN-06', 'Summer Solstice', '21-DEC-06', 'Winter
Solstice', 'Have a nice day!')
```

returns these values:

```
Summer Solstice on June 21, 2006
```

```
Winter Solstice on December 21, 2006
```

```
Have a nice day! on all other days
```

## EXP

EXP returns e raised to the nth power, where e = 2.71828183. The function returns a value of the same type as the argument.

### Return Value

NUMBER

### Syntax

**EXP** (n)

### Arguments

n is a numeric expression for the exponent.

### Example

EXP(4) returns the value 54.59815.



---

## EXTRACT (datetime)

EXTRACT returns the value of a specified field from a datetime or interval expression.

### Return Value

NUMBER

### Syntax

```

EXTRACT( ( { YEAR
          | MONTH
          | DAY
          | HOUR
          | MINUTE
          | SECOND
          }
        | { TIMEZONE_HOUR
          | TIMEZONE_MINUTE
          }
        | { TIMEZONE_REGION
          | TIMEZONE_ABBR
          }
        )
FROM { datetime_value_expression
      | interval_value_expression
      }
)

```

### Arguments

`datetime_value_expression` is an expression with a datetime data type.

`interval_value_expression` is an expression with an interval data type.

### Example

EXTRACT(MONTH FROM CURRENT\_TIMESTAMP) returns the value 8 for August when the current timestamp is 08-AUG-06 01.10.55.330120 PM -07:00.

EXTRACT(TIMEZONE\_HOUR FROM CURRENT\_TIMESTAMP) returns the value -7 from the same example.

---

## FLOOR

FLOOR returns the largest integer equal to or less than a specified number.

### Return Value

NUMBER

### Syntax

**FLOOR**(n)

### Arguments

n can be any numeric expression.

### Examples

FLOOR(15.7) returns the value 15.

FLOOR(-15.7) returns the value -16.

## FROM\_TZ

FROM\_TZ converts a timestamp value and a time zone to a `TIMESTAMP WITH TIME ZONE` data type.

### Return Value

`TIMESTAMP WITH TIME ZONE`

### Syntax

**FROM\_TZ** (timestamp\_value, time\_zone\_value)

### Arguments

timestamp\_value is an expression with a `TIMESTAMP` data type.

time\_zone\_value is a text expression that returns a string in the format `TZH:TZM` or in `TZR` with optional `TZD` format.

### Example

`FROM_TZ(TIMESTAMP '2008-03-26 08:00:00', '3:00')` returns the value `26-MAR-08 08.00.00.000000 AM +03:00`.

## GREATEST

GREATEST returns the largest expression in a list of expressions. All expressions after the first are implicitly converted to the data type of the first expression before the comparison. Text expressions are compared character by character.

To retrieve the smallest expression in a list of expressions, use LEAST.

### Return Value

The data type of the first expression

### Syntax

```
GREATEST(expr [, expr ]...)
```

### Arguments

expr can be any expression.

### Examples

GREATEST('Harry', 'Harriot', 'Harold') returns the value Harry.

GREATEST(7, 19, 3) returns the value 19.

## HEXTORAW

HEXTORAW converts a hexadecimal value to a raw value.

### Return Value

RAW

### Syntax

**HEXTORAW** (char)

### Arguments

char is a hexadecimal value in the CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type.

### Example

HEXTORAW('7D') returns the RAW value 7D.

## INITCAP

INITCAP returns a specified text expression, with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or non-alphanumeric characters. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

**INITCAP**(char)

### Arguments

char can be any text expression.

### Example

INITCAP('top ten tunes') and INITCAP('TOP TEN TUNES') both return the string Top Ten Tunes.

---

## INSTR

The `INSTR` functions search `string` for `substring`. The function returns an integer indicating the position of the character in `string`, or a zero (0) if does not find a match.

- `INSTR` calculates strings using characters as defined by the input character set.
- `INSTRB` uses bytes instead of characters.
- `INSTRC` uses Unicode complete characters.

`REGEXP_INSTR` provides additional options.

### Return Value

NUMBER

### Syntax

```
{ INSTR | INSTRB | INSTRC }  
(string , substring [, position [, occurrence ] ])
```

### Arguments

`string` is the text expression to search.

`substring` is the text string to search for.

`position` is a nonzero integer indicating the character in `string` where the function begins the search. When `position` is negative, then `INSTR` counts and searches backward from the end of `string`. The default value of `position` is 1, which means that the function begins searching at the first character of `string`.

`occurrence` is an integer indicating which occurrence of `string` the function should search for. The value of `occurrence` must be positive. The default values of `occurrence` is 1, meaning the function searches for the first occurrence of `substring`.

### Example

`INSTR('CORPORATE FLOOR', 'OR', 3, 2)` searches the string `CORPORATE FLOOR` beginning with the third character (`R`) for the second instance of the substring `OR`. It returns the value 14, which is the position of the second `O` in `FLOOR`.

---

## LAST\_DAY

LAST\_DAY returns the last day of the month in which a particular date falls.

### Return Value

DATE

### Syntax

**LAST\_DAY**(date)

### Arguments

date can be any datetime expression.

### Example

LAST\_DAY('26-MAR-06') returns the value 31-MAR-06.



## LEAST

LEAST returns the smallest expression in a list of expressions. All expressions after the first are implicitly converted to the data type of the first expression before the comparison. Text expressions are compared character by character.

To retrieve the largest expression in a list of expressions, use GREATEST.

### Return Value

The data type of the first expression

### Syntax

```
LEAST(expr [, expr ]...)
```

### Arguments

expr can be any expression.

### Examples

LEAST('Harry', 'Harriot', 'Harold') returns the value Harold.

LEAST(19, 3, 7) returns the value 3.

## LENGTH

The `LENGTH` functions return the length of a text expression.

- `LENGTH` counts the number of characters.
- `LENGTHB` uses bytes instead of characters.
- `LENGTHC` uses Unicode complete characters.

### Return Value

NUMBER

### Syntax

```
{ LENGTH | LENGTHB | LENGTHC }(char)
```

### Arguments

char is any text expression.

### Example

`LENGTH('CANDIDE')` returns the value 7.

## LN

LN returns the natural logarithm of a number greater than 0.

### Return Value

NUMBER

### Syntax

**LN**(n)

### Arguments

n can be any numeric expression with a value greater than 0.

### Example

LN(95) returns the value 4.55387689.

## LNNVL

LNNVL evaluates a condition when one or both operands of the condition may be null. LNNVL can be used anywhere a scalar expression can appear, even in contexts where the IS [NOT] NULL, AND, or OR conditions are not valid but would otherwise be required to account for potential nulls.

**NOTE:** This function returns 1 (true) if the condition is false or unknown, and 0 (false) if the condition is true.

### Return Value

NUMBER

### Syntax

**LNNVL**(condition)

### Arguments

condition can be any expression containing scalar values.

### Examples

LNNVL(1 > 4) returns 1 (true).

## LOCALTIMESTAMP

LOCALTIMESTAMP returns the current date and time in the session time zone.

### Return Value

TIMESTAMP

### Syntax

**LOCALTIMESTAMP** [ (timestamp\_precision) ]

### Arguments

timestamp\_precision specifies the fractional second precision of the time value returned.

### Examples

LOCALTIMESTAMP returns a value such as 09-AUG-06 08.11.37.045186 AM.

LOCALTIMESTAMP(2) returns a value such as 09-AUG-06 08.11.37.040000 AM.

## LOG

LOG computes the logarithm of an expression.

### Return Value

NUMBER

### Syntax

**LOG**(n2, n1)

### Arguments

n2 is the base by which to compute the logarithm.

n1 is the value whose logarithm is calculated. It can be any numeric expression that is greater than zero. When the value is equal to or less than zero, LOG returns a null value.

### Example

LOG(10,100) returns the value 2.

## LOWER

LOWER converts all alphabetic characters in a text expression to lowercase. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

**LOWER**(char)

### Arguments

char can be any text expression.

### Example

LOWER('STOP SHOUTING') returns the string stop shouting.

## LPAD

LPAD adds characters to the left of an expression to a specified length. The data type of the return value is the same as the original text.

Use RPAD to add characters to the right.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
LPAD(expr1, n [, expr2 ])
```

### Arguments

expr1 is a text expression for the base string.

n is the total length of the returned expression. If expr1 is longer than n, then this function truncates expr1 to n characters.

expr2 is a text expression for the padding characters. By default, it is a space.

### Example

LPAD('Page 1',15,'\*.\*') returns the value \*.\*.\*.\*Page 1.

LPAD('Stay tuned', 4) returns the value Stay.



## LTRIM

LTRIM scans a text expression from left to right and removes all the characters that match the characters in the trim expression, until it finds an unmatched character. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
LTRIM(char [, set ])
```

### Arguments

char is the text expression to be trimmed.

set is a text expression with the characters to remove. The default value of set is a single blank.

### Examples

LTRIM(' . . . .Last Word', ' .') returns the value Last Word.

## MOD

MOD returns the remainder after a number is divided by another, or the number if the divisor is 0 (zero).

### Return Value

NUMBER

### Syntax

**MOD**(n2, n1)

### Arguments

n2 is a numeric expression for the number to be divided.

n1 is a numeric expression for the divisor.

### Example

MOD(13,7) returns the value 6.

## MONTHS\_BETWEEN

`MONTHS_BETWEEN` calculates the number of months between two dates. When the two dates have the same day component or are both the last day of the month, then the return value is a whole number. Otherwise, the return value includes a fraction that considers the difference in the days based on a 31-day month.

### Return Value

NUMBER

### Syntax

```
MONTHS_BETWEEN(date1, date2)
```

### Arguments

`date1` and `date2` are datetime expressions. If `date1` is later than `date2`, then the result is positive. If `date1` is earlier than `date2`, then the result is negative.

### Example

```
MONTHS_BETWEEN('15-APR-06', '01-JAN-06') returns the value 3.4516129.
```

## NANVL

`NANVL` checks if a value is a number. If it is, then `NANVL` returns that value. If not, it returns an alternate value. This function is typically used to convert a binary double or binary float NaN (Not a Number) value to zero or null.

### Return Value

datatype

### Syntax

`NANVL` (expression, alternate)

### Arguments

expression can be any value.

alternate is the numeric value returned if expression is not a number.

---

## NEW\_TIME

`NEW_TIME` converts the date and time from one time zone to another. Before using this function, set the `NLS_DATE_FORMAT` parameter to display 24-hour time.

### Return Value

DATE

### Syntax

`NEW_TIME`(date, timezone1, timezone2)

### Arguments

date is a datetime expression to be converted to a new time zone.

timezone1 is the time zone of date.

timezone2 is the new time zone.

The time zone arguments are limited to the values in [Table 3–2](#). For other time zones, use `FROM_TZ`.

**Table 3–2 Time Zones**

Time Zone	Abbreviation
Alaska-Hawaii Daylight Time	HDT
Alaska-Hawaii Standard Time	HST
Atlantic Daylight Time	ADT
Atlantic Standard Time	AST
Bering Daylight Time	BDT
Bering Standard Time	BST
Central Daylight Time	CDT
Central Standard Time	CST
Eastern Daylight Time	EDT
Eastern Standard Time	EST
Greenwich Mean Time	GMT
Mountain Daylight Time	MDT
Mountain Standard Time	MST
Newfoundland Standard Time	NST
Pacific Daylight Time	PDT
Pacific Standard Time	PST
Yukon Daylight Time	YDT
Yukon Standard Time	YST

**Example**

`NEW_TIME(SYSDATE, 'PST', 'EST')` returns a value such as 18-JAN-07 04:38:07 in Eastern Standard Time when `SYSDATE` is 18-JAN-07 01:38:07 in Pacific Standard Time. For this example, `NLS_DATE_FORMAT` is set to `DD-MON-RR HH:MI:SS`.

## NEXT\_DAY

NEXT\_DAY returns the date of the first instance of a particular day of the week that follows the specified date.

### Return Value

DATE

### Syntax

**NEXT\_DAY**(date, char)

### Arguments

date is a datetime expression.

char is a text expression that identifies a day of the week (for example, Monday) in the language of your session.

### Example

NEXT\_DAY('11-SEP-01', 'Monday') returns the value 17-SEP-01.

## NLS\_CHARSET\_ID

NLS\_CHARSET\_ID returns the identification number corresponding to a specified character set name.

### Return Value

NUMBER

### Syntax

```
NLS_CHARSET_ID ( charset_name )
```

### Arguments

charset\_name is a VARCHAR2 expression that is a valid character set name.

### Example

NLS\_CHARSET\_ID('AL32UTF8') returns the value 873.



## NLS\_CHARSET\_NAME

NLS\_CHARSET\_NAME returns the name corresponding to a specified character set number.

### Return Value

VARCHAR2

### Syntax

**NLS\_CHARSET\_NAME** (charset\_id)

### Arguments

charset\_id is a valid character set number or one of these keywords:

- CHAR\_CS represents the database character set.
- NCHAR\_CS represents the national character set. The national character set for the database can be either UTF-8 or AL16UTF16 (default). However, the national character set for analytic workspaces is always UTF-8.

If the number does not correspond to a character set, then the function returns NULL.

### Example

NLS\_CHARSET\_NAME(2000) returns the value AL16UTF16.

## NLS\_INITCAP

NLS\_INITCAP returns a string in which each word begins with a capital followed by lower-case letters. White space and nonalphanumeric characters delimit the words.

### Return Value

VARCHAR2

### Syntax

```
NLS_INITCAP (char [, 'nlsparm' ])
```

### Arguments

char can be any text string.

nlsparm can have the form 'NLS\_SORT =sort' where sort is either a linguistic sort sequence or BINARY. The linguistic sort sequence handles special linguistic requirements for case conversions. These requirements can result in a return value of a different length than char. If you omit nlsparm, then this function uses the default sort sequence for your session.

### Example

NLS\_INITCAP('WALKING&THROUGH\*A\*winter wonderland') returns the value Walking#Through\*A\*Winter Wonderland.

NLS\_INITCAP('ijsland') returns the value Ijsland, but NLS\_INITCAP(NLS\_INITCAP('ijsland', 'NLS\_SORT = XDutch')) returns IJsland.

---

## NLS\_LOWER

NLS\_LOWER converts all alphabetic characters in a text expression to lowercase. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
NLS_LOWER (char [, 'nlsparm' ])
```

### Arguments

char can be any text expression.

nlsparm is a linguistic sort sequence in the form NLS\_SORT =sort[\_ai |\_ci], where sort is an NLS language. You can add a suffix to the language to modify the sort: \_ai for an accent-insensitive sort, or \_ci for a case-insensitive sort.

### Example

NLS\_LOWER('STOP SHOUTING') returns the string stop shouting.

## NLS\_UPPER

NLS\_UPPER converts all alphabetic characters in a text expression to uppercase. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
NLS_UPPER (char [, 'nlsparam' ])
```

### Arguments

char can be any text expression.

nlsparam is a linguistic sort sequence in the form NLS\_SORT =sort[\_ai |\_ci], where sort is an NLS language. You can add a suffix to the language to modify the sort: \_ai for an accent-insensitive sort, or \_ci for a case-insensitive sort.

### Example

NLS\_UPPER('This is an emergency') returns the string THIS IS AN EMERGENCY.

---

## NLSSORT

NLSSORT returns the string of bytes used to sort a text string. You can use this function to specify sorting and comparison operations based on a linguistic sort sequence rather than on the binary value of a string.

**Note:** NLSSORT returns a RAW value, which you may pass to another function. To create a measure or a calculated measure for the values returned by NLSSORT, use the RAWTOHEX function.

For more information about linguistic sorting, refer to the *Oracle Database Globalization Support Guide*.

### Return Value

RAW

### Syntax

```
NLSSORT(char [, 'nlsparam' ])
```

### Arguments

char is a text expression.

nlsparam is a linguistic sort sequence in the form NLS\_SORT =sort[\_ai |\_ci], where sort is an NLS language. You can add a suffix to the language to modify the sort: \_ai for an accent-insensitive sort, or \_ci for a case-insensitive sort.

### Example

NLSSORT('Rumpelstiltskin') returns the value 52756D70656C7374696C74736B696E00 for a default linguistic sort, which in this case is American.

## NULLIF

NULLIF compares one expression with another. It returns `NULL` when the expressions are equal, or the first expression when they are not.

### Return Value

Data type of the first argument

### Syntax

```
NULLIF(expr1, expr2)
```

### Arguments

`expr1` is the base expression. It cannot be a literal null.

`expr2` is the expression to compare with the base expression.

### Example

NULLIF('red', 'Red') returns the value red.

## NUMTODSINTERVAL

NUMTODSINTERVAL converts a number to an INTERVAL DAY TO SECOND data type.

### Return Value

INTERVAL DAY TO SECOND

### Syntax

```
NUMTODSINTERVAL(n, 'interval_unit')
```

### Arguments

n can be any numeric expression.

interval\_unit is a text expression that specifies the units. It must resolve to one of the following values:

- DAY
- HOUR
- MINUTE
- SECOND

These values are case insensitive.

### Example

NUMTODSINTERVAL(100, 'MINUTE') returns the value +00 01:40:00.000000.

## NUMTOYMINTERVAL

NUMTOYMINTERVAL converts a number to an INTERVAL YEAR TO MONTH data type.

### Return Value

INTERVAL YEAR TO MONTH

### Syntax

**NUMTOYMINTERVAL**(n, 'interval\_unit')

### Arguments

n can be any numeric expression.

interval\_unit is a text expression that specifies the units. It must resolve to one of the following values:

- YEAR
- MONTH

These values are case insensitive.

### Example

NUMTOYMINTERVAL(18, 'MONTH') returns the value +01-06.



## NVL

NVL replaces a null with a string. NVL returns the replacement string when the base expression is null, and the value of the base expression when it is not null.

To replace an expression with one value if it is null and a different value if it is not, use NVL2.

### Return Value

Data type of the first argument

### Syntax

```
NVL(expr1, expr2)
```

### Arguments

expr1 is the base expression that is evaluated.

expr2 is the replacement string that is returned when expr1 is null.

### Examples

`NVL('First String', 'Second String')` returns the value First String.

`NVL(null, 'Second String')` returns the value Second String.

## NVL2

NVL2 returns one value when the value of a specified expression is not null, or another value when the value of the specified expression is null.

To replace a null value with a string, use NVL.

### Return Value

Data type of the first argument

### Syntax

```
NVL2(expr1, expr2, expr3)
```

### Arguments

expr1 is the base expression whose value this function evaluates.

expr2 is an expression whose value is returned when expr1 is not null.

expr3 is an expression whose value is returned when expr1 is null.

### Example

NVL2('Which string?', 'First String', 'Second String') returns the value First String.

---

## ORA\_HASH

ORA\_HASH generates hash values for an expression. You can use it to randomly assign a set of values into several buckets for analysis, or to generate a set of random numbers.

### Return Value

NUMBER

### Syntax

**ORA\_HASH** (expr [, max\_bucket [, seed\_value ] ])

### Arguments

expr can be any expression that provides the data for generating the hash values.

max\_bucket is the maximum bucket number. For example, when max\_bucket is set to 5, ORA\_HASH returns values of 0 to 5, creating six buckets. Set this value from 0 to 4294967295 or  $2^{32}-1$  (default).

seed\_value is a value used by ORA\_HASH to generate the hash values. Enter a different seed\_value for different results. Set this value from 0 (default) to 4294967295 or  $2^{32}-1$ .

### Example

ORA\_HASH(PRODUCT\_CUBE.PRICES, 5) returns a value in the range of 0 to 5 for each value of the Prices measure, as shown in the Hash 5 column. The rows are also sorted on the Hash 5 column.

ORA\_HASH(PRODUCT\_CUBE.PRICES, 5, 13) also returns values in the range of 0 to 5, but uses a different seed.

Product	Prices	Hash 5	Seed 13
ENVY STD	200539.83	0	4
ENVY EXE	255029.31	0	5
1GB USB DRV	44645.65	1	2
DLX MOUSE	1379.49	2	2
144MB DISK	3011.43	2	5
512 USB DRV	22139.99	2	2
19 SVGA	34837.16	3	0
56KPS MODEM	12478	3	2
ENVY EXT KBD	4312.22	3	5
17 SVGA	22605.55	4	1
EXT CD ROM	17990.14	4	0
ENVY ABM	205462.25	5	1

## POWER

POWER raises a number to a power.

### Return Value

NUMBER

### Syntax

**POWER**(n2, n1)

### Arguments

n2 is any numeric expression that is raised to a power.

n1 is the exponent.

### Example

POWER(3, 2) returns the value 9.

## RAWTOHEX

RAWTOHEX converts raw data to a character value containing its hexadecimal representation.

### Return Value

VARCHAR2

### Syntax

**RAWTOHEX** (*raw*)

### Arguments

*raw* can be any scalar data type other than LONG, LONG RAW, CLOB, BLOB, or BFILE.

### Example

RAWTOHEX(NLSSORT('Rumpelstiltskin')) converts the raw value returned by NLSSORT to the hexadecimal value 52756D70656C7374696C74736B696E00.

## REGEXP\_COUNT

REGEXP\_COUNT searches a string for a regular pattern and returns the number of times the pattern occurs. If no match is found, the function returns 0.

The function evaluates strings using characters as defined by the input character set.

### Return Value

NUMBER

### Syntax

```
REGEXP_COUNT (source_char, pattern  
              [, position  
               [, match_parameter ]  
              ]  
            )
```

### Arguments

source\_char is the text expression to search.

pattern is the string to search for. A period matches any character. For a list of operators, refer to the *Oracle Database SQL Language Reference*, Appendix D, "Oracle Regular Expression Support."

position is a nonzero integer indicating the character of source\_char where the function begins the search. When position is negative, then the function counts and searches backward from the end of string. The default value of position is 1, which means that the function begins searching at the first character of source\_char.

match\_parameter is a text literal that lets you change the default matching behavior of the function. You can specify one or more of the following values:

- c: Case-sensitive matching.
- i: Case-insensitive matching.
- m: Treat the source string as multiple lines. The function interprets ^ and \$ as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. By default, the function treats the source string as a single line.
- n: New-line character is among the characters matched by a period (the wildcard character). By default, it is not.
- x: Ignore whitespace characters.

### Example

REGEXP\_COUNT('Mississippi', 'i', 1) searches the string Mississippi for the letter i, beginning the search at the first letter. It returns the value 4.

## REGEXP\_INSTR

REGEXP\_INSTR searches a string for a regular pattern. It can return an integer indicating either the beginning or the ending position of the matched substring. If no match is found, then the function returns 0.

The function evaluates strings using characters as defined by the input character set.

### Return Value

NUMBER

### Syntax

```
REGEXP_INSTR (source_char, pattern
             [, position
             [, occurrence
             [, return_option
             [, match_parameter ]
             ]
             ]
             )
```

### Arguments

source\_char is the text expression to search.

pattern is the string to search for. A period matches any character. For a list of operators, refer to the *Oracle Database SQL Language Reference*, Appendix D, Oracle Regular Expression Support.

position is a nonzero integer indicating the character of source\_char where the function begins the search. When position is negative, then the function counts and searches backward from the end of string. The default value of position is 1, which means that the function begins searching at the first character of source\_char.

occurrence is an integer indicating which occurrence of pattern the function should search for. The value of occurrence must be positive. The default values of occurrence is 1, meaning the function searches for the first occurrence of pattern.

return\_option is either 0 to return the position of the match (default), or 1 to return the position of the character following the match.

match\_parameter is a text literal that lets you change the default matching behavior of the function. You can specify one or more of the following values:

- c: Case-sensitive matching.
- i: Case-insensitive matching.
- m: Treat the source string as multiple lines. The function interprets ^ and \$ as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. By default, the function treats the source string as a single line.
- n: New-line character is among the characters matched by a period (the wildcard character). By default, it is not.
- x: Ignore whitespace characters.

**Example**

REGEXP\_INSTR('Mississippi', 'i', 1, 3) searches the string Mississippi for the third instance of the letter i, beginning the search at the first letter. It returns the value 8.



## REGEXP\_REPLACE

REGEXP\_REPLACE searches a string for a regular pattern and replaces it with another string. By default, the function returns `source_char` with every occurrence of the regular expression pattern replaced with `replace_string`.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
REGEXP_REPLACE(source_char, pattern
               [, replace_string
               [, position
               [, occurrence
               [, match_parameter ]
               ]
               ]
               )
```

### Arguments

`source_char` is the text expression that is searched.

`pattern` is the text expression to search for. It is usually a text literal and can contain up to 512 bytes. For a list of operators, refer to the *Oracle Database SQL Language Reference*, Appendix D, "Oracle Regular Expression Support."

`replace_string` is the text that replaces `pattern` in `source_char`.

`position` is a nonzero integer indicating the character of `source_char` where the function begins the search. When `position` is negative, then the function counts and searches backward from the end of string. The default value of `position` is 1, which means that the function begins searching at the first character of `source_char`.

`occurrence` is an integer indicating which occurrence of `pattern` the function should search for. The value of `occurrence` must be positive. The default values of `occurrence` is 1, meaning the function searches for the first occurrence of `pattern`.

`match_parameter` is a text literal that lets you change the default matching behavior of the function. You can specify one or more of the following values:

- `c`: Case-sensitive matching.
- `i`: Case-insensitive matching.
- `m`: Treat the source string as multiple lines. The function interprets `^` and `$` as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. By default, the function treats the source string as a single line.
- `n`: New-line character is among the characters matched by a period (the wildcard character). By default, it is not.
- `x`: Ignore whitespace characters.

**Example**

```
REGEXP_REPLACE('500 Oracle Parkway, Redwood Shores, CA', '( ){2,}', ' ')
eliminates extra spaces and returns the string
500 Oracle Parkway, Redwood Shores, CA
```

---

## REGEXP\_SUBSTR

REGEXP\_SUBSTR searches a string for a pattern and returns the matching string.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR

### Syntax

```
REGEXP_SUBSTR(source_char, pattern
              [, position
              [, occurrence
              [, match_parameter ]
              ]
              ]
              )
```

### Arguments

`source_char` is the text expression that is searched.

`pattern` is the text expression to search for. It is usually a text literal and can contain up to 512 bytes. For a list of operators, refer to the *Oracle Database SQL Language Reference*, Appendix D, "Oracle Regular Expression Support."

`position` is a nonzero integer indicating the character of `source_char` where the function begins the search. When `position` is negative, then the function counts and searches backward from the end of string. The default value of `position` is 1, which means that the function begins searching at the first character of `source_char`.

`occurrence` is an integer indicating which occurrence of `pattern` the function should search for. The value of `occurrence` must be positive. The default values of `occurrence` is 1, meaning the function searches for the first occurrence of `pattern`.

`match_parameter` is a text expression that lets you change the default matching behavior of the function. You can specify one or more of the following values:

- `c`: Case-sensitive matching.
- `i`: Case-insensitive matching.
- `m`: Treat the source string as multiple lines. The function interprets `^` and `$` as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. By default, the function treats the source string as a single line.
- `n`: New-line character is among the characters matched by a period (the wildcard character). By default, it is not.
- `x`: Ignore whitespace characters.

### Examples

REGEXP\_SUBSTR('7 W 96th St, New York, NEW YORK', 'new york', 10, 2, 'i') starts searching at the tenth character and matches NEW YORK in a case-insensitive match.

REGEXP\_SUBSTR('parsley, sage, rosemary, thyme', 's[^,]+e', 1, 2) starts searching at the first character and matches the second substring consisting of the

letter *s*, any number of characters that are not commas, and the letter *e*. In this example, the function returns the value *sage*.

## REMAINDER

REMAINDER returns a rounded remainder when one number is divided by another using this equation:

$$n2 - (n1 * N)$$

where N is the integer nearest  $n2/n1$ .

### Return Value

NUMBER

### Syntax

**REMAINDER**(n2, n1)

### Arguments

n1 is a numeric expression for the divisor.

n2 is a numeric expression for the dividend.

### Example

REMAINDER(18, 7) returns the value -3.

## REPLACE

REPLACE searches a string for a regular pattern, replaces it with another string, and returns the modified string.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
REPLACE(char, search_string [, replacement_string ])
```

### Arguments

char is the text expression that is searched.

search\_string is the text expression to search for.

replacement\_string is the text expression that replaces search\_string in char.

### Example

REPLACE('Nick Nack', 'N', 'Cl') returns the string Click Clack.

## ROUND (date)

ROUND returns a date rounded to the unit specified by the date format.

### Return Value

DATE

### Syntax

```
ROUND(date [, fmt ])
```

### Arguments

date is an expression that identifies a date and time.

fmt is a text literal with a date format, as described in the *Oracle Database SQL Language Reference*.

### Examples

ROUND(SYSDATE, 'YEAR') returns the value 01-JAN-07 for any day in the last half of 2006.

ROUND(TO\_DATE('13-OCT-06'), 'MONTH') returns the value 01-OCT-06.

---

## ROUND (number)

ROUND returns a number rounded to a specified number of places.

### Return Value

NUMBER

### Syntax

**ROUND**(n [, integer ])

### Arguments

n is the number to round.

integer is the number of decimal places of the rounded number. A negative value rounds to the left of the decimal point. The default value is 0.

### Examples

ROUND(15.193) returns the value 15.

ROUND(15.193,1) returns the value 15.2.

ROUND(15.193,-1) returns the value 20.



## ROWIDTOCHAR

ROWIDTOCHAR converts a row address from a ROWID data type to text. The return value is always 18 characters long in the database character set.

### Return Value

VARCHAR2

### Syntax

**ROWIDTOCHAR**(rowid)

### Arguments

rowid is a row address to convert.

## ROWIDTONCHAR

ROWIDTONCHAR converts a row address from the ROWID data type to text. The return value is always 18 characters in the national character set.

### Return Value

NVARCHAR2

### Syntax

**ROWIDTONCHAR**(rowid)

### Arguments

rowid is a row address to convert.

---

## RPAD

RPAD adds characters to the right of an expression to a specified length. The data type of the return value is the same as the original text.

Use LPAD to add characters to the left.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
RPAD(expr1 , n [, expr2 ])
```

### Arguments

*expr1* is a text expression for the base string.

*n* is the total length of the returned expression. If *expr1* is longer than *n*, then this function truncates *expr1* to *n* characters.

*expr2* is a text expression for the padding characters. By default, it is a space.

### Example

RPAD('Stay tuned', 15, '. ') returns the value Stay tuned. . .

RPAD('Stay tuned', 4) returns the value Stay.

## RTRIM

RTRIM scans a text expression from right to left and removes all the characters that match the characters in the trim expression, until it finds an unmatched character. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
RTRIM(char [, set ])
```

### Arguments

char is the text expression to be trimmed.

set is a text expression with the characters to remove. The default value of set is a single blank.

### Examples

RTRIM('You did that!?!?!?!', '?!') returns the value You did that.

RTRIM('3.14848', '84') returns the text value 3.1.

## SESSIONTIMEZONE

SESSIONTIMEZONE returns the time zone of the current session. The return type is a time zone offset from Coordinated Universal Time (UTC) or a time zone region name.

### Return Value

VARCHAR2

### Syntax

**SESSIONTIMEZONE**

### Example

SESSIONTIMEZONE returns the value -05:00 for Eastern Standard Time.

## SIGN

SIGN returns a value that indicates whether a specified number is less than, equal to, or greater than 0 (zero):

- -1 if  $n < 0$
- 0 if  $n = 0$
- 1 if  $n > 0$

### Return Value

NUMBER

### Syntax

**SIGN**(n)

### Arguments

n is a numeric expression.

### Example

SIGN(-15) returns the value -1.

## SIN

SIN returns the sine of an angle.

### Return Value

NUMBER

### Syntax

**SIN**(n)

### Arguments

n is a numeric expression for an angle in radians.

### Example

`SIN(30 * 3.1415927/180)` calculates the sine of a 30 degrees angle as the value 0.500000007. The numeric expression converts degrees to radians.

## SINH

SINH returns the sine of a hyperbolic angle.

### Return Value

NUMBER

### Syntax

**SINH**(n)

### Arguments

n is a numeric expression for a hyperbolic angle.

### Example

SINH(1) returns the value 1.17520119.



## SOUNDEX

SOUNDEX returns a character string containing the phonetic representation of a text expression. This function lets you compare words that are spelled differently but sound alike.

The function is based on the algorithm described in Donald Knuth's *The Art of Computer Programming*. This algorithm was designed specifically for English. Its results for other languages other than English are unpredictable and typically unsatisfactory.

### Return Value

VARCHAR2

### Syntax

**SOUNDEX** (char)

### Arguments

char can be any text expression.

### Example

All of these examples return the value D500:

```
soundex('Donna')
```

```
soundex('Diane')
```

```
soundex('Dana')
```

## SQRT

SQRT returns the square root of a number.

### Return Value

NUMBER

### Syntax

**SQRT**(n)

### Arguments

n is a numeric expression for a positive number.

### Example

SQRT(13) returns the value 3.60555128.

---

## SUBSTR

SUBSTR returns a portion of string, beginning at a specified character position and extending a specified number of characters.

- SUBSTR calculates lengths using characters as defined by the input character set.
- SUBSTRB uses bytes instead of characters.
- SUBSTRC uses Unicode complete characters.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

```
{ SUBSTR | SUBSTRB | SUBSTRC }(char, position [, substring_length ])
```

### Arguments

char is a text expression that provides the base string from which the substring is derived.

position identifies the first character of the substring:

- When position is positive, then the function counts from the beginning of char to find the first character.
- When position is negative, then the function counts backward from the end of char.
- When position is 0 (zero), then the first character is the beginning of the substring.

substring\_length is the number of characters in the returned string. By default, the function returns all characters to the end of the base string. If you specify a value less than 1, then the function returns a null.

### Examples

SUBSTR('firefly', 1, 4) returns the substring fire.

SUBSTR('firefly', -3, 3) returns the substring fly.

## SYS\_CONTEXT

`SYS_CONTEXT` returns the value of an attribute of a named context. The context, attribute, and value must already be defined in the database. If the context is not defined, `SYS_CONTEXT` returns `NULL`.

### Return Value

VARCHAR2

### Syntax

```
SYS_CONTEXT ('namespace', 'parameter')
```

### Arguments

`namespace` can be any named context in the database. `USERENV` is a built-in context that describes the current session.

`parameter` is a defined attribute of `namespace`. [Table 3–3](#) describes the predefined attributes of `USERENV` that are most likely to have values. For a complete list, refer to the `SYS_CONTEXT` entry in the *Oracle Database SQL Language Reference*.

**Table 3–3 USERENV Attributes**

USERENV Attribute	Description
AUTHENTICATED_IDENTITY	The identity used for authentication, such as database user name, schema name, or operating system login name.
AUTHENTICATION_METHOD	The method of authentication, such as PASSWORD, OS, or SSL.
CURRENT_EDITION_ID	The session edition identifier, such as 100.
CURRENT_EDITION_NAME	The session edition name, such as ORA\$BASE.
CURRENT_SCHEMA	The name of the currently active default schema, such as SH.
CURRENT_SCHEMA_ID	The numeric identifier of the currently active default schema, such as 80.
CURRENT_USER	The name of the database user whose privileges are currently active, such as SH.
CURRENT_USERID	The numeric identifier of the database user whose privileges are currently active, such as 80.
DATABASE_ROLE	Data Guard role of the database: PRIMARY, PHYSICAL STANDBY, LOGICAL STANDBY, or SNAPSHOT STANDBY.
DB_DOMAIN	The network domain of the database as specified by the DB_DOMAIN initialization parameter, such as us.example.com.
DB_NAME	The name of the database as specified by the DB_NAME initialization parameter.
DB_UNIQUE_NAME	The unique name of the database within the domain as specified by the DB_UNIQUE_NAME initialization parameter.
ENTERPRISE_IDENTITY	The enterprise-wide identity of the user, or NULL for local users, SYSDBA, and SYSOPER.
FG_JOB_ID	Job identifier of the current session if a client foreground process opened it; otherwise, NULL.

**Table 3–3 (Cont.) USERENV Attributes**

<b>USERENV Attribute</b>	<b>Description</b>
GLOBAL_CONTEXT_MEMORY	The number used in the System Global Area by the globally accessed context.
GLOBAL_UID	The global user identification from Oracle Internet Directory for Enterprise User Security logins; otherwise, NULL.
HOST	The name of the client host computer.
IDENTIFICATION_TYPE	The way the user schema was created in the database: LOCAL, EXTERNAL, GLOBAL_SHARED, or GLOBAL_PRIVATE.
INSTANCE	The identification number of the current instance, such as 1.
INSTANCE_NAME	The name of the database instance.
IP_ADDRESS	The IP address of the client, such as 10.255.255.255.
ISDBA	TRUE if the user was authenticated with DBA privileges; otherwise, FALSE.
LANG	A short name for the session language, such as US for AMERICAN.
LANGUAGE	The language, territory, and database character set in the form <i>language_territory.characterset</i> , such as AMERICA_AMERICAN.WE8DEC.
MODULE	The application name set through the DBMS_APPLICATION_INFO package or OCI, such as JDBC Thin Client or SQL Developer.
NETWORK_PROTOCOL	The network protocol being used for communication, such as TCP.
NLS_CALENDAR	The session calendar, such as GREGORIAN.
NLS_CURRENCY	The session currency mark, such as \$.
NLS_DATE_FORMAT	The session date format, such as DD-MON-RR.
NLS_DATE_LANGUAGE	The session date language, such as AMERICAN.
NLS_SORT	BINARY or a linguistic sort basis, such as XSPANISH.
NLS_TERRITORY	The session territory, such as AMERICA.
OS_USER	The operating system user name of the client process that initiated the database session.
SERVER_HOST	The host name of the computer where the database instance is running.
SERVICE_NAME	The name of the service the session is connected to, such as SYS\$USERS.
SESSION_USER	The database user name or schema name that identified the user at login, such as SH.
SESSIONID	The session identifier, such as 120456.
SID	The session number, such as 86.

## Example

`SYS_CONTEXT('USERENV', 'NLS_DATE_FORMAT')` returns a value such as DD-MON-RR.

## SYSDATE

SYSDATE returns the current date and time of the operating system on which the database resides. The format of the value depends on the value of the NLS\_DATE\_FORMAT initialization parameter.

### Return Value

DATE

### Syntax

**SYSDATE**

### Examples

SYSDATE returns a value such as 13-AUG-06 with NLS\_DATE\_FORMAT set to DD-MON-RR.

TO\_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') returns a value such as 08-13-2006 17:20:47. The date format provided in the call to TO\_CHAR displays both the date and the time.

## SYSTIMESTAMP

SYSTIMESTAMP returns the system date, including fractional seconds and time zone, of the system on which the database resides.

### Return Value

TIMESTAMP WITH TIME ZONE

### Syntax

**SYSTIMESTAMP**

### Example

SYSTIMESTAMP returns a value such as  
13-AUG-06 05.28.10.385799 PM -08:00.

## TAN

TAN returns the tangent of an angle.

### Return Value

NUMBER

### Syntax

**TAN**(n)

### Arguments

n is a numeric expression for an angle in radians.

### Example

TAN(135 \* 3.1415927/180) calculates the tangent of a 135 degree angle as the value -0.99999993. The expression converts degrees to radians.



## TANH

TANH returns the tangent of a hyperbolic angle.

### Return Value

NUMBER

### Syntax

**TANH** (n)

### Arguments

n is a numeric expression for a hyperbolic angle.

### Example

TANH (.5) returns the value 0.462117157.

## TO\_BINARY\_DOUBLE

TO\_BINARY\_DOUBLE converts a text or numeric expression to a double-precision floating-point number.

### Return Value

BINARY\_DOUBLE

### Syntax

```
TO_BINARY_DOUBLE (expr [, fmt [, 'nlsparam' ] ])
```

### Arguments

expr can be any text or numeric expression.

fmt is a text expression that identifies a number format model as described in the *Oracle Database SQL Language Reference*.

nlsparam specifies the characters used by these number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument has the format shown here:

```
'NLS_NUMERIC_CHARACTERS = 'dg'  
NLS_CURRENCY = 'text'  
NLS_ISO_CURRENCY = territory '
```

The d is the decimal character, and the g is the group separator. They must be different single-byte characters. Within the quoted string, use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit this argument or any of the NLS parameters, then this function uses the default values for your session to interpret expr.

### Example

All of these examples return the value 1.235E+003:

```
TO_BINARY_DOUBLE(1234.56)
```

```
TO_BINARY_DOUBLE('$1,234.56', '$9,999.99')
```

```
TO_BINARY_DOUBLE('1.234,56', '9G999D99', 'NLS_NUMERIC_CHARACTERS=','.')')
```

---

## TO\_BINARY\_FLOAT

TO\_BINARY\_FLOAT converts a text or numeric expression to a single-precision floating-point number.

### Return Value

BINARY\_FLOAT

### Syntax

```
TO_BINARY_FLOAT (expr [, fmt [, 'nlsparam' ] ])
```

### Arguments

*n* can be any text or numeric expression.

*fmt* is a text expression that identifies a number format model as described in the *Oracle Database SQL Language Reference*.

*nlsparam* specifies the characters used by these number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument has the format shown here:

```
'NLS_NUMERIC_CHARACTERS = 'dg''
NLS_CURRENCY = 'text'
NLS_ISO_CURRENCY = territory '
```

The *d* is the decimal character, and the *g* is the group separator. They must be different single-byte characters. Within the quoted string, use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit this argument or any of the NLS parameters, then this function uses the default values for your session to interpret *expr*.

### Examples

All of these examples return the value 1.235E+003:

```
TO_BINARY_FLOAT(1234.56)
```

```
TO_BINARY_FLOAT('$1,234.56', '$9,999.99')
```

```
TO_BINARY_FLOAT('1.234,56', '9G999D99', 'NLS_NUMERIC_CHARACTERS=','.')')
```

## TO\_CHAR (character)

TO\_CHAR(character) converts a text expression to the database character set.

### Return Value

VARCHAR2

### Syntax

**TO\_CHAR**(exp)

### Arguments

char is a text expression. If it contains characters that are not represented in the database character set, then the conversion results in a loss of data.

### Examples

TO\_CHAR('¿Una qué sorpresa!') returns the value ?Una qu? sorpresa! Two letters are lost in the conversion (¿ and é) because they are not in the database character set.

TO\_CHAR('David Ortiz') returns the value David Ortiz in the database character set. No characters are lost in this conversion because all of them are in the database character set.

---

## TO\_CHAR (datetime)

TO\_CHAR(datetime) converts a datetime or interval expression to a text string in a specified format.

### Return Value

VARCHAR2

### Syntax

```
TO_CHAR({ datetime | interval } [, fmt [, 'nlsparam' ] ])
```

### Arguments

datetime is a datetime expression to be converted to text.

interval is an interval expression to be converted to text.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the NLS\_TERRITORY initialization parameter or can be set explicitly by the NLS\_DATE\_FORMAT parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

nlsparam specifies the language in which month and day names and abbreviations are returned. This argument can have this form:

```
'NLS_DATE_LANGUAGE = language'
```

By default, the return value is in the session date language.

### Examples

TO\_CHAR(SYSDATE) returns a value such as 11-APR-08.

TO\_CHAR(SYSDATE, 'Day: MONTH DD, YYYY') returns a value such as Friday : APRIL 11, 2008.

TO\_CHAR(SYSDATE, 'Day: MONTH DD, YYYY', 'NLS\_DATE\_LANGUAGE = Spanish') returns a value such as Viernes : ABRIL 11, 2008.

## TO\_CHAR (number)

TO\_CHAR (number) converts a numeric expression to a text value in the database character set.

### Return Value

VARCHAR2

### Syntax

```
TO_CHAR (n [, fmt [, 'nlsparam' ] ])
```

### Arguments

n is a numeric expression to be converted.

fmt is a text expression that identifies a number format model as described in the *Oracle Database SQL Language Reference*.

nlsparam specifies the characters that are returned by these number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument has the format shown here:

```
'NLS_NUMERIC_CHARACTERS = 'dg'  
NLS_CURRENCY = 'text'  
NLS_ISO_CURRENCY = territory '
```

The characters d and g represent the decimal character and group separator, respectively. They must be different single-byte characters. Within the quoted string, use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit this argument or any of the NLS parameters, then this function uses the default values for your session.

### Examples

TO\_CHAR (1234567, 'C9G999G999D99') returns a text string such as USD1,234,567.00.

TO\_CHAR (1234567, 'C9G999G999D99', 'NLS\_ISO\_CURRENCY = SPAIN') returns the text string EUR1,234,567.00.

---

## TO\_DATE

TO\_DATE converts a text expression to a DATE data type.

### Return Value

DATE

### Syntax

```
TO_DATE(char [, fmt [, 'nlsparam' ] ])
```

### Arguments

char is a text expression that represents a date.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the NLS\_TERRITORY initialization parameter or can be set explicitly by the NLS\_DATE\_FORMAT parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

nlsparam specifies the language of char. This argument can have this form:

```
'NLS_DATE_LANGUAGE = language'
```

By default, char is in the session date language.

### Examples

TO\_DATE('October 13, 2008', 'MONTH DD, YYYY') returns the value 13-OCT-08.

TO\_DATE('13 Octubre 2008', 'dd month yyyy', 'NLS\_DATE\_LANGUAGE=SPANISH') also returns the value 13-OCT-08

## TO\_DSINTERVAL

TO\_DSINTERVAL converts a text expression to an INTERVAL DAY TO SECOND data type.

### Return Value

INTERVAL DAY TO SECOND

### Syntax

**TO\_DSINTERVAL**(char)

### Arguments

char is a text expression to be converted.

### Example

TO\_DSINTERVAL('360 12:45:49') returns the value +360 12:45:49.000000.



## TO\_NCHAR (character)

TO\_NCHAR(character) converts a character string to the national character set.

### Return Value

NVARCHAR2

### Syntax

**TO\_NCHAR**(exp)

### Arguments

exp is a text expression. If it contains characters that are not represented in the national character set, then the conversion results in a loss of data.

### Example

TO\_NCHAR('David Ortiz') returns the value David Ortiz in the national character set.

## TO\_NCHAR (datetime)

TO\_NCHAR(datetime) converts a datetime or interval value to the national character set.

### Return Value

NVARCHAR2

### Syntax

```
TO_NCHAR({ datetime | interval }  
         [, fmt [, 'nlsparam' ] ]  
         )
```

### Arguments

datetime is a datetime expression to be converted to text.

interval is an interval expression to be converted to text.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the NLS\_TERRITORY initialization parameter or can be set explicitly by the NLS\_DATE\_FORMAT parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

nlsparam specifies the language in which month and day names and abbreviations are returned. This argument can have this form:

```
'NLS_DATE_LANGUAGE = language'
```

By default, the return value is in the session date language.

### Examples

TO\_NCHAR(SYSDATE) returns a value such as 11-APR-08.

TO\_NCHAR(SYSDATE, 'Day: MONTH DD, YYYY') returns a value such as Friday :  
APRIL 11, 2008.

TO\_NCHAR(SYSDATE, 'Day: MONTH DD, YYYY', 'NLS\_DATE\_LANGUAGE = Spanish')  
returns a value such as Viernes : ABRIL 11, 2008.

---

## TO\_NCHAR (number)

TO\_NCHAR (number) converts a number to the national character set.

### Return Value

NVARCHAR2

### Syntax

```
TO_CHAR(n [, fmt [, 'nlsparam' ] ])
```

### Arguments

n is a numeric expression to be converted.

fmt is a text expression that identifies a number format model as described in the *Oracle Database SQL Language Reference*.

nlsparam is a text expression that specifies the characters that are returned by these number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument has the format shown here:

```
'NLS_NUMERIC_CHARACTERS = 'dg'  
  NLS_CURRENCY = 'text'  
  NLS_ISO_CURRENCY = territory '
```

The characters d and g represent the decimal character and group separator, respectively. They must be different single-byte characters. Within the quoted string, use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit this argument or any of the NLS parameters, then this function uses the default values for your session.

### Examples

TO\_NCHAR(1234567, 'C9G999G999D99') returns a text string such as USD1,234,567.00.

TO\_NCHAR(1234567, 'C9G999G999D99', 'NLS\_ISO\_CURRENCY = SPAIN') returns the text string EUR1,234,567.00.

## TO\_NUMBER

TO\_NUMBER converts a text expression containing a number to a value of NUMBER data type.

### Return Value

NUMBER

### Syntax

```
TO_NUMBER(expr [, fmt [, 'nlsparam' ] ])
```

### Arguments

*expr* is an expression to be converted to a number.

*fmt* is a text expression that identifies a number format model as described in the *Oracle Database SQL Language Reference*.

*nlsparam* specifies the characters used by these number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument has the format shown here:

```
'NLS_NUMERIC_CHARACTERS = 'dg'  
NLS_CURRENCY = 'text'  
NLS_ISO_CURRENCY = territory '
```

The *d* is the decimal character, and the *g* is the group separator. They must be different single-byte characters. Within the quoted string, use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit this argument or any of the NLS parameters, then this function uses the default values for your session to interpret *expr*.

### Examples

TO\_NUMBER('\$1,234,567.89', 'L999G999G999D99') returns the value 1234567.89.

TO\_NUMBER('EUR1,234,567.89', 'C999G999G999D99', 'NLS\_ISO\_CURRENCY=SPAIN') also returns the value 1234567.89.

---

## TO\_TIMESTAMP

TO\_TIMESTAMP converts a text expression to a value of TIMESTAMP.

### Return Value

TIMESTAMP

### Syntax

```
TO_TIMESTAMP(char [, fmt [ 'nlsparam' ] ])
```

### Arguments

char is a text expression to be converted.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the NLS\_TERRITORY initialization parameter or can be set explicitly by the NLS\_DATE\_FORMAT parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

nlsparam specifies the language in which month and day names and abbreviations given in char. This argument has this form:

```
'NLS_DATE_LANGUAGE = language'
```

By default, char is in the session date language.

### Examples

```
TO_TIMESTAMP('10-SEP-0614:10:10.123000', 'DD-MON-RRHH24:MI:SS.FF') returns  
the value 10-SEP-06 02.10.10.123000 PM.
```

```
TO_TIMESTAMP('10-AGOSTO-0714:10:10', 'DD-MON-RRHH24:MI:SS.FF', 'NLS_DATE_  
LANGUAGE=SPANISH') returns the value 10-AUG-07 02.10.10.000000 PM.
```

## TO\_TIMESTAMP\_TZ

TO\_TIMESTAMP\_TZ converts a text expression to a value of `TIMESTAMPWITHTIMEZONE` data type.

### Return Value

TIMESTAMP WITH TIME ZONE

### Syntax

```
TO_TIMESTAMP_TZ(char [, fmt [ 'nlsparam' ] ])
```

### Arguments

char is a text expression to be converted.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the `NLS_TERRITORY` initialization parameter or can be set explicitly by the `NLS_DATE_FORMAT` parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

nlsparam specifies the language in which month and day names and abbreviations given in char. This argument has this form:

```
'NLS_DATE_LANGUAGE = language'
```

By default, char is in the session date language.

### Examples

```
TO_TIMESTAMP_TZ('2006-03-26 7:33:00 -4:00', 'YYYY-MM-DD HH:MI:SS TZH:TZM')  
returns the value 26-MAR-06 07.33.00.000000 AM -04:00.
```

```
TO_TIMESTAMP_TZ('2006-AGOSTO-13 7:33:00 -4:00', 'YYYY-MONTH-DD HH:MI:SS  
TZH:TZM', 'NLS_DATE_LANGUAGE=SPANISH') returns the value 13-AUG-06  
07.33.00.000000 AM -04:00.
```

---

## TO\_YMINTERVAL

TO\_YMINTERVAL converts a text expression to an INTERVAL YEAR TO MONTH data type. The function accepts argument in one of the two formats:

- SQL interval format compatible with the SQL standard (ISO/IEC 9075:2003)
- ISO duration format compatible with the ISO 8601:2004 standard

### Return Value

INTERVAL YEAR TO MONTH

### Syntax

```
TO_YMINTERVAL ( ' { sql_format | ym_iso_format } ' )
```

```
sql_format ::=
[+|-] years - months
```

```
ym_iso_format ::=
[-] P [ years Y ] [months M ] [days D ] [ T [hours H ] [minutes M ] [seconds [.
frac_secs] S ] ]
```

### Arguments

#### *In SQL format:*

years is an integer between 0 and 999999999

months is an integer between 0 and 11.

Additional blanks are allowed between format elements.

#### *In ISO format:*

years and months are integers between 0 and 999999999.

days, hours, minutes, seconds, and frac\_secs are nonnegative integers and are ignored.

No blanks are allowed in the value.

### Examples

TO\_YMINTERVAL('1-6') and TO\_YMINTERVAL('P1Y6M') return the value +01-06 for 1 year and 6 months.

SYSDATE + TO\_YMINTERVAL('1-6') adds one year and six months to the current date. When SYSDATE is 15-APR-08, the value is 15-OCT-09.

SYSDATE + TO\_YMINTERVAL('P1Y6M') adds one year and six months to the current date using ISO format. When SYSDATE is 15-APR-08, the value is 15-OCT-09.

SYSDATE + TO\_YMINTERVAL('-1-2') subtracts one year and two months from the current date. When SYSDATE is 15-APR-08, the value is 15-FEB-07.

## TRANSLATE

TRANSLATE enables you to make several single-character, one-to-one substitutions in one operation. This expression returns an expression with all occurrences of each character in one string replaced by its corresponding character in a second string.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

**TRANSLATE**(*expr*, *from\_string*, *to\_string*)

### Arguments

*expr* is a text expression to be modified.

*from\_string* consists of one or more characters to be replaced in *expr*.

*to\_string* consists of one or more characters that replace the corresponding characters in *from\_string*. This string can be shorter than *from\_string*, so that a null replaces the extra characters. However, *to\_string* cannot be empty.

### Example

TRANSLATE('disk', 'dk', 'Dc') returns the value Disc.



---

## TRANSLATE (USING)

TRANSLATE converts a text string between the database character set and the national character set.

**Note:** The TRANSLATE USING function is supported primarily for ANSI compatibility. Oracle recommends that you use the TO\_CHAR and TO\_NCHAR functions for converting data to the database or national character sets. TO\_CHAR and TO\_NCHAR can take as arguments a greater variety of data types than TRANSLATE USING, which accepts only character data.

### Return Value

VARCHAR2 | NVARCHAR2

### Syntax

```
TRANSLATE (char USING { CHAR_CS | NCHAR_CS } )
```

### Arguments

char is a text expression to be converted to the database character set (USING CHAR\_CS) or the national character set (USING NCHAR\_CS).

### Example

TRANSLATE('north by northwest' USING NCHAR\_CS) returns the value north by northwest in the national character set.

## TRIM

TRIM removes leading or trailing characters (or both) from a character string.

### Return Value

VARCHAR2

### Syntax

```
TRIM([ { { LEADING | TRAILING | BOTH } [ trim_character ]
      | trim_character
      }
      FROM
      ]
      trim_source
      )
```

### Arguments

LEADING removes matching characters from the beginning of the string.

TRAILING removes matching characters from the end of the string.

BOTH removes matching characters from both the beginning and the end of the string.  
(Default)

trim\_character is a single character to be removed. By default, it is a space.

trim\_source is the text expression to be trimmed.

### Examples

TRIM('0' FROM '00026501.6703000') returns the value 26501.6703.

TRIM(LEADING '!' FROM '!!Help! Help!!') returns the value Help! Help!.

## TRUNC (number)

TRUNC shortens a numeric expression to a specified number of decimal places.

### Return Value

NUMBER

### Syntax

**TRUNC**(n1 [, n2 ])

### Arguments

n1 is the numeric expression to be truncated.

n2 is the number of decimal places. A positive number truncates digits to the right of the decimal point, and a negative number replaces digits to the left of the decimal point. The default value is zero (0).

### Examples

TRUNC(15.79) returns the value 15.

TRUNC(15.79, 1) returns the value 15.7.

TRUNC(15.79, -1) returns the value 10.

## TZ\_OFFSET

TZ\_OFFSET returns the time zone offset from Coordinated Universal Time (UTC).

### Return Value

VARCHAR2

### Syntax

```
TZ_OFFSET( { 'time_zone_name' | '{ + | - } hh:mi'  
            | SESSIONTIMEZONE | DBTIMEZONE  
            }  
          )
```

### Arguments

time\_zone\_name is the name of a time zone.

hh:mm are hours and minutes. This argument simply returns itself.

SESSIONTIMEZONE returns the session time zone.

DBTIMEZONE returns the database time zone.

### Examples

TZ\_OFFSET('US/Eastern') returns the offset -04:00 during Daylight Savings Time.

TZ\_OFFSET('EST') returns the offset -05:00.

TZ\_OFFSET(DBTIMEZONE) returns the offset -07:00 for Mountain Standard Time.

## UID

UID returns a unique identifier (UID) for the session user (the user who logged on).

### Return Value

INTEGER

### Syntax

**UID**

### Example

UID returns a value such as 76.

## UNISTR

UNISTR converts a text string to the national character set.

### Return Value

NVARCHAR2

### Syntax

```
UNISTR ( string )
```

### Arguments

*string* can be any text expression. For portability, Oracle recommends using only ASCII characters and Unicode encoding values as text literals. A Unicode encoding value has the form `\xxxx` where *xxxx* is the hexadecimal value of a character. Supplementary characters are encoded as two code units, the first from the high-surrogates range (U+D800 to U+DBFF), and the second from the low-surrogates range (U+DC00 to U+DFFF). To include a literal backslash in the string, precede it with another backslash (`\\`).

### Example

`UNISTR('abc\00e5\00f1\00f6')` returns the value `abcåñö`.

## UPPER

UPPER converts all alphabetic characters in a text expression to uppercase. The data type of the return value is the same as the original text.

### Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

### Syntax

**UPPER**(char)

### Arguments

char can be any text expression.

### Example

UPPER('This is an emergency') returns the string THIS IS AN EMERGENCY.

---

## USER

USER returns the name of the session user (the user who logged on).

### Return Value

VARCHAR2

### Syntax

**USER**

### Example

USER returns a value such as GLOBAL.



## VSIZE

VSIZE returns the number of bytes in the internal representation of an expression. It returns NULL for a null expression.

### Return Value

NUMBER

### Syntax

**VSIZE** (expr)

### Arguments

expr can be an expression of any data type.

### Example

VSIZE('Sound of thunder') returns the value 16.

VSIZE(CHANNEL.LONG\_DESCRIPTION) returns the following values:

<b>Channel</b>	<b>VSIZE</b>
Catalog	7
Direct Sales	12
Internet	8

## WIDTH\_BUCKET

`WIDTH_BUCKET` enables you to construct a histogram range divided into intervals of identical size. The function returns the bucket number into which the value of an expression falls.

When needed, `WIDTH_BUCKET` creates an underflow bucket numbered 0 and an overflow bucket numbered `num_buckets+1`. These buckets handle values outside the specified range and are helpful in checking the reasonableness of the end points.

### Return Value

NUMBER

### Syntax

```
WIDTH_BUCKET  
(expr, min_value, max_value, num_buckets)
```

### Arguments

`expr` is the expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value. If `expr` evaluates to null, then the function returns NULL.

`min_value` and `max_value` are expressions for the end points of the acceptable range for `expr`. Both of these expressions must evaluate to numeric or datetime values, and neither can evaluate to null.

`num_buckets` is an expression for the number of buckets. This expression must evaluate to a positive integer.

### Example

`WIDTH_BUCKET(13, 0, 20, 4)` returns the value 3. It creates four buckets from 0 to 20 and sorts the value 13 into bucket 3.

`WIDTH_BUCKET(-5, 0, 20, 4)` returns the value 0. The value -5 is below the beginning of the range.

---

---

## Reserved Words

This appendix lists the reserved words for the OLAP expression syntax.

### Reserved Words

The following are reserved words for the OLAP Expression Syntax. You should not use them or Oracle Database reserved words as object names. However, if you do, be sure to enclose them in double quotes in the expression syntax.

Refer to the *Oracle Database SQL Language Reference* for a list of Oracle Database reserved words.

AGGREGATE  
AGGREGATES  
ALL  
ALLOW  
ANALYZE  
ANCESTOR  
AND  
ANY  
AS  
ASC  
AT  
AVG  
BETWEEN  
BINARY\_DOUBLE  
BINARY\_FLOAT  
BLOB  
BRANCH  
BUILD  
BY  
BYTE  
CASE  
CAST  
CHAR  
CHILD  
CLEAR  
CLOB  
COMMIT  
COMPILE  
CONSIDER  
COUNT  
DATATYPE

DATE  
DATE\_MEASURE  
DAY  
DECIMAL  
DELETE  
DESC  
DESCENDANT  
DIMENSION  
DISALLOW  
DIVISION  
DML  
ELSE  
END  
ESCAPE  
EXECUTE  
FIRST  
FLOAT  
FOR  
FROM  
HIERARCHIES  
HIERARCHY  
HOUR  
IGNORE  
IN  
INFINITE  
INSERT  
INTEGER  
INTERVAL  
INTO  
IS  
LAST  
LEAF\_DESCENDANT  
LEAVES  
LEVEL  
LEVELS  
LIKE  
LIKEC  
LIKE2  
LIKE4  
LOAD  
LOCAL  
LOG\_SPEC  
LONG  
MAINTAIN  
MAX  
MEASURE  
MEASURES  
MEMBER  
MEMBERS  
MERGE  
MLSLABEL  
MIN  
MINUTE  
MODEL  
MONTH

---

NAN  
NCHAR  
NCLOB  
NO  
NONE  
NOT  
NULL  
NULLS  
NUMBER  
NVARCHAR2  
OF  
OLAP  
OLAP\_DML\_EXPRESSION  
ON  
ONLY  
OPERATOR  
OR  
ORDER  
OVER  
OVERFLOW  
PARALLEL  
PARENT  
PLSQL  
PRUNE  
RAW  
RELATIVE  
ROOT\_ANCESTOR  
ROWID  
SCN  
SECOND  
SELF  
SERIAL  
SET  
SOLVE  
SOME  
SORT  
SPEC  
SUM  
SYNCH  
TEXT\_MEASURE  
THEN  
TIME  
TIMESTAMP  
TO  
UNBRANCH  
UPDATE  
USING  
VALIDATE  
VALUES  
VARCHAR2  
WHEN  
WHERE  
WITHIN  
WITH  
YEAR

ZERO  
ZONE

## Special Symbols

[Table A-1](#) lists symbols that are used in the OLAP expression syntax. To enter them as literal characters, enclose them in quotes the same as any other literal text.

**Table A-1** *OLAP Expression Syntax Symbols*

<b>Symbol</b>	<b>Description</b>
,	Comma
	Concatenate
/	Divide
=	Equals
>	Greater than
>=	Greater than or equal to
[	Left bracket
(	Left parenthesis
<	Less than
<=	Less than or equal to
-	Minus
!=	Not equal to
<>	Not equal to
^=	Not equal to
+	Plus
]	Right bracket
)	Right parenthesis
*	Star

## A

---

ancestor function, 2-14  
AND conjunction, 1-8  
arguments, 1-13  
average function, 2-6  
AVERAGE\_RANK, 2-4  
AVG, 2-6

## B

---

BETWEEN operator, 1-8  
binary operators, 1-4  
BINARY\_DOUBLE, 1-2  
BINARY\_FLOAT, 1-2

## C

---

CASE expressions, 1-10  
CHAR, 1-2  
character literal, 1-10  
child count function, 2-16  
comparison operators, 1-6  
concatenation operator, 1-5  
conditions, 1-6  
conjunctions, 1-8  
COUNT, 2-9  
counting functions, 2-9, 2-16  
cumulative averages, 2-6

## D

---

data types, 1-2  
DATE, 1-2  
DECIMAL, 1-2  
DENSE\_RANK, 2-12  
depth function, 2-18  
difference from future period, 2-37  
difference from prior period, 2-30, 2-32  
DML expressions, 2-49

## F

---

FLOAT, 1-2  
future period functions, 2-34, 2-37, 2-40

## G

---

group comparison operators, 1-7

## H

---

HIER\_ANCESTOR, 2-14  
HIER\_CHILD\_COUNT, 2-16  
HIER\_DEPTH, 2-18  
HIER\_LEVEL, 2-20  
HIER\_ORDER, 2-22  
HIER\_PARENT, 2-24  
HIER\_TOP, 2-26

## I

---

identifiers, 1-1  
IF ... THEN ... ELSE, 1-10  
INTEGER, 1-2  
INTERVAL, 1-2  
IS operator, 1-9

## L

---

LAG, 2-28  
LAG\_VARIANCE, 2-30  
LAG\_VARIANCE\_PERCENT, 2-32  
largest values, 2-43  
later period functions, 2-34, 2-37, 2-40  
LEAD, 2-34  
LEAD\_VARIANCE, 2-37  
LEAD\_VARIANCE\_PERCENT, 2-40  
level depth function, 2-18  
level function, 2-20  
LIKE operators, 1-9  
literal expressions, 1-10

## M

---

MAX, 2-43  
MIN, 2-46  
moving averages, 2-6

## N

---

naming conventions, 1-1  
NCHAR, 1-2

NOT operator, 1-8  
NUMBER, 1-3  
NVARCHAR2, 1-3

windowing functions, 2-6, 2-9, 2-43, 2-46, 2-56

---

## O

object names, 1-1  
OLAP\_DML\_EXPRESSION, 2-49  
OR conjunction, 1-8  
ordering function, 2-22

---

## P

parent function, 2-24  
percent difference from future period, 2-40  
percent difference from prior period, 2-32  
period-to-date function, 2-56  
prior period functions, 2-28, 2-30, 2-32

---

## Q

qualified data references, 1-13  
quotes  
    double, for reserved words, A-1  
    in naming conventions, 1-1  
    single, for text literals, 1-10

---

## R

RANK, 2-50  
ranking functions, 2-4, 2-12, 2-50, 2-52  
ratios, 2-54  
ROW\_NUMBER, 2-52

---

## S

SHARE, 2-54  
smallest values, 2-46  
sorting functions, 2-4, 2-12, 2-22, 2-50, 2-52  
SPL expressions, 2-49  
string, 1-10  
SUM, 2-56

---

## T

text literal, 1-10  
TIMESTAMP, 1-3  
top function, 2-26

---

## U

unary operators, 1-4

---

## V

VARCHAR2, 1-3  
variance functions, 2-30, 2-32, 2-37, 2-40

---

## W

WHEN ... THEN, 1-10