2 OLAP Functions

OLAP Functions in Alphabetical Order 2-1
OLAP Functions By Category 2-1
   Aggregate Functions 2-2
   Analytic Functions 2-2
   Hierarchical Functions 2-2
   Lag Functions 2-2
   OLAP DML Functions 2-3
   Rank Functions 2-3
Share Functions 2-3
Window Functions 2-3

AVERAGE_RANK 2-3
AVG 2-5
COUNT 2-7
DENSE_RANK 2-9
HIER_ANCESTOR 2-11
HIER_CHILD_COUNT 2-12
HIER_DEPTH 2-14
HIER_LEVEL 2-15
HIER_ORDER 2-16
HIER_PARENT 2-17
HIER_TOP 2-19
LAG 2-20
LAG_VARIANCE 2-22
LAG_VARIANCE_PERCENT 2-24
LEAD 2-26
LEAD_VARIANCE 2-28
LEAD_VARIANCE_PERCENT 2-31
MAX 2-33
MIN 2-35
OLAP_DML_EXPRESSION 2-37
RANK 2-38
ROW_NUMBER 2-40
SHARE 2-42
SUM 2-44

3 Row Functions

Row Functions in Alphabatical Order 3-1
Row Functions By Category 3-4

Numeric Functions 3-4
Character Functions That Return Characters 3-5
NLS Character Functions 3-6
Character Functions That Return Numbers 3-6
Datetime Functions 3-6
General Comparison Functions 3-6
Conversion Functions 3-7
Encoding and Decoding Function 3-7
Null-Related Functions 3-7
Environment and Identifier Functions 3-8
MONTHS_BETWEEN 3-26
NANVL 3-26
NEW_TIME 3-27
NEXT_DAY 3-28
NLS_CHARSET_ID 3-28
NLS_CHARSET_NAME 3-29
NLS_INITCAP 3-29
NLS_LOWER 3-30
NLS_UPPER 3-30
NLSSORT 3-31
NULLIF 3-31
NUMTODSINTERVAL 3-32
NUMTOYMINTERVAL 3-32
NVL 3-33
NVL2 3-33
ORA_HASH 3-34
POWER 3-35
RAWTOHEX 3-35
REGEXP_COUNT 3-35
REGEXP_REPLACE 3-36
REGEXP_INSTR 3-38
REGEXP_SUBSTR 3-39
REMAINDER 3-40
REPLACE 3-40
ROUND (date) 3-41
ROUND (number) 3-41
ROWIDTOCHAR 3-42
ROWIDTONCHAR 3-42
RPAD 3-42
RTRIM 3-43
SESSIONTIMEZONE 3-43
SIGN 3-43
SIN 3-44
SINH 3-44
SOUNDEX 3-45
SQRT 3-45
SUBSTR 3-46
SYS_CONTEXT 3-46
SYSDATE 3-48
SYSTIMESTAMP 3-49
TAN 3-49
TANH 3-49
TO_BINARY_DOUBLE 3-50
TO_BINARY_FLOAT 3-51
TO_CHAR (character) 3-52
TO_CHAR (datetime) 3-52
TO_CHAR (number) 3-53
TO_DATE 3-54
TO_DSINTERVAL 3-54
TO_NCHAR (character) 3-55
TO_NCHAR (datetime) 3-55
TO_NCHAR (number) 3-56
TO_NUMBER 3-57
TO_TIMESTAMP 3-57
TO_TIMESTAMP_TZ 3-58
TO_YMINTERVAL 3-59
TRANSLATE 3-60
TRANSLATE (USING) 3-60
TRIM 3-61
TRUNC (number) 3-61
TZ_OFFSET 3-62
UID 3-62
UNISTR 3-63
UPPER 3-63
USER 3-63
VSIZE 3-64
WIDTH_BUCKET 3-64

A

Reserved Words

Reserved Words A-1
Special Symbols A-4

Index
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Naming Conventions for Dimensional Objects</td>
<td>1-1</td>
</tr>
<tr>
<td>1-2</td>
<td>Dimensional Data Types</td>
<td>1-2</td>
</tr>
<tr>
<td>1-3</td>
<td>Unary Operators</td>
<td>1-4</td>
</tr>
<tr>
<td>1-4</td>
<td>Binary Operators</td>
<td>1-4</td>
</tr>
<tr>
<td>1-5</td>
<td>Multiplication Operator Example</td>
<td>1-5</td>
</tr>
<tr>
<td>1-6</td>
<td>Simple Comparison Operators</td>
<td>1-6</td>
</tr>
<tr>
<td>1-7</td>
<td>Group Comparison Operators</td>
<td>1-7</td>
</tr>
<tr>
<td>1-8</td>
<td>Conjunctions</td>
<td>1-9</td>
</tr>
<tr>
<td>1-9</td>
<td>Special Conditions Operators</td>
<td>1-9</td>
</tr>
<tr>
<td>1-10</td>
<td>LIKE Pattern-Matching Operators</td>
<td>1-10</td>
</tr>
<tr>
<td>3-1</td>
<td>Compatible Data Types</td>
<td>3-12</td>
</tr>
<tr>
<td>3-2</td>
<td>Time Zones</td>
<td>3-27</td>
</tr>
<tr>
<td>3-3</td>
<td>USERENV Attributes</td>
<td>3-47</td>
</tr>
<tr>
<td>A-1</td>
<td>OLAP Expression Syntax Symbols</td>
<td>A-4</td>
</tr>
</tbody>
</table>
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 that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

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

- Oracle OLAP User's Guide
  - 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 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:

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

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:

<table>
<thead>
<tr>
<th>Symbol or Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Brackets enclose optional items.</td>
</tr>
<tr>
<td>{ }</td>
<td>Braces enclose a choice of items, only one of which is required.</td>
</tr>
<tr>
<td></td>
<td>A vertical bar separates alternatives within brackets or braces.</td>
</tr>
<tr>
<td>...</td>
<td>Ellipses indicate that the preceding syntactic element can be repeated.</td>
</tr>
<tr>
<td>delimiters</td>
<td>Delimiters other than brackets, braces, and vertical bars must be entered as shown.</td>
</tr>
<tr>
<td><strong>boldface</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Convention</th>
<th>Quoted ID</th>
<th>Unquoted ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Character</td>
<td>Any character.</td>
<td>Alphabetic character from the database character set.</td>
</tr>
<tr>
<td>Other Characters</td>
<td>All characters, punctuation marks, and spaces are permitted. Double quotation marks and nulls (\0) are not permitted.</td>
<td>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.</td>
</tr>
<tr>
<td>Reserved Words</td>
<td>Permitted but not recommended.</td>
<td>Not permitted.</td>
</tr>
</tbody>
</table>

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 view.

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>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY_DOUBLE</td>
<td>A 64-bit floating number. A <code>BINARY_DOUBLE</code> value requires 9 bytes.</td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>A 32-bit floating number. A <code>BINARY_FLOAT</code> value requires 5 bytes.</td>
</tr>
<tr>
<td>CHAR (<code>size</code> [BYTE</td>
<td>CHAR])</td>
</tr>
<tr>
<td>DATE</td>
<td>A valid date in the range from January 1, 4712 BC to December 31, 9999 CE. It contains the datetime fields <code>YEAR</code>, <code>MONTH</code>, <code>DAY</code>, <code>HOUR</code>, <code>MINUTE</code>, and <code>SECOND</code>. It does not have fractional seconds or a time zone. The default format is determined explicitly by the <code>NLS_DATE_FORMAT</code> parameter and implicitly by the <code>NLS_TERRITORY</code> parameter. A <code>DATE</code> value requires 7 bytes.</td>
</tr>
<tr>
<td>DECIMAL (<code>p</code>,<code>s</code>)</td>
<td>A decimal number with precision <code>p</code> and scale <code>s</code> represented as a <code>NUMBER</code> data type.</td>
</tr>
<tr>
<td>FLOAT [(<code>p</code>)]</td>
<td>A subtype of <code>NUMBER</code> with precision <code>p</code>. A <code>FLOAT</code> is represented internally as <code>NUMBER</code>. The precision can range from 1 to 126 binary digits. A <code>FLOAT</code> value requires from 1 to 22 bytes.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>A whole number represented as a <code>NUMBER</code> data type with a scale of 0.</td>
</tr>
<tr>
<td>INTERVAL DAY[(<code>day_precision</code>)] TO SECOND[(<code>second_precision</code>)]</td>
<td>A period of time in days, hours, minutes, and seconds. The day precision is the maximum number of digits in the <code>DAY</code> datetime field. The default is 2. The second precision is the number of digits in the fractional part of the <code>SECOND</code> field. The default value is 6. Both day and second precision can have a value from 0 to 9. An <code>INTERVAL DAY TO SECOND</code> value requires 11 bytes.</td>
</tr>
<tr>
<td>INTERVAL YEAR[(<code>precision</code>)] TO MONTH</td>
<td>A period of time in years and months. The precision is the number of digits in the <code>YEAR</code> datetime field, which can have a value of 0 to 9. The default precision is 2 digits. An <code>INTERVAL YEAR TO MONTH</code> value requires 5 bytes.</td>
</tr>
<tr>
<td>NCHAR[(<code>size</code>)]</td>
<td>A fixed length character string with a length of <code>size</code> 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.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUMBER ((p[,s]))</td>
<td>A decimal number with precision (p) and scale (s). 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.</td>
</tr>
<tr>
<td>NVARCHAR2(size)</td>
<td>A variable length Unicode character string with a maximum length of size 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.</td>
</tr>
<tr>
<td>TIMESTAMP([\text{precision}])</td>
<td>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.</td>
</tr>
</tbody>
</table>
| TIMESTAMP \([\text{precision}]\) WITH LOCAL TIME ZONE | A valid date with the same description as TIMESTAMP WITH TIME ZONE with these exceptions:  
• The data is stored in the database with the database time zone.  
• The data is converted to the session time zone when it is retrieved.  
• A TIMESTAMP WITH LOCAL TIME ZONE value requires from 7 to 11 bytes depending on the precision. |
| TIMESTAMP\([\text{precision}]\) 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 TIME ZONE value requires 13 bytes. |
| VARCHAR2\((size \{BYTE|CHAR\})\) | A variable length character string with a maximum length of size 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”.

Unary Arithmetic Operators

A unary operator operates on only one operand.

Table 1-3   Unary Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Positive value</td>
</tr>
<tr>
<td>-</td>
<td>Negative value</td>
</tr>
</tbody>
</table>

Syntax

operator operand

Example

-5 is a negative number.

Binary Arithmetic Operators

A binary operator operates on two operands.

Table 1-4   Binary Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td>-</td>
<td>Subtract</td>
</tr>
<tr>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Level</th>
<th>Sales</th>
<th>Sales Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>CLASS</td>
<td>124191336</td>
<td>131642816</td>
</tr>
<tr>
<td>Desktop PCs</td>
<td>FAMILY</td>
<td>74556528</td>
<td>79029919</td>
</tr>
<tr>
<td>Monitors</td>
<td>FAMILY</td>
<td>3972142</td>
<td>4210470</td>
</tr>
<tr>
<td>Memory</td>
<td>FAMILY</td>
<td>5619219</td>
<td>5956372</td>
</tr>
<tr>
<td>Modems/Fax</td>
<td>FAMILY</td>
<td>5575726</td>
<td>5910269</td>
</tr>
<tr>
<td>CD/DVD</td>
<td>FAMILY</td>
<td>16129497</td>
<td>17097267</td>
</tr>
<tr>
<td>Portable PCs</td>
<td>FAMILY</td>
<td>18338225</td>
<td>19438518</td>
</tr>
</tbody>
</table>

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

\[
\text{UNITS\_CUBE.SALES} - \text{UNITS\_CUBE.COST}
\]

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

Concatenation Operator

The concatenation operator (||) combines text expressions.

Syntax

\[
\text{operand} \ || \ \text{operand}
\]

Example

'\text{The date today is: }' || \text{sysdate} \text{ generates a text string such as } \text{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.

\[
\text{PRODUCT.LEVEL\_NAME} \ || \ ' ' \ || \text{PRODUCT.DIM\_KEY}
\]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dim Key</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS</td>
<td>HRD</td>
<td>CLASS HRD</td>
</tr>
<tr>
<td>FAMILY</td>
<td>DTPC</td>
<td>FAMILY DTPC</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>!= <code>^</code> &lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

Syntax

expr | > | >= | < | <= | = | != `^` <> | expr
Arguments

\( expr \) can be any expression.

Examples

\( 5 > 3 \) is true, \( 4 \neq 5 \) is true, \( 6 \geq 9 \) 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>
<thead>
<tr>
<th>Table 1-7</th>
<th>Group Comparison Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>( &gt; )</td>
<td>Greater than</td>
</tr>
<tr>
<td>( &gt;= )</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>( &lt; )</td>
<td>Less than</td>
</tr>
<tr>
<td>( &lt;= )</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>( = )</td>
<td>Equal to</td>
</tr>
<tr>
<td>( \neq )</td>
<td>Not equal to</td>
</tr>
<tr>
<td>ANY</td>
<td>Returns true if one or more values in the list match, or false if no values match.</td>
</tr>
<tr>
<td>SOME</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>Returns true if all values in the list match, or false if one or more values do not match.</td>
</tr>
</tbody>
</table>

Syntax

\[ expr \{ = | \neq | \sim\neq | < > | > | < | >\geq | \leq \} \{ ANY | SOME | ALL \} \{ expression_list | subquery \} \]

Examples

\( 5 \leq \text{ALL} (5, 10, 15) \) is true, \( 5 \neq \text{ANY (5, 10, 15)} \) 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

\[ expr1 \text{ IS [ NOT ] [ PARENT} \text{ ]} \]
CHILD
ANCESTOR
ROOT_ANCESTOR
DESCENDANT
LEAF_DESCENDANT
RELATIVE

} OR SELF ] OF expr2 WITHIN hierarchy

Arguments

dim1 is any expression, including a literal or a column, that resolves to a dimension member.

PARENT compares dim1 as the parent of expr2.

CHILD compares dim1 as a child of expr2.

ANCESTOR compares dim1 as an ancestor of expr2.

ROOT_ANCESTOR compares dim1 as the highest-level ancestor of expr2.

DESCENDANT compares dim1 as a descendant of expr2.

LEAF_DESCENDANT compares dim1 as a descendant that has no children of expr2.

RELATIVE compares dim1 as a dimension member that has a parent in common with expr2.

OR SELF compares dim1 as the same dimension member as expr2.

dim2 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
dim1 and dim2, 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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Returns true if both component conditions are true. Returns false if either is false.</td>
</tr>
<tr>
<td>OR</td>
<td>Returns true if either component condition is true. Returns false if both are false.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS INFINITE</td>
<td>Returns true if the expression is infinite, or false otherwise. For mappings only.</td>
</tr>
<tr>
<td>IS NAN</td>
<td>Returns true if the expression is not a number, or false otherwise. For mappings only.</td>
</tr>
<tr>
<td>IS NULL</td>
<td>Returns true if the expression is null, or false otherwise.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIKE</td>
<td>Uses characters in the input character set.</td>
</tr>
<tr>
<td>LIKEC</td>
<td>Uses Unicode complete characters. It treats a Unicode supplementary character as two characters.</td>
</tr>
<tr>
<td>LIKE2</td>
<td>Uses UCS2 code points. It treats a Unicode supplementary character as one character.</td>
</tr>
<tr>
<td>LIKE4</td>
<td>Uses UCS4 code points. It treats a composite character as one character.</td>
</tr>
</tbody>
</table>

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

```plaintext
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:

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Package</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.44MB External 3.5&quot; Diskette</td>
<td>Executive</td>
<td>Value Pack</td>
</tr>
<tr>
<td>1GB USB Drive</td>
<td>--</td>
<td>Single Item</td>
</tr>
<tr>
<td>512MB USB Drive</td>
<td>--</td>
<td>Single Item</td>
</tr>
<tr>
<td>56Kbps V.90 Type II Modem</td>
<td>Executive</td>
<td>Value Pack</td>
</tr>
<tr>
<td>56Kbps V.92 Type II Fax/Modem</td>
<td>Laptop Value Pack</td>
<td>Value Pack</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>Executive</td>
<td>Value Pack</td>
</tr>
<tr>
<td>Envoy Ambassador</td>
<td>--</td>
<td>Single Item</td>
</tr>
<tr>
<td>Envoy Executive</td>
<td>Executive</td>
<td>Value Pack</td>
</tr>
<tr>
<td>Envoy External Keyboard</td>
<td>Executive</td>
<td>Value Pack</td>
</tr>
<tr>
<td>Envoy Standard Laptop</td>
<td>Value Pack</td>
<td>Value Pack</td>
</tr>
<tr>
<td>External - DVD-RW - 8X</td>
<td>Executive</td>
<td>Value Pack</td>
</tr>
<tr>
<td>External 101-key keyboard</td>
<td>Multimedia</td>
<td>Value Pack</td>
</tr>
<tr>
<td>External 48X CD-ROM</td>
<td>--</td>
<td>Single Item</td>
</tr>
</tbody>
</table>
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
```

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
```
<table>
<thead>
<tr>
<th>Product</th>
<th>Time</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Q1.05</td>
<td>28172590</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q2.05</td>
<td>34520379</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q3.05</td>
<td>29466573</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q4.05</td>
<td>32031795</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q1.06</td>
<td>32711891</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q2.06</td>
<td>33637473</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q3.06</td>
<td>29227635</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q4.06</td>
<td>31319881</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q1.07</td>
<td>--</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q2.07</td>
<td>--</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q3.07</td>
<td>--</td>
</tr>
<tr>
<td>Hardware</td>
<td>Q4.07</td>
<td>--</td>
</tr>
</tbody>
</table>

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


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.

time
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)

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

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

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>Calculates all time periods at the same level. (Default)</td>
</tr>
<tr>
<td>PARENT</td>
<td>Calculates time periods at the same level with the same parent.</td>
</tr>
<tr>
<td>GREGORIAN YEAR</td>
<td>Calculates time periods within the same Gregorian year.</td>
</tr>
<tr>
<td>GREGORIAN QUARTER</td>
<td>Calculates time periods within the same Gregorian quarter.</td>
</tr>
<tr>
<td>GREGORIAN MONTH</td>
<td>Calculates time periods within the same Gregorian month.</td>
</tr>
<tr>
<td>GREGORIAN WEEK</td>
<td>Calculates time periods within the same Gregorian week.</td>
</tr>
<tr>
<td>ANCESTOR</td>
<td>Includes time periods at the same level and with the same ancestor at a specified level.</td>
</tr>
</tbody>
</table>
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)
```

<table>
<thead>
<tr>
<th>TIME</th>
<th>PARENT</th>
<th>UNITS</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1.05</td>
<td>CY2005</td>
<td>143607</td>
<td>143607</td>
</tr>
<tr>
<td>Q2.05</td>
<td>CY2005</td>
<td>138096</td>
<td>140852</td>
</tr>
<tr>
<td>Q3.05</td>
<td>CY2005</td>
<td>138953</td>
<td>140219</td>
</tr>
<tr>
<td>Q4.05</td>
<td>CY2005</td>
<td>145062</td>
<td>141430</td>
</tr>
<tr>
<td>Q1.06</td>
<td>CY2006</td>
<td>146819</td>
<td>146819</td>
</tr>
<tr>
<td>Q2.06</td>
<td>CY2006</td>
<td>145233</td>
<td>146026</td>
</tr>
<tr>
<td>Q3.06</td>
<td>CY2006</td>
<td>143572</td>
<td>145208</td>
</tr>
<tr>
<td>Q4.06</td>
<td>CY2006</td>
<td>149305</td>
<td>146232</td>
</tr>
</tbody>
</table>

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::=
```

Chapter 2

COUNT
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.

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>Calculates all time periods at the same level. (Default)</td>
</tr>
<tr>
<td>PARENT</td>
<td>Calculates time periods at the same level with the same parent.</td>
</tr>
<tr>
<td>GREGORIAN YEAR</td>
<td>Calculates time periods within the same Gregorian year.</td>
</tr>
<tr>
<td>GREGORIAN QUARTER</td>
<td>Calculates time periods within the same Gregorian quarter.</td>
</tr>
<tr>
<td>GREGORIAN MONTH</td>
<td>Calculates time periods within the same Gregorian month.</td>
</tr>
<tr>
<td>Range</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GREGORIAN WEEK</td>
<td>Calculates time periods within the same Gregorian week.</td>
</tr>
<tr>
<td>ANCESTOR</td>
<td>Includes time periods at the same level and with the same ancestor at a specified level.</td>
</tr>
</tbody>
</table>

**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)
```

<table>
<thead>
<tr>
<th>TIME</th>
<th>UNITS</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN-06</td>
<td>47776</td>
<td>1</td>
</tr>
<tr>
<td>FEB-06</td>
<td>47695</td>
<td>2</td>
</tr>
<tr>
<td>MAR-06</td>
<td>51348</td>
<td>3</td>
</tr>
<tr>
<td>APR-06</td>
<td>47005</td>
<td>4</td>
</tr>
<tr>
<td>MAY-06</td>
<td>52809</td>
<td>5</td>
</tr>
<tr>
<td>JUN-06</td>
<td>45419</td>
<td>6</td>
</tr>
<tr>
<td>JUL-06</td>
<td>48388</td>
<td>7</td>
</tr>
<tr>
<td>AUG-06</td>
<td>48830</td>
<td>8</td>
</tr>
<tr>
<td>SEP-06</td>
<td>46354</td>
<td>9</td>
</tr>
<tr>
<td>OCT-06</td>
<td>47411</td>
<td>10</td>
</tr>
<tr>
<td>NOV-06</td>
<td>46842</td>
<td>11</td>
</tr>
<tr>
<td>DEC-06</td>
<td>55052</td>
<td>12</td>
</tr>
</tbody>
</table>

**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_byClause::=**

```
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)

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

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)
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Ancestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>--</td>
</tr>
<tr>
<td>Q1.06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>Q2.06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>Q3.06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>Q4.06</td>
<td>CY2006.Q4</td>
</tr>
<tr>
<td>JAN-06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>FEB-06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>MAR-06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>APR-06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>MAY-06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>JUN-06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>JUL-06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>AUG-06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>SEP-06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>OCT-06</td>
<td>CY2006.Q4</td>
</tr>
<tr>
<td>NOV-06</td>
<td>CY2006.Q4</td>
</tr>
<tr>
<td>DEC-06</td>
<td>CY2006.Q4</td>
</tr>
</tbody>
</table>

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")

<table>
<thead>
<tr>
<th>Time</th>
<th>Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>4</td>
</tr>
<tr>
<td>Q1.06</td>
<td>3</td>
</tr>
<tr>
<td>Q2.06</td>
<td>3</td>
</tr>
<tr>
<td>Q3.06</td>
<td>3</td>
</tr>
<tr>
<td>Q4.06</td>
<td>3</td>
</tr>
<tr>
<td>JAN-06</td>
<td>0</td>
</tr>
<tr>
<td>FEB-06</td>
<td>0</td>
</tr>
<tr>
<td>MAR-06</td>
<td>0</td>
</tr>
<tr>
<td>APR-06</td>
<td>0</td>
</tr>
<tr>
<td>MAY-06</td>
<td>0</td>
</tr>
<tr>
<td>JUN-06</td>
<td>0</td>
</tr>
<tr>
<td>JUL-06</td>
<td>0</td>
</tr>
<tr>
<td>AUG-06</td>
<td>0</td>
</tr>
<tr>
<td>SEP-06</td>
<td>0</td>
</tr>
<tr>
<td>OCT-06</td>
<td>0</td>
</tr>
<tr>
<td>NOV-06</td>
<td>0</td>
</tr>
<tr>
<td>DEC-06</td>
<td>0</td>
</tr>
</tbody>
</table>
The next example returns 8 as the number of children for Government within the Market hierarchy of the Customer dimension.

\[ \text{HIER\_CHILD\_COUNT('GOV' WITHIN HIERARCHY CUSTOMER.MARKET)} \]

**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**

\[ \text{HIER\_DEPTH (} \text{[member\_expression] \text{[WITHIN]}} \]
\[ \text{\{DIMENSION dimension\_id | HIERARCHY hierarchy\_id\})} \]

**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.

\[ \text{HIER\_DEPTH(DIMENSION "TIME")} \]

<table>
<thead>
<tr>
<th>Time</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>1</td>
</tr>
<tr>
<td>Q1.06</td>
<td>2</td>
</tr>
<tr>
<td>Q2.06</td>
<td>2</td>
</tr>
<tr>
<td>Q3.06</td>
<td>2</td>
</tr>
<tr>
<td>Q4.06</td>
<td>2</td>
</tr>
<tr>
<td>JAN-06</td>
<td>3</td>
</tr>
<tr>
<td>FEB-06</td>
<td>3</td>
</tr>
<tr>
<td>MAR-06</td>
<td>3</td>
</tr>
<tr>
<td>Time</td>
<td>Depth</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>APR-06</td>
<td>3</td>
</tr>
<tr>
<td>MAY-06</td>
<td>3</td>
</tr>
<tr>
<td>JUN-06</td>
<td>3</td>
</tr>
<tr>
<td>JUL-06</td>
<td>3</td>
</tr>
<tr>
<td>AUG-06</td>
<td>3</td>
</tr>
<tr>
<td>SEP-06</td>
<td>3</td>
</tr>
<tr>
<td>OCT-06</td>
<td>3</td>
</tr>
<tr>
<td>NOV-06</td>
<td>3</td>
</tr>
<tr>
<td>DEC-06</td>
<td>3</td>
</tr>
</tbody>
</table>

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")
<table>
<thead>
<tr>
<th>Time</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>CALENDAR_YEAR</td>
</tr>
<tr>
<td>Q1.06</td>
<td>CALENDAR_QUARTER</td>
</tr>
<tr>
<td>Q2.06</td>
<td>CALENDAR_QUARTER</td>
</tr>
<tr>
<td>Q3.06</td>
<td>CALENDAR_QUARTER</td>
</tr>
<tr>
<td>Q4.06</td>
<td>CALENDAR_QUARTER</td>
</tr>
<tr>
<td>JAN-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>FEB-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>MAR-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>APR-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>MAY-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>JUN-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>JUL-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>AUG-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>SEP-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>OCT-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>NOV-06</td>
<td>MONTH</td>
</tr>
<tr>
<td>DEC-06</td>
<td>MONTH</td>
</tr>
</tbody>
</table>

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")
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>138</td>
</tr>
<tr>
<td>Q1.06</td>
<td>139</td>
</tr>
<tr>
<td>JAN-06</td>
<td>140</td>
</tr>
<tr>
<td>FEB-06</td>
<td>141</td>
</tr>
<tr>
<td>MAR-06</td>
<td>142</td>
</tr>
<tr>
<td>Q2.06</td>
<td>143</td>
</tr>
<tr>
<td>APR-06</td>
<td>144</td>
</tr>
<tr>
<td>MAY-06</td>
<td>145</td>
</tr>
<tr>
<td>JUN-06</td>
<td>146</td>
</tr>
<tr>
<td>Q3.06</td>
<td>147</td>
</tr>
<tr>
<td>JUL-06</td>
<td>148</td>
</tr>
<tr>
<td>AUG-06</td>
<td>149</td>
</tr>
<tr>
<td>SEP-06</td>
<td>150</td>
</tr>
<tr>
<td>Q4.06</td>
<td>151</td>
</tr>
<tr>
<td>OCT-06</td>
<td>152</td>
</tr>
<tr>
<td>NOV-06</td>
<td>153</td>
</tr>
<tr>
<td>DEC-06</td>
<td>154</td>
</tr>
</tbody>
</table>

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

\[ \text{HIER\_PARENT} ( \\text{[member\_expression]} \ [\text{WITHIN}] \ \{ \text{DIMENSION} \ \text{dimension\_id} \ | \ \text{HIERARCHY} \ \text{hierarchy\_id} \} ) \]

Arguments

\textit{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.

\textit{dimension\_id}

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

\textit{hierarchy\_id}

The hierarchy over which the values are calculated. If \textit{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.

\text{HIER\_PARENT(DIMENSION GLOBAL\_TIME)}

<table>
<thead>
<tr>
<th>Time</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>TOTAL</td>
</tr>
<tr>
<td>Q1.06</td>
<td>CY2006</td>
</tr>
<tr>
<td>Q2.06</td>
<td>CY2006</td>
</tr>
<tr>
<td>Q3.06</td>
<td>CY2006</td>
</tr>
<tr>
<td>Q4.06</td>
<td>CY2006</td>
</tr>
<tr>
<td>JAN-06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>FEB-06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>MAR-06</td>
<td>CY2006.Q1</td>
</tr>
<tr>
<td>APR-06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>MAY-06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>JUN-06</td>
<td>CY2006.Q2</td>
</tr>
<tr>
<td>JUL-06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>AUG-06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>SEP-06</td>
<td>CY2006.Q3</td>
</tr>
<tr>
<td>OCT-06</td>
<td>CY2006.Q4</td>
</tr>
<tr>
<td>NOV-06</td>
<td>CY2006.Q4</td>
</tr>
<tr>
<td>DEC-06</td>
<td>CY2006.Q4</td>
</tr>
</tbody>
</table>
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")
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>TOTAL</td>
</tr>
<tr>
<td>Q1.06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>Q2.06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>Q3.06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>Q4.06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>JAN-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>FEB-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>MAR-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>APR-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>Time</td>
<td>Top</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>MAY-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>JUN-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>JUL-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>AUG-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>SEP-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>OCT-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>NOV-06</td>
<td>TOTAL</td>
</tr>
<tr>
<td>DEC-06</td>
<td>TOTAL</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Offset Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>The member at the same level offset periods before the current member. (Default)</td>
</tr>
<tr>
<td>PARENT</td>
<td>The member at the same level with the same parent offset periods before the current member.</td>
</tr>
</tbody>
</table>
### Offset Unit

<table>
<thead>
<tr>
<th>Offset Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREGORIAN YEAR</td>
<td>The period at the same level with a start date exactly offset years before the start date of the current period.</td>
</tr>
<tr>
<td>GREGORIAN QUARTER</td>
<td>The period at the same level with a start date exactly offset quarters before the start date of the current period.</td>
</tr>
<tr>
<td>GREGORIAN MONTH</td>
<td>The period at the same level with a start date exactly offset months before the start date of the current period.</td>
</tr>
<tr>
<td>GREGORIAN WEEK</td>
<td>The period at the same level with a start date exactly offset weeks before the start date of the current period.</td>
</tr>
<tr>
<td>GREGORIAN DAY</td>
<td>The period at the same level with a start date exactly offset days before the start date of the current period.</td>
</tr>
<tr>
<td>ANCESTOR</td>
<td>The period at the same level as the current period and whose ancestor is offset positions before the ancestor of the current period.</td>
</tr>
</tbody>
</table>

**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:

**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)

<table>
<thead>
<tr>
<th>Time</th>
<th>Units</th>
<th>Last Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1.05</td>
<td>143607</td>
<td>146529</td>
</tr>
<tr>
<td>Q2.05</td>
<td>138096</td>
<td>143070</td>
</tr>
<tr>
<td>Q3.05</td>
<td>138953</td>
<td>148292</td>
</tr>
<tr>
<td>Q4.05</td>
<td>145062</td>
<td>149528</td>
</tr>
<tr>
<td>Q1.06</td>
<td>146819</td>
<td>143607</td>
</tr>
<tr>
<td>Q2.06</td>
<td>145233</td>
<td>138096</td>
</tr>
<tr>
<td>Q3.06</td>
<td>143572</td>
<td>138953</td>
</tr>
<tr>
<td>Q4.06</td>
<td>149305</td>
<td>145062</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Offset Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>The member at the same level offset periods before the current member. (Default)</td>
</tr>
</tbody>
</table>
### Offset Units

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

### 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:
Examples

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

\[
\text{LAG\_VARIANCE (GLOBAL\_UNITS\_CUBE\_UNITS, 1) OVER (HIERARCHY GLOBAL\_TIME\_CALENDAR ANCESTOR AT DIMENSION LEVEL GLOBAL\_TIME\_CALENDAR\_YEAR)}
\]

<table>
<thead>
<tr>
<th>Time</th>
<th>Units</th>
<th>Last Year</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1.05</td>
<td>143607</td>
<td>146529</td>
<td>-2922</td>
</tr>
<tr>
<td>Q2.05</td>
<td>138096</td>
<td>143070</td>
<td>-4974</td>
</tr>
<tr>
<td>Q3.05</td>
<td>138953</td>
<td>148292</td>
<td>-9339</td>
</tr>
<tr>
<td>Q4.05</td>
<td>145062</td>
<td>149528</td>
<td>-4466</td>
</tr>
<tr>
<td>Q1.06</td>
<td>146819</td>
<td>143607</td>
<td>3212</td>
</tr>
<tr>
<td>Q2.06</td>
<td>145233</td>
<td>138096</td>
<td>7137</td>
</tr>
<tr>
<td>Q3.06</td>
<td>143572</td>
<td>138953</td>
<td>4619</td>
</tr>
<tr>
<td>Q4.06</td>
<td>149305</td>
<td>145062</td>
<td>4243</td>
</tr>
</tbody>
</table>

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

\[
\text{LAG\_VARIANCE\_PERCENT (lag\_args) OVER (lag\_clause)}
\]

\[
\text{lag\_args}::=\]

expression, offset [, {default\_expression | CLOSEST} ]

\[
\text{lag\_clause}::=\]

[ (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

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

**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:

Examples

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

\[
\text{LAG\_VARIANCE\_PERCENT (GLOBAL.UNIT\_CUBE.UNIT\_S, 1) OVER (HIERARCHY GLOBAL.\text{TIME}.CALENDAR ANCESTOR AT DIMENSION LEVEL GLOBAL.\text{TIME}.CALENDAR\_YEAR)}
\]

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

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)

\textit{lead\_args}::= 

expression, offset \[, \{default\_expression | CLOSEST \} \]

\textit{lead\_clause}::= 

\[ (\text{DIMENSION dimension\_id | HIERARCHY hierarchy\_id}) \]

\[ \{ \text{BY} ( LEVEL \text PARENT \text{GREGORIAN} \{ \text{YEAR | QUARTER | MONTH | WEEK | DAY} \} \text{ANCESTOR AT} ( \text{DIMENSION LEVEL dim\_level\_id | HIERARCHY LEVEL hier\_level\_id} \} \} \]
Arguments

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

**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:

**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)
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Parent</th>
<th>Units</th>
<th>Next Qtr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>TOTAL</td>
<td>565718</td>
<td>--</td>
</tr>
<tr>
<td>Q1-05</td>
<td>CY2005</td>
<td>143607</td>
<td>138096</td>
</tr>
<tr>
<td>Q2-05</td>
<td>CY2005</td>
<td>138096</td>
<td>138953</td>
</tr>
<tr>
<td>Q3-05</td>
<td>CY2005</td>
<td>138953</td>
<td>145062</td>
</tr>
<tr>
<td>Q4-05</td>
<td>CY2005</td>
<td>145062</td>
<td>146819</td>
</tr>
<tr>
<td>Jan-05</td>
<td>CY2005.Q1</td>
<td>50098</td>
<td>40223</td>
</tr>
<tr>
<td>Feb-05</td>
<td>CY2005.Q1</td>
<td>43990</td>
<td>45477</td>
</tr>
<tr>
<td>Mar-05</td>
<td>CY2005.Q1</td>
<td>49519</td>
<td>52396</td>
</tr>
<tr>
<td>Apr-05</td>
<td>CY2005.Q2</td>
<td>40223</td>
<td>45595</td>
</tr>
<tr>
<td>May-05</td>
<td>CY2005.Q2</td>
<td>45477</td>
<td>46882</td>
</tr>
<tr>
<td>Jun-05</td>
<td>CY2005.Q2</td>
<td>52396</td>
<td>46476</td>
</tr>
<tr>
<td>Jul-05</td>
<td>CY2005.Q3</td>
<td>45595</td>
<td>47476</td>
</tr>
<tr>
<td>Aug-05</td>
<td>CY2005.Q3</td>
<td>46882</td>
<td>47496</td>
</tr>
<tr>
<td>Sep-05</td>
<td>CY2005.Q3</td>
<td>46476</td>
<td>50090</td>
</tr>
<tr>
<td>Oct-05</td>
<td>CY2005.Q4</td>
<td>47476</td>
<td>47776</td>
</tr>
<tr>
<td>Nov-05</td>
<td>CY2005.Q4</td>
<td>47496</td>
<td>47695</td>
</tr>
<tr>
<td>Dec-05</td>
<td>CY2005.Q4</td>
<td>50090</td>
<td>51348</td>
</tr>
</tbody>
</table>

**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

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

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:

**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.

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

<table>
<thead>
<tr>
<th>Time</th>
<th>Parent</th>
<th>Units</th>
<th>Next Qtr</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>TOTAL</td>
<td>565718</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Q1-05</td>
<td>CY2005</td>
<td>143607</td>
<td>138096</td>
<td>5511</td>
</tr>
<tr>
<td>Q2-05</td>
<td>CY2005</td>
<td>138096</td>
<td>138953</td>
<td>-857</td>
</tr>
<tr>
<td>Q3-05</td>
<td>CY2005</td>
<td>138953</td>
<td>145062</td>
<td>-6109</td>
</tr>
<tr>
<td>Q4-05</td>
<td>CY2005</td>
<td>145062</td>
<td>146819</td>
<td>-1757</td>
</tr>
<tr>
<td>Jan-05</td>
<td>CY2005.Q1</td>
<td>50098</td>
<td>40223</td>
<td>9875</td>
</tr>
<tr>
<td>Feb-05</td>
<td>CY2005.Q1</td>
<td>43990</td>
<td>45477</td>
<td>-1487</td>
</tr>
<tr>
<td>Mar-05</td>
<td>CY2005.Q1</td>
<td>49519</td>
<td>52396</td>
<td>-2877</td>
</tr>
<tr>
<td>Apr-05</td>
<td>CY2005.Q2</td>
<td>40223</td>
<td>45595</td>
<td>-5372</td>
</tr>
<tr>
<td>May-05</td>
<td>CY2005.Q2</td>
<td>45477</td>
<td>46882</td>
<td>-1405</td>
</tr>
<tr>
<td>Jun-05</td>
<td>CY2005.Q2</td>
<td>52396</td>
<td>46476</td>
<td>5920</td>
</tr>
<tr>
<td>Jul-05</td>
<td>CY2005.Q3</td>
<td>45595</td>
<td>47476</td>
<td>-1881</td>
</tr>
<tr>
<td>Aug-05</td>
<td>CY2005.Q3</td>
<td>46882</td>
<td>47496</td>
<td>-614</td>
</tr>
<tr>
<td>Sep-05</td>
<td>CY2005.Q3</td>
<td>46476</td>
<td>50090</td>
<td>-3614</td>
</tr>
<tr>
<td>Nov-05</td>
<td>CY2005.Q4</td>
<td>47496</td>
<td>47695</td>
<td>-199</td>
</tr>
<tr>
<td>Dec-05</td>
<td>CY2005.Q4</td>
<td>50090</td>
<td>51348</td>
<td>-1258</td>
</tr>
</tbody>
</table>

**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

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

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:

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)

<table>
<thead>
<tr>
<th>Time</th>
<th>Parent</th>
<th>Units</th>
<th>Next Qtr</th>
<th>Difference</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>TOTAL</td>
<td>565718</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Q1-05</td>
<td>CY2005</td>
<td>143607</td>
<td>138096</td>
<td>5511</td>
<td>.04</td>
</tr>
<tr>
<td>Q2-05</td>
<td>CY2005</td>
<td>138096</td>
<td>138953</td>
<td>-857</td>
<td>-.01</td>
</tr>
<tr>
<td>Q3-05</td>
<td>CY2005</td>
<td>138953</td>
<td>145062</td>
<td>-6109</td>
<td>-.04</td>
</tr>
<tr>
<td>Q4-05</td>
<td>CY2005</td>
<td>145062</td>
<td>146819</td>
<td>-1757</td>
<td>-.01</td>
</tr>
<tr>
<td>Jan-05</td>
<td>CY2005.Q1</td>
<td>50098</td>
<td>40223</td>
<td>9875</td>
<td>.25</td>
</tr>
<tr>
<td>Feb-05</td>
<td>CY2005.Q1</td>
<td>43990</td>
<td>45477</td>
<td>-1487</td>
<td>-.03</td>
</tr>
<tr>
<td>Mar-05</td>
<td>CY2005.Q1</td>
<td>49519</td>
<td>52396</td>
<td>-2877</td>
<td>-.05</td>
</tr>
<tr>
<td>Apr-05</td>
<td>CY2005.Q2</td>
<td>40223</td>
<td>45595</td>
<td>-5372</td>
<td>-.12</td>
</tr>
<tr>
<td>May-05</td>
<td>CY2005.Q2</td>
<td>45477</td>
<td>46882</td>
<td>-1405</td>
<td>-.03</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>Calculates all time periods at the same level. (Default)</td>
</tr>
<tr>
<td>PARENT</td>
<td>Calculates time periods at the same level with the same parent.</td>
</tr>
<tr>
<td>GREGORIAN YEAR</td>
<td>Calculates time periods within the same Gregorian year.</td>
</tr>
<tr>
<td>GREGORIAN QUARTER</td>
<td>Calculates time periods within the same Gregorian quarter.</td>
</tr>
<tr>
<td>GREGORIAN MONTH</td>
<td>Calculates time periods within the same Gregorian month.</td>
</tr>
<tr>
<td>GREGORIAN WEEK</td>
<td>Calculates time periods within the same Gregorian week.</td>
</tr>
<tr>
<td>ANCESTOR</td>
<td>Includes time periods at the same level and with the same ancestor at a specified level.</td>
</tr>
</tbody>
</table>

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.

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.TIMECALENDAR_YEAR)

<table>
<thead>
<tr>
<th>Time</th>
<th>Units</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN-06</td>
<td>47776</td>
<td>47776</td>
</tr>
<tr>
<td>FEB-06</td>
<td>47695</td>
<td>47776</td>
</tr>
<tr>
<td>MAR-06</td>
<td>51348</td>
<td>51348</td>
</tr>
<tr>
<td>APR-06</td>
<td>47005</td>
<td>51348</td>
</tr>
<tr>
<td>MAY-06</td>
<td>52809</td>
<td>52809</td>
</tr>
<tr>
<td>JUN-06</td>
<td>45419</td>
<td>52809</td>
</tr>
<tr>
<td>JUL-06</td>
<td>48388</td>
<td>52809</td>
</tr>
<tr>
<td>AUG-06</td>
<td>48830</td>
<td>52809</td>
</tr>
<tr>
<td>SEP-06</td>
<td>46354</td>
<td>52809</td>
</tr>
<tr>
<td>OCT-06</td>
<td>47411</td>
<td>52809</td>
</tr>
<tr>
<td>NOV-06</td>
<td>46842</td>
<td>52809</td>
</tr>
<tr>
<td>DEC-06</td>
<td>55052</td>
<td>55052</td>
</tr>
</tbody>
</table>

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 | followingboundary
| 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

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>Calculates all time periods at the same level. (Default)</td>
</tr>
<tr>
<td>PARENT</td>
<td>Calculates time periods at the same level with the same parent.</td>
</tr>
<tr>
<td>GREGORIAN YEAR</td>
<td>Calculates time periods within the same Gregorian year.</td>
</tr>
<tr>
<td>GREGORIAN QUARTER</td>
<td>Calculates time periods within the same Gregorian quarter.</td>
</tr>
<tr>
<td>GREGORIAN MONTH</td>
<td>Calculates time periods within the same Gregorian month.</td>
</tr>
<tr>
<td>GREGORIAN WEEK</td>
<td>Calculates time periods within the same Gregorian week.</td>
</tr>
<tr>
<td>ANCESTOR</td>
<td>Includes time periods at the same level and with the same ancestor at a specified level.</td>
</tr>
</tbody>
</table>

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.

**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)
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Units</th>
<th>Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN-06</td>
<td>47776</td>
<td>45419</td>
</tr>
<tr>
<td>FEB-06</td>
<td>47695</td>
<td>45419</td>
</tr>
<tr>
<td>MAR-06</td>
<td>51348</td>
<td>45419</td>
</tr>
<tr>
<td>APR-06</td>
<td>47005</td>
<td>45419</td>
</tr>
<tr>
<td>MAY-06</td>
<td>52809</td>
<td>45419</td>
</tr>
<tr>
<td>JUN-06</td>
<td>45419</td>
<td>45419</td>
</tr>
<tr>
<td>JUL-06</td>
<td>48388</td>
<td>46354</td>
</tr>
<tr>
<td>AUG-06</td>
<td>48830</td>
<td>46354</td>
</tr>
<tr>
<td>SEP-06</td>
<td>46354</td>
<td>46354</td>
</tr>
<tr>
<td>OCT-06</td>
<td>47411</td>
<td>46842</td>
</tr>
<tr>
<td>NOV-06</td>
<td>46842</td>
<td>46842</td>
</tr>
<tr>
<td>DEC-06</td>
<td>55052</td>
<td>55052</td>
</tr>
</tbody>
</table>

**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)
```

---

Chapter 2

**OLAP_DML_EXPRESSION**

2-37
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)

<table>
<thead>
<tr>
<th>Time</th>
<th>Product</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Hardware</td>
<td>1.07663806</td>
</tr>
<tr>
<td>2005</td>
<td>Software/Other</td>
<td>1.08295738</td>
</tr>
<tr>
<td>2006</td>
<td>Hardware</td>
<td>1.08707305</td>
</tr>
<tr>
<td>2006</td>
<td>Software/Other</td>
<td>1.09730881</td>
</tr>
</tbody>
</table>

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::=

equation [ASC | DESC] [NULLS {FIRST | LAST}]
Arguments

**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.

**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.

**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)
```
<table>
<thead>
<tr>
<th>Product</th>
<th>Time</th>
<th>Cost</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deluxe Mouse</td>
<td>MAR-02</td>
<td>24.05</td>
<td>1</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>APR-02</td>
<td>23.95</td>
<td>2</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>FEB-02</td>
<td>23.94</td>
<td>3</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>AUG-02</td>
<td>23.88</td>
<td>4</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>MAY-02</td>
<td>23.84</td>
<td>5</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>JAN-02</td>
<td>23.73</td>
<td>6</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>JUL-02</td>
<td>23.73</td>
<td>6</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>JUN-02</td>
<td>23.72</td>
<td>8</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>SEP-02</td>
<td>23.71</td>
<td>9</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>NOV-02</td>
<td>23.65</td>
<td>10</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>DEC-02</td>
<td>23.62</td>
<td>11</td>
</tr>
<tr>
<td>Deluxe Mouse</td>
<td>OCT-02</td>
<td>23.37</td>
<td>12</td>
</tr>
</tbody>
</table>

Related Topics

AVG_RN, DENSE_RN, ROW_RN

**ROW_RN**

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

**ROW_RN** assigns a unique rank to each dimension member; for identical values, the rank is arbitrary. For example, **ROW_RN** 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_RN ( ) 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

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.

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.

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)
### 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

Share is calculated with these formulas:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT</td>
<td>current member/parent</td>
</tr>
</tbody>
</table>
### SHARE

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP</td>
<td>current member/root ancestor</td>
</tr>
<tr>
<td>MEMBER</td>
<td>current member/specified member</td>
</tr>
<tr>
<td>DIMENSION LEVEL</td>
<td>current member/ancestor at specified level or null if the current member is above the specified level.</td>
</tr>
</tbody>
</table>

**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`.

**Example**

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

\[(\text{SHARE(UNITS_CUBE.SALES OF HIERARCHY PRODUCT.PRIMARY PARENT)}) \times 100\]

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

\[(\text{SHARE(UNITS_CUBE.SALES OF HIERARCHY PRODUCT.PRIMARY TOP)}) \times 100\]

<table>
<thead>
<tr>
<th>Product</th>
<th>Parent</th>
<th>Sales</th>
<th>Share of Parent</th>
<th>Share of Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop PCs</td>
<td>HRD</td>
<td>74556528</td>
<td>60</td>
<td>54</td>
</tr>
<tr>
<td>Portable PCs</td>
<td>HRD</td>
<td>18338225</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>CD/DVD</td>
<td>HRD</td>
<td>16129497</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>Memory</td>
<td>HRD</td>
<td>5619219</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Modems/Fax</td>
<td>HRD</td>
<td>5575726</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Monitors</td>
<td>HRD</td>
<td>3972142</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Accessories</td>
<td>SFT</td>
<td>6213535</td>
<td>49</td>
<td>5</td>
</tr>
</tbody>
</table>
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 }
|

<table>
<thead>
<tr>
<th>Product</th>
<th>Parent</th>
<th>Sales</th>
<th>Share of Parent</th>
<th>Share of Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Systems</td>
<td>SFT</td>
<td>4766857</td>
<td>37</td>
<td>3</td>
</tr>
<tr>
<td>Documentation</td>
<td>SFT</td>
<td>1814844</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>Hardware</td>
<td>TOTAL</td>
<td>124191336</td>
<td>91</td>
<td>91</td>
</tr>
<tr>
<td>Software/Other</td>
<td>TOTAL</td>
<td>12795236</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
Arguments

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>Calculates all time periods at the same level. (Default)</td>
</tr>
<tr>
<td>PARENT</td>
<td>Calculates time periods at the same level with the same parent.</td>
</tr>
<tr>
<td>GREGORIAN YEAR</td>
<td>Calculates time periods within the same Gregorian year.</td>
</tr>
<tr>
<td>GREGORIAN QUARTER</td>
<td>Calculates time periods within the same Gregorian quarter.</td>
</tr>
<tr>
<td>GREGORIAN MONTH</td>
<td>Calculates time periods within the same Gregorian month.</td>
</tr>
<tr>
<td>GREGORIAN WEEK</td>
<td>Calculates time periods within the same Gregorian week.</td>
</tr>
<tr>
<td>ANCESTOR</td>
<td>Includes time periods at the same level and with the same ancestor at a specified level.</td>
</tr>
</tbody>
</table>

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.
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.

\[
\text{SUM(UNITS\_CUBE.SALES) OVER (DIMENSION "TIME" BETWEEN 1 PRECEDING AND CURRENT MEMBER WITHIN LEVEL)}
\]

<table>
<thead>
<tr>
<th>Time</th>
<th>Sales</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1.04</td>
<td>146529</td>
<td>289599</td>
</tr>
<tr>
<td>Q2.04</td>
<td>143070</td>
<td>291362</td>
</tr>
<tr>
<td>Q3.04</td>
<td>148292</td>
<td>297820</td>
</tr>
<tr>
<td>Q4.04</td>
<td>149528</td>
<td>293135</td>
</tr>
<tr>
<td>Q1.05</td>
<td>143607</td>
<td>281703</td>
</tr>
<tr>
<td>Q2.05</td>
<td>138096</td>
<td>277049</td>
</tr>
<tr>
<td>Q3.05</td>
<td>138953</td>
<td>284015</td>
</tr>
<tr>
<td>Q4.05</td>
<td>145062</td>
<td>291881</td>
</tr>
</tbody>
</table>

The next example calculates Year-to-Date Sales.

\[
\text{SUM(UNITS\_CUBE.SALES) OVER (HIERARCHY TIME.CALENDAR BETWEEN UNBOUNDED PRECEDING AND CURRENT MEMBER WITHIN ANCESTOR AT DIMENSION LEVEL TIME.CALENDAR\_YEAR)}
\]

<table>
<thead>
<tr>
<th>Time</th>
<th>Sales</th>
<th>Sales YTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN-05</td>
<td>12093518</td>
<td>12093518</td>
</tr>
<tr>
<td>FEB-05</td>
<td>10103162</td>
<td>22196680</td>
</tr>
<tr>
<td>MAR-05</td>
<td>9184658</td>
<td>31381338</td>
</tr>
<tr>
<td>APR-05</td>
<td>9185964</td>
<td>40567302</td>
</tr>
<tr>
<td>MAY-05</td>
<td>11640216</td>
<td>52207519</td>
</tr>
<tr>
<td>JUN-05</td>
<td>16816661</td>
<td>69024079</td>
</tr>
<tr>
<td>JUL-05</td>
<td>11110903</td>
<td>80134982</td>
</tr>
<tr>
<td>AUG-05</td>
<td>9475807</td>
<td>89610789</td>
</tr>
<tr>
<td>SEP-05</td>
<td>12030538</td>
<td>101641328</td>
</tr>
<tr>
<td>OCT-05</td>
<td>11135032</td>
<td>112776359</td>
</tr>
<tr>
<td>NOV-05</td>
<td>11067754</td>
<td>123844113</td>
</tr>
</tbody>
</table>

Related Topics

AVG, COUNT, MAX, MIN
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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>BIN_TO_NUM</td>
<td>CAST</td>
</tr>
<tr>
<td>ACOS</td>
<td>BITAND</td>
<td>CEIL</td>
</tr>
<tr>
<td>ADD_MONTHS</td>
<td>CHARTOROWID</td>
<td>CHAR</td>
</tr>
<tr>
<td>ASCII</td>
<td>CHR</td>
<td>COALESCE</td>
</tr>
<tr>
<td>ASCISTR</td>
<td>COALESCE</td>
<td>CONCAT</td>
</tr>
<tr>
<td>ASIN</td>
<td>COS</td>
<td>COSH</td>
</tr>
<tr>
<td>ATAN</td>
<td>COSH</td>
<td>CURRENT_DATE</td>
</tr>
<tr>
<td>ATAN2</td>
<td>CURRENT_TIMESTAMP</td>
<td>CURRENT_TIMESTAMP</td>
</tr>
</tbody>
</table>
Chapter 3
Row Functions in Alphabetical Order

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)
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:
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
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
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\OE5ne.

**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 - 2π) 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 - 2π) numeric value of the arc tangent of a given ratio. The function returns values in the range of -π to π, 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

\[ \text{BITAND}(7, 29) \text{ returns the value } 5. \]

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 \text{type\_name}.

Syntax

\[ \text{CAST}(\text{expr AS type\_name}) \]

Arguments

\text{expr} can be an expression in one of the data types.

\text{type\_name} is one of the data types listed in Table 1-2.

\textbf{Table 3-1} shows which data types can be cast into which other built-in data types. \text{NUMBER} includes \text{NUMBER}, \text{DECIMAL}, and \text{INTEGER}. \text{DATETIME} includes \text{DATE}, \text{TIMESTAMP}, \text{TIMESTAMP WITH TIMEZONE}, and \text{TIMESTAMP WITH LOCAL TIMEZONE}. \text{INTERVAL} includes \text{INTERVAL DAY TO SECOND} and \text{INTERVAL YEAR TO MONTH}.

\begin{tabular}{|c|c|c|c|c|}
\hline
From & To BINARY\_FLOAT, BINARY\_DOUBLE & To CHAR, VARCHAR2 & To NUMBER & To DATETIME, INTERVAL & To NCHAR, NVARCHAR2 \\
\hline
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 \\
\hline
\end{tabular}

Example

\[ \text{CAST('123.4567' AS NUMBER(10,2)) returns the value 123.46.} \]

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 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

\texttt{EXP}(n)

Arguments

\(n\) is a numeric expression for the exponent.

Example

\texttt{EXP}(4) returns the value \(54.59815\).

**EXTRACT** (datetime)

\texttt{EXTRACT} returns the value of a specified field from a datetime or interval expression.

**Return Value**

\texttt{NUMBER}

**Syntax**

\texttt{EXTRACT} \{ \{ \texttt{YEAR} \ |
\texttt{MONTH} \ |
\texttt{DAY} \ |
\texttt{HOUR} \ |
\texttt{MINUTE} \ |
\texttt{SECOND} \ |
\texttt{TIMEZONE\_HOUR} \ |
\texttt{TIMEZONE\_MINUTE} \ |
\texttt{TIMEZONE\_REGION} \ |
\texttt{TIMEZONE\_ABBREV} \} \}

\texttt{FROM} \{ \texttt{datetime\_value\_expression} \ |
\texttt{interval\_value\_expression} \}

Arguments

\texttt{datetime\_value\_expression} is an expression with a datetime data type.

\texttt{interval\_value\_expression} is an expression with an interval data type.

Example

\texttt{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.

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

**FLOOR**

\texttt{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

\texttt{GREATEST}(\texttt{expr}, \texttt{expr} \ldots)

Arguments

\texttt{expr} can be any expression.

Examples

\texttt{GREATEST('Harry','Harriot','Harold')} \textit{returns the value} Harry.

\texttt{GREATEST(7, 19, 3)} \textit{returns the value} 19.

\textbf{HEXTORAW}

\texttt{HEXTORAW} converts a hexadecimal value to a raw value.

Return Value

\texttt{RAW}

Syntax

\texttt{HEXTORAW} \texttt{(char)}

Arguments

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

Example

\texttt{HEXTORAW('7D')} \textit{returns the RAW value} 7D.

\textbf{INITCAP}

\texttt{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

\texttt{CHAR | NCHAR | VARCHAR2 | NVARCHAR2}

Syntax

\texttt{INITCAP} \texttt{(char)}

Arguments

\texttt{char} can be any text expression.

Example

\texttt{INITCAP('top ten tunes')} \textit{and} \texttt{INITCAP('TOP TEN TUNES')} \textit{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

\texttt{NANVL} \{expression, alternate\}

Arguments

\textit{expression} can be any value.

\textit{alternate} is the numeric value returned if \textit{expression} is not a number.

\textbf{NEW\_TIME}

\texttt{NEW\_TIME} converts the date and time from one time zone to another. Before using this function, set the \texttt{NLS\_DATE\_FORMAT} parameter to display 24-hour time.

Return Value

\texttt{DATE}

Syntax

\texttt{NEW\_TIME\{date, timezone1, timezone2\}}

Arguments

\textit{date} is a datetime expression to be converted to a new time zone.

\textit{timezone1} is the time zone of \textit{date}.

\textit{timezone2} is the new time zone.

The time zone arguments are limited to the values in \texttt{Table 3-2}. For other time zones, use \texttt{FROM\_TZ}.

\textbf{Table 3-2  Time Zones}

<table>
<thead>
<tr>
<th>Time Zone</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alaska-Hawaii Daylight Time</td>
<td>HDT</td>
</tr>
<tr>
<td>Alaska-Hawaii Standard Time</td>
<td>HST</td>
</tr>
<tr>
<td>Atlantic Daylight Time</td>
<td>ADT</td>
</tr>
<tr>
<td>Atlantic Standard Time</td>
<td>AST</td>
</tr>
<tr>
<td>Bering Daylight Time</td>
<td>BDT</td>
</tr>
<tr>
<td>Bering Standard Time</td>
<td>BST</td>
</tr>
<tr>
<td>Central Daylight Time</td>
<td>CDT</td>
</tr>
<tr>
<td>Central Standard Time</td>
<td>CST</td>
</tr>
<tr>
<td>Eastern Daylight Time</td>
<td>EDT</td>
</tr>
<tr>
<td>Eastern Standard Time</td>
<td>EST</td>
</tr>
<tr>
<td>Greenwich Mean Time</td>
<td>GMT</td>
</tr>
<tr>
<td>Mountain Daylight Time</td>
<td>MDT</td>
</tr>
<tr>
<td>Mountain Standard Time</td>
<td>MST</td>
</tr>
<tr>
<td>Newfoundland Standard Time</td>
<td>NST</td>
</tr>
</tbody>
</table>
Table 3-2  (Cont.) Time Zones

<table>
<thead>
<tr>
<th>Time Zone</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pacific Daylight Time</td>
<td>PDT</td>
</tr>
<tr>
<td>Pacific Standard Time</td>
<td>PST</td>
</tr>
<tr>
<td>Yukon Daylight Time</td>
<td>YDT</td>
</tr>
<tr>
<td>Yukon Standard Time</td>
<td>YST</td>
</tr>
</tbody>
</table>

Example

NEW_TIME(SYSDATE, 'PST', 'EST') returns a value such as 18-JAN-07 04:38:07 in
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 [, 'nlsparam' ])

Arguments

char can be any text string.

nlsparam 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 nlsparam, 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 [, 'nlsparam' ])

Arguments
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_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
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.

<table>
<thead>
<tr>
<th>Product</th>
<th>Prices</th>
<th>Hash 5</th>
<th>Seed 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENVY STD</td>
<td>200539.83</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>ENVY EXE</td>
<td>255029.31</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1GB USB DRV</td>
<td>44645.65</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DLX MOUSE</td>
<td>1379.49</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>144MB DISK</td>
<td>3011.43</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>512 USB DRV</td>
<td>22139.99</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>19 SVGA</td>
<td>34837.16</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>56KPS MODEM</td>
<td>12478</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ENVY EXT KBD</td>
<td>4312.22</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>17 SVGA</td>
<td>22605.55</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>EXT CD ROM</td>
<td>17990.14</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
### 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 ('Rumpelstiltakin')) 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_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_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.
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_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:

\[ n2 - (n1 \times N) \]

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**

\[
\text{sign}(n)
\]

**Arguments**

\( n \) is a numeric expression.

**Example**

\[
\text{sign}(-15) \text{ returns the value } -1.
\]

**SIN**

SIN returns the sine of an angle.

**Return Value**

NUMBER

**Syntax**

\[
\text{sine}(n)
\]

**Arguments**

\( n \) is a numeric expression for an angle in radians.

**Example**

\[
\text{sine}(30 \times 3.1415927/180) \text{ calculates the sine of a 30 degrees angle as the value 0.500000007. The numeric expression converts degrees to radians.}
\]

**SINH**

SINH returns the sine of a hyperbolic angle.

**Return Value**

NUMBER

**Syntax**

\[
\text{sineh}(n)
\]

**Arguments**

\( n \) is a numeric expression for a hyperbolic angle.
Example

\( \text{SINH}(1) \) 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**

\( \text{SOUNDEX} \ (\text{char}) \)

**Arguments**

char can be any text expression.

**Example**

All of these examples return the value D500:

\[
\text{soundex('Donna')} \\
\text{soundex('Diane')} \\
\text{soundex('Dana')}
\]

**SQRT**

SQRT returns the square root of a number.

**Return Value**

NUMBER

**Syntax**

\( \text{SQRT} \ (n) \)

**Arguments**

n is a numeric expression for a positive number.

**Example**

\( \text{SQRT}(13) \) 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.
- **SUBSTR** uses bytes instead of characters.
- **SUBSTR** uses Unicode complete characters.

#### Return Value

<table>
<thead>
<tr>
<th>CHAR</th>
<th>NCHAR</th>
<th>VARCHAR2</th>
<th>NVARCHAR2</th>
</tr>
</thead>
</table>

#### Syntax

```sql
{ 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

<table>
<thead>
<tr>
<th>VARCHAR2</th>
</tr>
</thead>
</table>

#### Syntax

```sql
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

<table>
<thead>
<tr>
<th>USERENV Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHENTICATED_IDENTITY</td>
<td>The identity used for authentication, such as database user name, schema name, or operating system login name.</td>
</tr>
<tr>
<td>AUTHENTICATION_METHOD</td>
<td>The method of authentication, such as PASSWORD, OS, or SSL.</td>
</tr>
<tr>
<td>CURRENT_EDITION_ID</td>
<td>The session edition identifier, such as 100.</td>
</tr>
<tr>
<td>CURRENT_EDITION_NAME</td>
<td>The session edition name, such as ORA$BASE.</td>
</tr>
<tr>
<td>CURRENT_SCHEMA</td>
<td>The name of the currently active default schema, such as SH.</td>
</tr>
<tr>
<td>CURRENT_SCHEMA_ID</td>
<td>The numeric identifier of the currently active default schema, such as 80.</td>
</tr>
<tr>
<td>CURRENT_USER</td>
<td>The name of the database user whose privileges are currently active, such as SH.</td>
</tr>
<tr>
<td>CURRENT_USERID</td>
<td>The numeric identifier of the database user whose privileges are currently active, such as 80.</td>
</tr>
<tr>
<td>DATABASE_ROLE</td>
<td>Data Guard role of the database: PRIMARY, PHYSICAL STANDBY, LOGICAL STANDBY, or SNAPSHOT STANDBY.</td>
</tr>
<tr>
<td>DB_DOMAIN</td>
<td>The network domain of the database as specified by the DB_DOMAIN initialization parameter, such as us.example.com.</td>
</tr>
<tr>
<td>DB_NAME</td>
<td>The name of the database as specified by the DB_NAME initialization parameter.</td>
</tr>
<tr>
<td>DB_UNIQUE_NAME</td>
<td>The unique name of the database within the domain as specified by the DB_UNIQUE_NAME initialization parameter.</td>
</tr>
<tr>
<td>ENTERPRISE.IDENTITY</td>
<td>The enterprise-wide identity of the user, or NULL for local users, SYSDBA, and SYSOPER.</td>
</tr>
<tr>
<td>FG_JOB_ID</td>
<td>Job identifier of the current session if a client foreground process opened it; otherwise, NULL.</td>
</tr>
<tr>
<td>GLOBAL_CONTEXT_MEMORY</td>
<td>The number used in the System Global Area by the globally accessed context.</td>
</tr>
<tr>
<td>GLOBAL_UID</td>
<td>The global user identification from Oracle Internet Directory for Enterprise User Security logins; otherwise, NULL.</td>
</tr>
<tr>
<td>HOST</td>
<td>The name of the client host computer.</td>
</tr>
<tr>
<td>IDENTIFICATION_TYPE</td>
<td>The way the user schema was created in the database: LOCAL, EXTERNAL, GLOBAL SHARED, or GLOBAL PRIVATE.</td>
</tr>
<tr>
<td>INSTANCE</td>
<td>The identification number of the current instance, such as 1.</td>
</tr>
<tr>
<td>INSTANCE_NAME</td>
<td>The name of the database instance.</td>
</tr>
<tr>
<td>IP_ADDRESS</td>
<td>The IP address of the client, such as 10.255.255.255.</td>
</tr>
</tbody>
</table>
### Table 3-3  (Cont.) USERENV Attributes

<table>
<thead>
<tr>
<th>USERENV Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISDBA</td>
<td>TRUE if the user was authenticated with DBA privileges; otherwise, FALSE.</td>
</tr>
<tr>
<td>LANG</td>
<td>A short name for the session language, such as US for AMERICAN.</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>The language, territory, and database character set in the form language_territory.characterset, such as AMERICA_AMERICAN.WE8DEC.</td>
</tr>
<tr>
<td>MODULE</td>
<td>The application name set through the DBMS_APPLICATION_INFO package or OCI, such as JDBC Thin Client or SQL Developer.</td>
</tr>
<tr>
<td>NETWORK_PROTOCOL</td>
<td>The network protocol being used for communication, such as TCP.</td>
</tr>
<tr>
<td>NLSCALENDAR</td>
<td>The session calendar, such as GREGORIAN.</td>
</tr>
<tr>
<td>NLS_CURRENCY</td>
<td>The session currency mark, such as $.</td>
</tr>
<tr>
<td>NLS_DATE_FORMAT</td>
<td>The session date format, such as DD-MON-RR.</td>
</tr>
<tr>
<td>NLS_DATE_LANGUAGE</td>
<td>The session date language, such as AMERICAN.</td>
</tr>
<tr>
<td>NLS_SORT</td>
<td>BINARY or a linguistic sort basis, such as XSPANISH.</td>
</tr>
<tr>
<td>NLS_TERRITORY</td>
<td>The session territory, such as AMERICA.</td>
</tr>
<tr>
<td>OS_USER</td>
<td>The operating system user name of the client process that initiated the database session.</td>
</tr>
<tr>
<td>SERVER_HOST</td>
<td>The host name of the computer where the database instance is running.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>The name of the service the session is connected to, such as SYS$USERS.</td>
</tr>
<tr>
<td>SESSION_USER</td>
<td>The database user name or schema name that identified the user at login, such as SH.</td>
</tr>
<tr>
<td>SESSIONID</td>
<td>The session identifier, such as 120456.</td>
</tr>
<tr>
<td>SID</td>
<td>The session number, such as 86.</td>
</tr>
</tbody>
</table>

**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

\[ \text{TANH}(n) \]

Arguments

\( n \) is a numeric expression for a hyperbolic angle.

Example

\[ \text{TANH}(0.5) \text{ returns the value } 0.462117157. \]

TO_BINARY_DOUBLE

\text{TO_BINARY_DOUBLE} \text{ converts a text or numeric expression to a double-precision floating-point number.}

Return Value

BINARY_DOUBLE

Syntax

\( \text{TO_BINARY_DOUBLE}(\text{expr}[, \text{fmt}[, \text{'nlsparam'}]]) \)

Arguments

\( \text{n} \) can be any text or numeric expression.

\( \text{fmt} \) is a text expression that identifies a number format model as described in the \text{Oracle Database SQL Language Reference}.

\( \text{nlsparam} \) 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:

```
'\text{NLS\_NUMERIC\_CHARACTERS = ''dg''}
\text{NLS\_CURRENCY = ''text''}
\text{NLS\_ISO\_CURRENCY = territory}'
```

The \( \text{d} \) is the decimal character, and the \( \text{g} \) 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 \( \text{expr} \).

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(char)
```

**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(datatime) 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:

'EQUALS = '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

\texttt{TO_TIMESTAMP}(\texttt{char}, \texttt{fmt} [ 'nlsparam' ])

Arguments

\texttt{char} is a text expression to be converted.

\texttt{fmt} is a datetime model format specifying the format of \texttt{char}. The default date format is determined implicitly by the \texttt{NLS\_TERRITORY} initialization parameter or can be set explicitly by the \texttt{NLS\_DATE\_FORMAT} parameter. For data type formats, refer to the \textit{Oracle Database SQL Language Reference}.

\texttt{nlsparam} specifies the language in which month and day names and abbreviations given in \texttt{char}. This argument has this form:

'\texttt{NLS\_DATE\_LANGUAGE} = \texttt{language}'

By default, \texttt{char} is in the session date language.

Examples

\texttt{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.

\texttt{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.

\textbf{TO\_TIMESTAMP\_TZ}

\texttt{TO\_TIMESTAMP\_TZ} converts a text expression to a value of \texttt{TIMESTAMP\_WITH\_TIME\_ZONE} data type.

Return Value

\texttt{TIMESTAMP WITH TIME ZONE}

Syntax

\texttt{TO\_TIMESTAMP\_TZ}(\texttt{char}, \texttt{fmt} [ 'nlsparam' ])

Arguments

\texttt{char} is a text expression to be converted.

\texttt{fmt} is a datetime model format specifying the format of \texttt{char}. The default date format is determined implicitly by the \texttt{NLS\_TERRITORY} initialization parameter or can be set explicitly by the \texttt{NLS\_DATE\_FORMAT} parameter. For data type formats, refer to the \textit{Oracle Database SQL Language Reference}.

\texttt{nlsparam} specifies the language in which month and day names and abbreviations given in \texttt{char}. This argument has this form:

'\texttt{NLS\_DATE\_LANGUAGE} = \texttt{language}'

By default, \texttt{char} is in the session date language.
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:mm' |
  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:

<table>
<thead>
<tr>
<th>Channel</th>
<th>VSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog</td>
<td>7</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>12</td>
</tr>
<tr>
<td>Internet</td>
<td>8</td>
</tr>
</tbody>
</table>

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
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**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>[</td>
<td>Left bracket</td>
</tr>
<tr>
<td>(</td>
<td>Left parenthesis</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>-</td>
<td>Minus</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>^=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>+</td>
<td>Plus</td>
</tr>
<tr>
<td>]</td>
<td>Right bracket</td>
</tr>
<tr>
<td>)</td>
<td>Right parenthesis</td>
</tr>
<tr>
<td>*</td>
<td>Star</td>
</tr>
</tbody>
</table>
Index

A
- ancestor function, 2-11
- arguments, 1-14
- average function, 2-5
- AVERAGE_RANK, 2-3
- AVG, 2-5

B
- BETWEEN operator, 1-8
- binary operators, 1-4

C
- CASE expressions, 1-11
- character literal, 1-11
- child count function, 2-12
- comparison operators, 1-6
- concatenation operator, 1-5
- conditions, 1-6
- COUNT, 2-7
- counting functions, 2-7, 2-12
- cumulative averages, 2-5

D
- data types, 1-2
- DENSE_RANK, 2-9
- depth function, 2-14
- difference from future period, 2-28
- difference from prior period, 2-22, 2-24
- DML expressions, 2-37

F
- future period functions, 2-26, 2-28, 2-31

G
- group comparison operators, 1-7

H
- HIER_ANCESTOR, 2-11
- HIER_CHILD_COUNT, 2-12
- HIER_DEPTH, 2-14
- HIER_LEVEL, 2-15
- HIER_ORDER, 2-16
- HIER_PARENT, 2-17
- HIER_TOP, 2-19

I
- identifiers, 1-1
- IF ... THEN ... ELSE, 1-11
- IS operator, 1-9

L
- LAG, 2-20
- LAG_VARIANCE, 2-22
- LAG_VARIANCE_PERCENT, 2-24
- largest values, 2-33
- later period functions, 2-26, 2-28, 2-31
- LEAD, 2-26
- LEAD_VARIANCE, 2-28
- LEAD_VARIANCE_PERCENT, 2-31
- level depth function, 2-14
- level function, 2-15
- LIKE operators, 1-10
- literal expressions, 1-11

M
- MAX, 2-33
- MIN, 2-35
- moving averages, 2-5

N
- naming conventions, 1-1
- NOT operator, 1-9
O
object names, 1-1
OLAP_DML_EXPRESSION, 2-37
ordering function, 2-16

P
parent function, 2-17
percent difference from future period, 2-31
percent difference from prior period, 2-24
period-to-date function, 2-44
prior period functions, 2-20, 2-22, 2-24

Q
qualified data references, 1-14
quotes
double, for reserved words, A-1
in naming conventions, 1-1
single, for text literals, 1-11

R
RANK, 2-38
ranking functions, 2-3, 2-9, 2-38, 2-40
ratios, 2-42
ROW_NUMBER, 2-40

S
SHARE, 2-42
smallest values, 2-35
sorting functions, 2-3, 2-9, 2-16, 2-38, 2-40
SPL expressions, 2-37
string, 1-11
SUM, 2-44

T
text literal, 1-11
top function, 2-19

U
unary operators, 1-4

V
variance functions, 2-22, 2-24, 2-28, 2-31

W
WHEN ... THEN, 1-11
windowing functions, 2-5, 2-7, 2-33, 2-35, 2-44