Oracle® OLAP

Expression Syntax Reference

19c
E96406-01
January 2019

Oracle OLAP Expression Syntax Reference, 19c

E96406-01

Primary Author: David McDermid

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

Contents

2 OLAP Functions

List of Tables

Preface

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

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions
- Backus-Naur Form Syntax

Audience

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

Documentation Accessibility

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

Access to Oracle Support

Oracle customers 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:

<xref cid="1l1qda" format="html" href="olink:OLAUG" scope="external"><i>Oracle OLAP User's Guide</i></xref>

- *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:

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

Backus-Naur Form Syntax

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

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

1 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)

1.1 Dimensional Object Names

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

1.1.1 Syntax

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

Table 1-1 Naming Conventions for Dimensional Objects

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

1.1.2 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.

1.2 Dimensional Data Types

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

Table 1-2 Dimensional Data Types

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

Table 1-2 (Cont.) Dimensional Data Types

Data Type	Description
`NCHAR[(`*`size`*`)]`	A fixed length character string with a length of *size* 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.
`NUMBER [(`*`p`*`[,`*`s`*`])]`	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.
`NVARCHAR2(`*`size`*`)`	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.
`TIMESTAMP[(`*`precision`*`)]`	A valid date that contains the datetime fields `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, and `SECOND`. It does not have a time zone. The precision is the number of digits in the fractional part of the `SECOND` field, which can have a value of 0 to 9. The default precision is 6 digits. The default format is determined explicitly by the `NLS_DATE_FORMAT` parameter and implicitly by the `NLS_TERRITORY` parameter. A `TIMESTAMP` value requires from 7 to 11 bytes depending on the precision.
`TIMESTAMP [(`*`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[(`*`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 TIMEZONE` 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.

1.3 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".

1.3.1 Unary Arithmetic Operators

A unary operator operates on only one operand.

Table 1-3 Unary Operators

Operator	Description
+	Positive value
-	Negative value

1.3.1.1 Syntax

```
operator operand
```

1.3.1.2 Example

`-5` is a negative number.

1.3.2 Binary Arithmetic Operators

A binary operator operates on two operands.

Table 1-4 Binary Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

1.3.2.1 Syntax

```
operand operator operand
```

1.3.2.2 Examples

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

`7 * 2` is 14.

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

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

```
UNITS_CUBE.SALES * 1.06
```

Table 1-5 Multiplication Operator Example

Product	Level	Sales	Sales Budget
Hardware	CLASS	124191336	131642816
Desktop PCs	FAMILY	74556528	79029919
Monitors	FAMILY	3972142	4210470
Memory	FAMILY	5619219	5956372
Modems/Fax	FAMILY	5575726	5910269
CD/DVD	FAMILY	16129497	17097267
Portable PCs	FAMILY	18338225	19438518

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

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

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

1.3.3 Concatenation Operator

The concatenation operator (||) combines text expressions.

1.3.3.1 Syntax

```
operand || operand
```

1.3.3.2 Example

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

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

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

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

1.4 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)`

1.4.1 Simple Comparison Conditions

Comparison conditions compare one expression with another.

You can use these comparison operators:

Table 1-6 Simple Comparison Operators

Operator	Description
>	Greater than

Table 1-6 (Cont.) Simple Comparison Operators

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

1.4.1.1 Syntax

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

1.4.1.2 Arguments

`expr` can be any expression.

1.4.1.3 Examples

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

1.4.2 Group Comparison Conditions

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

You can use these comparison operators:

Table 1-7 Group Comparison Operators

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

1.4.2.1 Syntax

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

```
{ ANY | SOME | ALL }
({ expression_list | subquery })
```

1.4.2.2 Examples

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

1.4.3 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.

1.4.3.1 Syntax

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

1.4.3.2 Arguments

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

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

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

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

`ROOT_ANCESTOR` compares *expr1* as the highest-level ancestor of *expr2*.

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

`LEAF_DESCENDANT` compares *expr1* as a descendant that has no children of *expr2*.

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

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

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

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

1.4.3.3 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
```

1.4.4 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.

1.4.4.1 Syntax

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

1.4.4.2 Example

`7 NOT BETWEEN 10 AND 15` is true.

1.4.5 Multiple Conditions

Conjunctions compare a single expression with two conditions.

Table 1-8 Conjunctions

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

1.4.5.1 Syntax

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

1.4.5.2 Example

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

1.4.6 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.

1.4.6.1 Syntax

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

1.4.6.2 Example

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

1.4.7 Special Conditions

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

Table 1-9 Special Conditions Operators

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

1.4.7.1 Syntax

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

1.4.7.2 Example

`13 IS NOT NULL` is true.

1.4.8 Pattern-Matching Conditions

The pattern-matching conditions compare character data.

1.4.8.1 LIKE Operators

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

You can choose from these `LIKE` operators:

Table 1-10 LIKE Pattern-Matching Operators

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

1.4.8.2 Syntax

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

1.4.8.3 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 @.

1.4.8.4 Examples

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

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

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

1.5 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.

1.5.1 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
```

1.6 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.

1.6.1 Return Value

Same as the else_expression argument

1.6.2 Syntax

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

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

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

1.6.3 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.

1.6.4 Examples

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

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

Product	Package	Category
1.44MB External 3.5" Diskette	Executive	Value Pack
1GB USB Drive	--	Single Item
512MB USB Drive	--	Single Item
56Kbps V.90 Type II Modem	Executive	Value Pack
56Kbps V.92 Type II Fax/Modem	Laptop Value Pack	Value Pack
Deluxe Mouse	Executive	Value Pack
Envoy Ambassador	--	Single Item
Envoy Executive	Executive	Value Pack
Envoy External Keyboard	Executive	Value Pack
Envoy Standard Laptop	Value Pack	Value Pack
External - DVD-RW - 8X	Executive	Value Pack
External 101-key keyboard	Multimedia	Value Pack
External 48X CD-ROM	--	Single Item
Internal - DVD-RW - 6X	Multimedia	Value Pack

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

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

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

Product	Cost	Price
Sentinel Standard	1572.98	1610.53

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

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

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

1.7 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.

1.7.1 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.

1.7.2 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`.

1.7.3 Examples

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

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

2 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

2.1 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

2.2 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

2.2.1 Aggregate Functions

AVERAGE_RANK
AVG
COUNT
DENSE_RANK
MAX
MIN
RANK
SUM

2.2.2 Analytic Functions

AVERAGE_RANK
AVG
COUNT
DENSE_RANK
LAG
LAG_VARIANCE
LEAD_VARIANCE_PERCENT
MAX
MIN
RANK
ROW_NUMBER
SUM

2.2.3 Hierarchical Functions

HIER_ANCESTOR
HIER_CHILD_COUNT
HIER_DEPTH
HIER_LEVEL
HIER_ORDER
HIER_PARENT
HIER_TOP

2.2.4 Lag Functions

LAG

LAG_VARIANCE
LAG_VARIANCE_PERCENT
LEAD
LEAD_VARIANCE
LEAD_VARIANCE_PERCENT

2.2.5 OLAP DML Functions

OLAP_DML_EXPRESSION

2.2.6 Rank Functions

AVERAGE_RANK
DENSE_RANK
RANK
ROW_NUMBER

2.2.7 Share Functions

SHARE

2.2.8 Window Functions

AVG
COUNT
MAX
MIN
SUM

2.3 AVERAGE_RANK

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

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

Return Value

NUMBER

Syntax

AVERAGE_RANK ( ) OVER (rank_clause**)**

rank_clause::=

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

```
                             }
                  }
       ]
```

order_by_clause::=

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

Arguments

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

hierarchy_id
The hierarchy over which the values are calculated. If *`dimension_id`* is used instead, the default hierarchy is used.

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

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

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

NULLS {FIRST | LAST}
Determines whether members with null values are listed first or last.

WITHIN
Selects a set of related dimension members to be ranked.
`LEVEL` ranks all members at the same level.
`PARENT` ranks members at the same level with the same parent.
`ANCESTOR` ranks all members at the same level and with the same ancestor at a specified level.

dim_level_id
The name of a level of *`dimension_id`*.

hier_level_id
The name of a level of *`hierarchy_id`*.

Example

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

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

Product	Time	Cost	Average Rank
Deluxe Mouse	MAR-02	24.05	1
Deluxe Mouse	APR-02	23.95	2

Product	Time	Cost	Average Rank
Deluxe Mouse	FEB-02	23.94	3
Deluxe Mouse	AUG-02	23.88	4
Deluxe Mouse	MAY-02	23.84	5
Deluxe Mouse	JAN-02	23.73	6.5
Deluxe Mouse	JUL-02	23.73	6.5
Deluxe Mouse	JUN-02	23.72	8
Deluxe Mouse	SEP-02	23.71	9
Deluxe Mouse	NOV-02	23.65	10
Deluxe Mouse	DEC-02	23.62	11
Deluxe Mouse	OCT-02	23.37	12

Related Topics

DENSE_RANK, RANK, ROW_NUMBER

2.4 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.

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

Example

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

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

TIME	PARENT	UNITS	AVERAGE
Q1.05	CY2005	143607	143607
Q2.05	CY2005	138096	140852
Q3.05	CY2005	138953	140219
Q4.05	CY2005	145062	141430
Q1.06	CY2006	146819	146819
Q2.06	CY2006	145233	146026
Q3.06	CY2006	143572	145208
Q4.06	CY2006	149305	146232

Related Topics

COUNT, MAX, MIN, SUM

2.5 COUNT

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

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

Return Value

`NUMBER`

Syntax

```
COUNT (value_expr) OVER (window_clause)
```

window_clause::=

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

preceding_boundary::=

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

following_boundary::=

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

Arguments

value_expr
A dimensional expression whose values you want to calculate.

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

hierarchy_id
The hierarchy over which the values are calculated. If *dimension_id* is used instead, the default hierarchy is used.

dim_level_id
The name of a level of *dimension_id*.

hier_level_id
The name of a level of *hierarchy_id*.

boundaries
The *preceding_boundary* and *following_boundary* identify a range of time periods within the group identified by the dimension or hierarchy.
UNBOUNDED starts with the first period or ends with the last period of the group.
CURRENT MEMBER starts or ends the calculation at the current time period.

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

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

Range	Description
LEVEL	Calculates all time periods at the same level. (Default)
PARENT	Calculates time periods at the same level with the same parent.
GREGORIAN YEAR	Calculates time periods within the same Gregorian year.
GREGORIAN QUARTER	Calculates time periods within the same Gregorian quarter.

Range	Description
`GREGORIAN MONTH`	Calculates time periods within the same Gregorian month.
`GREGORIAN WEEK`	Calculates time periods within the same Gregorian week.
`ANCESTOR`	Includes time periods at the same level and with the same ancestor at a specified level.

Example

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

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

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

Related Topics

AVG, MAX, MIN, SUM

2.6 DENSE_RANK

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

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

Return Value

`NUMBER`

Syntax

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

rank_clause::=

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

order_by_clause::=

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

Arguments

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

hierarchy_id
The hierarchy over which the values are calculated. If *`dimension_id`* is used instead, the default hierarchy is used.

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

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

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

NULLS {FIRST | LAST}
Determines whether members with null values are listed first or last.

WITHIN
Selects a set of related dimension members to be ranked.
`LEVEL` ranks all members at the same level.
`PARENT` ranks members at the same level with the same parent.
`ANCESTOR` ranks all members at the same level and with the same ancestor at a specified level.

dim_level_id
The name of a level of *`dimension_id`*.

hier_level_id
The name of a level of *`hierarchy_id`*.

Example

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

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

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

Related Topics

AVERAGE_RANK, RANK, ROW_NUMBER

2.7 HIER_ANCESTOR

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

Return Value

`VARCHAR2`

Syntax

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

Arguments

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

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

hierarchy_id
The hierarchy over which the values are calculated. If `dimension_id` is used instead, the default hierarchy is used.

dim_level_id
The level of the ancestor in `dimension_id`.

hier_level_id
The level of the ancestor in `hierarchy_id`.

Example

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

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

Time	Ancestor
2006	--
Q1.06	CY2006.Q1
Q2.06	CY2006.Q2
Q3.06	CY2006.Q3
Q4.06	CY2006.Q4
JAN-06	CY2006.Q1
FEB-06	CY2006.Q1
MAR-06	CY2006.Q1
APR-06	CY2006.Q2
MAY-06	CY2006.Q2
JUN-06	CY2006.Q2
JUL-06	CY2006.Q3
AUG-06	CY2006.Q3
SEP-06	CY2006.Q3
OCT-06	CY2006.Q4
NOV-06	CY2006.Q4
DEC-06	CY2006.Q4

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

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

2.8 HIER_CHILD_COUNT

`HIER_CHILD_COUNT` returns the number of children of either all dimension members in a hierarchy or a particular member. The hierarchy can be either level-based or value-based.

Return Value

`NUMBER`

Syntax

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

Arguments

member_expression

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

dimension_id

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

hierarchy_id

The hierarchy over which the values are calculated. If *`dimension_id`* is used instead, the default hierarchy is used.

Example

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

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

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

Time	Children
JUL-06	0
AUG-06	0
SEP-06	0
OCT-06	0
NOV-06	0
DEC-06	0

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

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

2.9 HIER_DEPTH

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

Return Value

`NUMBER`

Syntax

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

Arguments

member_expression

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

dimension_id

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

hierarchy_id

The hierarchy over which the values are calculated. If *`dimension_id`* is used instead, the default hierarchy is used.

Example

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

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

Time	Depth
2006	1
Q1.06	2
Q2.06	2
Q3.06	2
Q4.06	2
JAN-06	3
FEB-06	3
MAR-06	3
APR-06	3
MAY-06	3
JUN-06	3
JUL-06	3
AUG-06	3
SEP-06	3
OCT-06	3
NOV-06	3
DEC-06	3

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

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

2.10 HIER_LEVEL

`HIER_LEVEL` returns the level of either all members of a hierarchy or a particular member. The hierarchy must be level-based.

Return Value

`VARCHAR2`

Syntax

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

Arguments

member_expression

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

dimension_id

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

hierarchy_id

The hierarchy over which the values are calculated. If `dimension_id` is used instead, the default hierarchy is used.

Example

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

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

Time	Level
2006	CALENDAR_YEAR
Q1.06	CALENDAR_QUARTER
Q2.06	CALENDAR_QUARTER
Q3.06	CALENDAR_QUARTER
Q4.06	CALENDAR_QUARTER
JAN-06	MONTH
FEB-06	MONTH
MAR-06	MONTH
APR-06	MONTH
MAY-06	MONTH
JUN-06	MONTH
JUL-06	MONTH
AUG-06	MONTH
SEP-06	MONTH
OCT-06	MONTH
NOV-06	MONTH
DEC-06	MONTH

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

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

2.11 HIER_ORDER

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

Return Value

`NUMBER`

Syntax

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

Arguments

member_expression

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

dimension_id

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

hierarchy_id

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

Example

This example orders the values of the Time dimension:

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

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

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

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

2.12 HIER_PARENT

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

Return Value

`VARCHAR2`

Syntax

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

Arguments

member_expression

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

dimension_id

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

hierarchy_id

The hierarchy over which the values are calculated. If *`dimension_id`* is used instead, the default hierarchy is used.

Example

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

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

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

Time	Parent
JUL-06	CY2006.Q3
AUG-06	CY2006.Q3
SEP-06	CY2006.Q3
OCT-06	CY2006.Q4
NOV-06	CY2006.Q4
DEC-06	CY2006.Q4

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

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

2.13 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")
```

Time	Top
2006	TOTAL

Time	Top
Q1.06	TOTAL
Q2.06	TOTAL
Q3.06	TOTAL
Q4.06	TOTAL
JAN-06	TOTAL
FEB-06	TOTAL
MAR-06	TOTAL
APR-06	TOTAL
MAY-06	TOTAL
JUN-06	TOTAL
JUL-06	TOTAL
AUG-06	TOTAL
SEP-06	TOTAL
OCT-06	TOTAL
NOV-06	TOTAL
DEC-06	TOTAL

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

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

2.14 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

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

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)
```

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

Related Topics

LAG_VARIANCE, LAG_VARIANCE_PERCENT, LEAD

2.15 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

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

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.

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

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

Related Topics

LAG, LAG_VARIANCE_PERCENT, LEAD

2.16 LAG_VARIANCE_PERCENT

`LAG_VARIANCE_PERCENT` returns the percent difference between values for the current time period and an earlier period.

Return Value

`NUMBER`

Syntax

```
LAG_VARIANCE_PERCENT (lag_args) OVER (lag_clause)
```

lag_args::=

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

lag_clause::=

```
[ {DIMENSION dimension_id | HIERARCHY hierarchy_id} ]
[ [BY] { LEVEL
       | PARENT
```

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

Arguments

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

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.

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

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

Related Topics

LAG, LAG_VARIANCE, LEAD

2.17 LEAD

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

Return Value

The same data type as the value expression

Syntax

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

lead_args::=

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

lead_clause::=

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

Arguments

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

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)
```

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

Related Topics

LAG, LEAD_VARIANCE, LEAD_VARIANCE_PERCENT

2.18 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

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

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.

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

Time	Parent	Units	Next Qtr	Difference
`2005`	TOTAL	565718	--	--
`Q1-05`	CY2005	143607	138096	5511
`Q2-05`	CY2005	138096	138953	-857
`Q3-05`	CY2005	138953	145062	-6109
`Q4-05`	CY2005	145062	146819	-1757
`Jan-05`	CY2005.Q1	50098	40223	9875
`Feb-05`	CY2005.Q1	43990	45477	-1487

Time	Parent	Units	Next Qtr	Difference
Mar-05	CY2005.Q1	49519	52396	-2877
Apr-05	CY2005.Q2	40223	45595	-5372
May-05	CY2005.Q2	45477	46882	-1405
Jun-05	CY2005.Q2	52396	46476	5920
Jul-05	CY2005.Q3	45595	47476	-1881
Aug-05	CY2005.Q3	46882	47496	-614
Sep-05	CY2005.Q3	46476	50090	-3614
Oct-05	CY2005.Q4	47476	47776	-300
Nov-05	CY2005.Q4	47496	47695	-199
Dec-05	CY2005.Q4	50090	51348	-1258

Related Topics

LAG, LEAD, LEAD_VARIANCE_PERCENT

2.19 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

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

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)
```

Time	Parent	Units	Next Qtr	Difference	Percent
2005	TOTAL	565718	--	--	--
Q1-05	CY2005	143607	138096	5511	.04
Q2-05	CY2005	138096	138953	-857	-.01
Q3-05	CY2005	138953	145062	-6109	-.04
Q4-05	CY2005	145062	146819	-1757	-.01
Jan-05	CY2005.Q1	50098	40223	9875	.25
Feb-05	CY2005.Q1	43990	45477	-1487	-.03
Mar-05	CY2005.Q1	49519	52396	-2877	-.05
Apr-05	CY2005.Q2	40223	45595	-5372	-.12
May-05	CY2005.Q2	45477	46882	-1405	-.03
Jun-05	CY2005.Q2	52396	46476	5920	.13
Jul-05	CY2005.Q3	45595	47476	-1881	-.04
Aug-05	CY2005.Q3	46882	47496	-614	-.01
Sep-05	CY2005.Q3	46476	50090	-3614	-.07
Oct-05	CY2005.Q4	47476	47776	-300	-.01
Nov-05	CY2005.Q4	47496	47695	-199	0
Dec-05	CY2005.Q4	50090	51348	-1258	-.02

Related Topics

LAG, LEAD, LEAD_VARIANCE

2.20 MAX

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

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

Return Value

NUMBER

Syntax

```
MAX (value_expr) OVER (window_clause)
```

window_clause::=

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

preceding_boundary::=

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

following_boundary::=

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

Arguments

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

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.TIME.CALENDAR_YEAR)
```

Time	Units	Maximum
JAN-06	47776	47776
FEB-06	47695	47776
MAR-06	51348	51348
APR-06	47005	51348
MAY-06	52809	52809
JUN-06	45419	52809
JUL-06	48388	52809
AUG-06	48830	52809
SEP-06	46354	52809
OCT-06	47411	52809
NOV-06	46842	52809
DEC-06	55052	55052

Related Topics

AVG, COUNT, MIN, SUM

2.21 MIN

`MIN` returns the smallest of a selection of data values calculated over a Time dimension.

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

Return Value

`NUMBER`

Syntax

```
MIN (value_expr) OVER (window_clause)
```

window_clause::=

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

preceding_boundary::=

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

following_boundary::=

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

Arguments

Range	Description
`LEVEL`	Calculates all time periods at the same level. (Default)
`PARENT`	Calculates time periods at the same level with the same parent.
`GREGORIAN YEAR`	Calculates time periods within the same Gregorian year.
`GREGORIAN QUARTER`	Calculates time periods within the same Gregorian quarter.

Range	Description
`GREGORIAN MONTH`	Calculates time periods within the same Gregorian month.
`GREGORIAN WEEK`	Calculates time periods within the same Gregorian week.
`ANCESTOR`	Includes time periods at the same level and with the same ancestor at a specified level.

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)
```

Time	Units	Minimum
JAN-06	47776	45419
FEB-06	47695	45419
MAR-06	51348	45419
APR-06	47005	45419
MAY-06	52809	45419
JUN-06	45419	45419
JUL-06	48388	46354
AUG-06	48830	46354
SEP-06	46354	46354
OCT-06	47411	46842
NOV-06	46842	46842
DEC-06	55052	55052

Related Topics

AVG, COUNT, MAX, SUM

2.22 OLAP_DML_EXPRESSION

`OLAP_DML_EXPRESSION` executes an expression in the OLAP DML language.

Return Value

The data type specified in the syntax

Syntax

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

Arguments

expression

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

datatype

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

Example

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

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

Time	Product	Random
2005	Hardware	1.07663806

Time	Product	Random
2005	Software/Other	1.08295738
2006	Hardware	1.08707305
2006	Software/Other	1.09730881

2.23 RANK

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

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

Return Value

NUMBER

Syntax

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

rank_clause::=

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

order_by_clause::=

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

Arguments

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)
```

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

Related Topics

AVERAGE_RANK, DENSE_RANK, ROW_NUMBER

2.24 ROW_NUMBER

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

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

Return Value

`NUMBER`

Syntax

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

rank_clause::=

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

order_by_clause::=

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

Arguments

`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)
```

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

Related Topics

AVERAGE_RANK, DENSE_RANK, RANK

2.25 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:

Keyword	Formula
PARENT	*current member/parent*
TOP	*current member/root ancestor*
MEMBER	*current member/specified member*
DIMENSION LEVEL	*current member/ancestor at specified level* or null if the current member is above the specified level.

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.

```
(SHARE(UNITS_CUBE.SALES OF HIERARCHY PRODUCT.PRIMARY PARENT))*100
```

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

```
(SHARE(UNITS_CUBE.SALES OF HIERARCHY PRODUCT.PRIMARY TOP))*100
```

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

2.26 SUM

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

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

Return Value

`NUMBER`

Syntax

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

window_clause::=

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

preceding_boundary::=

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

following_boundary::=

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

Arguments

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

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.

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

Time	Sales	Sum
Q1.04	146529	289599
Q2.04	143070	291362
Q3.04	148292	297820
Q4.04	149528	293135
Q1.05	143607	281703
Q2.05	138096	277049
Q3.05	138953	284015
Q4.05	145062	291881

The next example calculates Year-to-Date Sales.

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

Time	Sales	Sales YTD
JAN-05	12093518	12093518
FEB-05	10103162	22196680
MAR-05	9184658	31381338

Time	Sales	Sales YTD
APR-05	9185964	40567302
MAY-05	11640216	52207519
JUN-05	16816561	69024079
JUL-05	11110903	80134982
AUG-05	9475807	89610789
SEP-05	12030538	101641328
OCT-05	11135032	112776359
NOV-05	11067754	123844113

Related Topics

AVG, COUNT, MAX, MIN

3
Row Functions

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

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

- Row Functions in Alphabetical Order
- Row Functions By Category

3.1 Row Functions in Alphabetical Order

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

A

ABS
ACOS
ADD_MONTHS
ASCII
ASCIISTR
ASIN
ATAN
ATAN2

B

BIN_TO_NUM
BITAND

C

CAST
CEIL
CHARTOROWID
CHR
COALESCE
CONCAT
COS
COSH
CURRENT_DATE
CURRENT_TIMESTAMP

D

E

F

G

H

I

L

M

N

3.2 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

3.2.1 Numeric Functions

These functions accept numeric input and return numeric values:

- ABS
- ACOS
- ASIN
- ATAN
- ATAN2
- BITAND
- CEIL
- COS
- COSH
- EXP
- FLOOR
- LN
- LOG
- MOD
- NANVL
- POWER
- REMAINDER
- ROUND (number)
- SIGN
- SIN
- SINH
- SQRT
- TAN
- TANH
- TRUNC (number)
- WIDTH_BUCKET

3.2.2 Character Functions That Return Characters

These functions accept character input and return character values:

- CHR
- CONCAT
- INITCAP
- LOWER
- LPAD
- LTRIM
- NLS_CHARSET_NAME
- NLS_INITCAP
- NLS_LOWER
- NLS_UPPER
- NLSSORT
- REGEXP_REPLACE
- REGEXP_SUBSTR
- REPLACE
- RPAD
- RTRIM
- SOUNDEX
- SUBSTR
- TRANSLATE

TRIM
UPPER

3.2.3 NLS Character Functions

These functions return information about a character set:

NLS_CHARSET_ID
NLS_CHARSET_NAME

3.2.4 Character Functions That Return Numbers

These functions accept character input and return numeric values:

ASCII
INSTR
LENGTH
REGEXP_COUNT
REGEXP_INSTR

3.2.5 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

3.2.6 General Comparison Functions

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

GREATEST
LEAST

3.2.7 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

3.2.8 Encoding and Decoding Function

These functions return a numeric value for each input value:

DECODE
ORA_HASH
VSIZE

3.2.9 Null-Related Functions

These functions facilitate null handling:

COALESCE
LNNVL
NANVL
NULLIF
NVL

NVL2

3.2.10 Environment and Identifier Functions

These functions provide information about the instance and the session:

SYS_CONTEXT
UID
USER

3.3 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.

3.4 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.

3.5 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.

3.6 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.

3.7 ASCIISTR

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

Return Value

VARCHAR2

Syntax

ASCIISTR(char)

Arguments

char can be any character string.

Example

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

3.8 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`.

3.9 ATAN

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

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

Return Value

`NUMBER`

Syntax

`ATAN``(n)`

Arguments

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

Example

`ATAN(.3)` returns the value `0.291456794`.

3.10 ATAN2

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

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

Return Value

NUMBER

Syntax

ATAN2(n1, n2)

Arguments

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

Example

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

3.11 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.

3.12 BITAND

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

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

Return Value

NUMBER

Syntax

BITAND(expr1, expr2)

Arguments

expr1 and expr2 are numeric expressions for nonnegative integers.

Example

`BITAND(7, 29)` returns the value `5`.

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

3.13 CAST

`CAST` converts values from one data type to another.

Return Value

The data type specified by `type_name`.

Syntax

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

Arguments

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

`type_name` is one of the data types listed in Table 1-2.

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

Table 3-1 Compatible Data Types

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

Example

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

3.14 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`.

3.15 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.

3.16 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))`.

3.17 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`.

3.18 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`.

3.19 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.

3.20 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`.

3.21 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.

3.22 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.

3.23 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.

3.24 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

3.25 EXP

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

Return Value

NUMBER

Syntax

```
EXP(n)
```

Arguments

`n` is a numeric expression for the exponent.

Example

`EXP(4)` returns the value `54.59815`.

3.26 EXTRACT (datetime)

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

Return Value

`NUMBER`

Syntax

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

Arguments

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

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

Example

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

`EXTRACT(TIMEZONE_HOUR FROM CURRENT_TIMESTAMP)` returns the value `-7` from the same example.

3.27 FLOOR

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

Return Value

NUMBER

Syntax

FLOOR(n)

Arguments

n can be any numeric expression.

Examples

FLOOR(15.7) returns the value 15.

FLOOR(-15.7) returns the value -16.

3.28 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.

3.29 GREATEST

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

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

Return Value

The data type of the first expression

Syntax

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

Arguments

`expr` can be any expression.

Examples

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

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

3.30 HEXTORAW

`HEXTORAW` converts a hexadecimal value to a raw value.

Return Value

`RAW`

Syntax

```
HEXTORAW (char)
```

Arguments

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

Example

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

3.31 INITCAP

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

Return Value

`CHAR | NCHAR | VARCHAR2 | NVARCHAR2`

Syntax

```
INITCAP(char)
```

Arguments

`char` can be any text expression.

Example

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

3.32 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.

3.33 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`.

3.34 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`.

3.35 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`.

3.36 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`.

3.37 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).

3.38 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`.

3.39 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`.

3.40 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`.

3.41 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`.

3.42 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`.

3.43 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`.

3.44 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`.

3.45 NANVL

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

Return Value

datatype

Syntax

```
NANVL (expression, alternate)
```

Arguments

expression can be any value.

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

3.46 NEW_TIME

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

Return Value

DATE

Syntax

```
NEW_TIME(date, timezone1, timezone2)
```

Arguments

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

timezone1 is the time zone of date.

timezone2 is the new time zone.

The time zone arguments are limited to the values in Table 3-2. For other time zones, use FROM_TZ.

Table 3-2 Time Zones

Time Zone	Abbreviation
Alaska-Hawaii Daylight Time	HDT
Alaska-Hawaii Standard Time	HST
Atlantic Daylight Time	ADT
Atlantic Standard Time	AST
Bering Daylight Time	BDT
Bering Standard Time	BST

Table 3-2 (Cont.) Time Zones

Time Zone	Abbreviation
Central Daylight Time	CDT
Central Standard Time	CST
Eastern Daylight Time	EDT
Eastern Standard Time	EST
Greenwich Mean Time	GMT
Mountain Daylight Time	MDT
Mountain Standard Time	MST
Newfoundland Standard Time	NST
Pacific Daylight Time	PDT
Pacific Standard Time	PST
Yukon Daylight Time	YDT
Yukon Standard Time	YST

Example

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

3.47 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`.

3.48 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.

3.49 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.

3.50 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.

3.51 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

char can be any text expression.

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

Example

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

3.52 NLS_UPPER

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

Return Value

CHAR | NCHAR | VARCHAR2 | NVARCHAR2

Syntax

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

Arguments

char can be any text expression.

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

Example

NLS_UPPER('This is an emergency') returns the string THIS IS AN EMERGENCY.

3.53 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.

3.54 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`.

3.55 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.

3.56 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`.

3.57 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`.

3.58 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`.

3.59 ORA_HASH

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

Return Value

`NUMBER`

Syntax

```
ORA_HASH  (expr [, max_bucket [, seed_value ] ])
```

Arguments

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

`max_bucket` is the maximum bucket number. For example, when `max_bucket` is set to 5, `ORA_HASH` returns values of 0 to 5, creating six buckets. Set this value from 0 to 4294967295 or 2^32-1 (default).

`seed_value` is a value used by `ORA_HASH` to generate the hash values. Enter a different `seed_value` for different results. Set this value from 0 (default) to 4294967295 or 2^32-1.

Example

`ORA_HASH(PRODUCT_CUBE.PRICES, 5)` returns a value in the range of 0 to 5 for each value of the Prices measure, as shown in the Hash 5 column. The rows are also sorted on the Hash 5 column.

`ORA_HASH(PRODUCT_CUBE.PRICES, 5, 13)` also returns values in the range of 0 to 5, but uses a different seed.

Product	Prices	Hash 5	Seed 13
ENVY STD	200539.83	0	4
ENVY EXE	255029.31	0	5
1GB USB DRV	44645.65	1	2

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

3.60 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`.

3.61 RAWTOHEX

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

Return Value

`VARCHAR2`

Syntax

```
RAWTOHEX(raw)
```

Arguments

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

Example

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

3.62 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.

3.63 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`

3.64 REGEXP_INSTR

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

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

Return Value

`NUMBER`

Syntax

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

Arguments

`source_char` is the text expression to search.

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

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

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

`return_option` is either 0 to return the position of the match (default), or 1 to return the position of the character following the match.

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

- `c`: Case-sensitive matching.
- `i`: Case-insensitive matching.

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

Example

`REGEXP_INSTR('Mississippi', 'i', 1, 3)` searches the string `Mississippi` for the third instance of the letter `i`, beginning the search at the first letter. It returns the value `8`.

3.65 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`.

3.66 REMAINDER

`REMAINDER` returns a rounded remainder when one number is divided by another using this equation:

```
n2 - (n1 * N)
```

where `N` is the integer nearest `n2/n1`.

Return Value

`NUMBER`

Syntax

```
REMAINDER(n2, n1)
```

Arguments

`n1` is a numeric expression for the divisor.

`n2` is a numeric expression for the dividend.

Example

`REMAINDER(18,7)` returns the value `-3`.

3.67 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.

3.68 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.

3.69 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`.

3.70 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.

3.71 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.

3.72 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`.

3.73 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`.

3.74 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.

3.75 SIGN

`SIGN` returns a value that indicates whether a specified number is less than, equal to, or greater than 0 (zero):

- `-1` if `n < 0`
- `0` if `n = 0`
- `1` if `n > 0`

Return Value

`NUMBER`

Syntax

`SIGN``(n)`

Arguments

`n` is a numeric expression.

Example

`SIGN(-15)` returns the value `-1`.

3.76 SIN

`SIN` returns the sine of an angle.

Return Value

`NUMBER`

Syntax

`SIN``(n)`

Arguments

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

Example

`SIN(30 * 3.1415927/180)` calculates the sine of a 30 degrees angle as the value `0.500000007`. The numeric expression converts degrees to radians.

3.77 SINH

`SINH` returns the sine of a hyperbolic angle.

Return Value

`NUMBER`

Syntax

```
SINH(n)
```

Arguments

`n` is a numeric expression for a hyperbolic angle.

Example

`SINH(1)` returns the value `1.17520119`.

3.78 SOUNDEX

`SOUNDEX` returns a character string containing the phonetic representation of a text expression. This function lets you compare words that are spelled differently but sound alike.

The function is based on the algorithm described in Donald Knuth's *The Art of Computer Programming*. This algorithm was designed specifically for English. Its results for other languages other than English are unpredictable and typically unsatisfactory.

Return Value

`VARCHAR2`

Syntax

```
SOUNDEX (char)
```

Arguments

`char` can be any text expression.

Example

All of these examples return the value `D500`:

```
soundex('Donna')
```

```
soundex('Diane')
```

```
soundex('Dana')
```

3.79 SQRT

`SQRT` returns the square root of a number.

Return Value

`NUMBER`

Syntax

```
SQRT(n)
```

Arguments

`n` is a numeric expression for a positive number.

Example

`SQRT(13)` returns the value `3.60555128`.

3.80 SUBSTR

`SUBSTR` returns a portion of string, beginning at a specified character position and extending a specified number of characters.

- `SUBSTR` calculates lengths using characters as defined by the input character set.
- `SUBSTRB` uses bytes instead of characters.
- `SUBSTRC` uses Unicode complete characters.

Return Value

```
CHAR | NCHAR | VARCHAR2 | NVARCHAR2
```

Syntax

```
{ SUBSTR | SUBSTRB | SUBSTRC }(char, position [, substring_length ])
```

Arguments

`char` is a text expression that provides the base string from which the substring is derived.

`position` identifies the first character of the substring:

- When `position` is positive, then the function counts from the beginning of `char` to find the first character.
- When `position` is negative, then the function counts backward from the end of `char`.
- When `position` is 0 (zero), then the first character is the beginning of the substring.

`substring_length` is the number of characters in the returned string. By default, the function returns all characters to the end of the base string. If you specify a value less than 1, then the function returns a null.

Examples

`SUBSTR('firefly', 1, 4)` returns the substring `fire`.

`SUBSTR('firefly', -3, 3)` returns the substring `fly`.

3.81 SYS_CONTEXT

SYS_CONTEXT returns the value of an attribute of a named context. The context, attribute, and value must already be defined in the database. If the context is not defined, SYS_CONTEXT returns NULL.

Return Value

VARCHAR2

Syntax

```
SYS_CONTEXT ('namespace', 'parameter')
```

Arguments

namespace can be any named context in the database. USERENV is a built-in context that describes the current session.

parameter is a defined attribute of namespace. Table 3-3 describes the predefined attributes of USERENV that are most likely to have values. For a complete list, refer to the SYS_CONTEXT entry in the *Oracle Database SQL Language Reference*.

Table 3-3 USERENV Attributes

USERENV Attribute	Description
AUTHENTICATED_IDENTITY	The identity used for authentication, such as database user name, schema name, or operating system login name.
AUTHENTICATION_METHOD	The method of authentication, such as PASSWORD, OS, or SSL.
CURRENT_EDITION_ID	The session edition identifier, such as 100.
CURRENT_EDITION_NAME	The session edition name, such as ORA$BASE.
CURRENT_SCHEMA	The name of the currently active default schema, such as SH.
CURRENT_SCHEMA_ID	The numeric identifier of the currently active default schema, such as 80.
CURRENT_USER	The name of the database user whose privileges are currently active, such as SH.
CURRENT_USERID	The numeric identifier of the database user whose privileges are currently active, such as 80.
DATABASE_ROLE	Data Guard role of the database: PRIMARY, PHYSICAL STANDBY, LOGICAL STANDBY, or SNAPSHOT STANDBY.
DB_DOMAIN	The network domain of the database as specified by the DB_DOMAIN initialization parameter, such as us.example.com.
DB_NAME	The name of the database as specified by the DB_NAME initialization parameter.
DB_UNIQUE_NAME	The unique name of the database within the domain as specified by the DB_UNIQUE_NAME initialization parameter.
ENTERPRISE_IDENTITY	The enterprise-wide identity of the user, or NULL for local users, SYSDBA, and SYSOPER.

Table 3-3 (Cont.) USERENV Attributes

USERENV Attribute	Description
FG_JOB_ID	Job identifier of the current session if a client foreground process opened it; otherwise, NULL.
GLOBAL_CONTEXT_MEMORY	The number used in the System Global Area by the globally accessed context.
GLOBAL_UID	The global user identification from Oracle Internet Directory for Enterprise User Security logins; otherwise, NULL.
HOST	The name of the client host computer.
IDENTIFICATION_TYPE	The way the user schema was created in the database: LOCAL, EXTERNAL, GLOBAL SHARED, or GLOBAL PRIVATE.
INSTANCE	The identification number of the current instance, such as 1.
INSTANCE_NAME	The name of the database instance.
IP_ADDRESS	The IP address of the client, such as 10.255.255.255.
ISDBA	TRUE if the user was authenticated with DBA privileges; otherwise, FALSE.
LANG	A short name for the session language, such as US for AMERICAN.
LANGUAGE	The language, territory, and database character set in the form *language_territory.characterset*, such as AMERICA_AMERICAN.WE8DEC.
MODULE	The application name set through the DBMS_APPLICATION_INFO package or OCI, such as JDBC Thin Client or SQL Developer.
NETWORK_PROTOCOL	The network protocol being used for communication, such as TCP.
NLS_CALENDAR	The session calendar, such as GREGORIAN.
NLS_CURRENCY	The session currency mark, such as $.
NLS_DATE_FORMAT	The session date format, such as DD-MON-RR.
NLS_DATE_LANGUAGE	The session date language, such as AMERICAN.
NLS_SORT	BINARY or a linguistic sort basis, such as XSPANISH.
NLS_TERRITORY	The session territory, such as AMERICA.
OS_USER	The operating system user name of the client process that initiated the database session.
SERVER_HOST	The host name of the computer where the database instance is running.
SERVICE_NAME	The name of the service the session is connected to, such as SYS$USERS.
SESSION_USER	The database user name or schema name that identified the user at login, such as SH.
SESSIONID	The session identifier, such as 120456.
SID	The session number, such as 86.

Example

`SYS_CONTEXT('USERENV','NLS_DATE_FORMAT')` returns a value such as `DD-MON-RR`.

3.82 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.

3.83 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`.

3.84 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.

3.85 TANH

TANH returns the tangent of a hyperbolic angle.

Return Value

NUMBER

Syntax

```
TANH(n)
```

Arguments

n is a numeric expression for a hyperbolic angle.

Example

TANH(.5) returns the value 0.462117157.

3.86 TO_BINARY_DOUBLE

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

Return Value

BINARY_DOUBLE

Syntax

```
TO_BINARY_DOUBLE (expr [, fmt [, 'nlsparam' ] ])
```

Arguments

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.

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='',.''')
```

3.87 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='',.''')
```

3.88 TO_CHAR (character)

TO_CHAR(character) converts a text expression to the database character set.

Return Value

VARCHAR2

Syntax

TO_CHAR(exp)

Arguments

char is a text expression. If it contains characters that are not represented in the database character set, then the conversion results in a loss of data.

Examples

TO_CHAR('¡Una qué sorpresa!') returns the value ?Una qu? sorpresa! Two letters are lost in the conversion (¡ and é) because they are not in the database character set.

TO_CHAR('David Ortiz') returns the value David Ortiz in the database character set. No characters are lost in this conversion because all of them are in the database character set.

3.89 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`.

3.90 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`.

3.91 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`

3.92 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`.

3.93 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.

3.94 TO_NCHAR (datetime)

`TO_NCHAR(datetime)` converts a datetime or interval value to the national character set.

Return Value

`NVARCHAR2`

Syntax

```
TO_NCHAR({ datetime | interval }
         [, fmt [, 'nlsparam' ] ]
        )
```

Arguments

`datetime` is a datetime expression to be converted to text.

`interval` is an interval expression to be converted to text.

`fmt` is a datetime model format specifying the format of `char`. The default date format is determined implicitly by the `NLS_TERRITORY` initialization parameter or can be set explicitly by the `NLS_DATE_FORMAT` parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

`nlsparam` specifies the language in which month and day names and abbreviations are returned. This argument can have this form:

```
'NLS_DATE_LANGUAGE = language'
```

By default, the return value is in the session date language.

Examples

`TO_NCHAR(SYSDATE)` returns a value such as `11-APR-08`.

`TO_NCHAR(SYSDATE, 'Day: MONTH DD, YYYY')` returns a value such as `Friday : APRIL 11, 2008`.

`TO_NCHAR(SYSDATE, 'Day: MONTH DD, YYYY', 'NLS_DATE_LANGUAGE = Spanish')` returns a value such as `Viernes : ABRIL 11, 2008`.

3.95 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`.

3.96 TO_NUMBER

`TO_NUMBER` converts a text expression containing a number to a value of `NUMBER` data type.

Return Value

`NUMBER`

Syntax

```
TO_NUMBER(expr [, fmt [, 'nlsparam' ] ])
```

Arguments

`expr` is an expression to be converted to a number.

`fmt` is a text expression that identifies a number format model as described in the *Oracle Database SQL Language Reference*.

`nlsparam` specifies the characters used by these number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument has the format shown here:

```
'NLS_NUMERIC_CHARACTERS = ''dg''
 NLS_CURRENCY = ''text''
 NLS_ISO_CURRENCY = territory '
```

The `d` is the decimal character, and the `g` is the group separator. They must be different single-byte characters. Within the quoted string, use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit this argument or any of the NLS parameters, then this function uses the default values for your session to interpret `expr`.

Examples

`TO_NUMBER('$1,234,567.89', 'L999G999G999D99')` returns the value `1234567.89`.

TO_NUMBER('EUR1,234,567.89', 'C999G999G999D99', 'NLS_ISO_CURRENCY=SPAIN') also returns the value 1234567.89.

3.97 TO_TIMESTAMP

TO_TIMESTAMP converts a text expression to a value of TIMESTAMP.

Return Value

TIMESTAMP

Syntax

TO_TIMESTAMP(char [, fmt ['nlsparam']])

Arguments

char is a text expression to be converted.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the NLS_TERRITORY initialization parameter or can be set explicitly by the NLS_DATE_FORMAT parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

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

```
'NLS_DATE_LANGUAGE = language'
```

By default, char is in the session date language.

Examples

TO_TIMESTAMP('10-SEP-0614:10:10.123000','DD-MON-RRHH24:MI:SS.FF') returns the value 10-SEP-06 02.10.10.123000 PM.

TO_TIMESTAMP('10-AGOSTO-0714:10:10', 'DD-MON-RRHH24:MI:SS.FF', 'NLS_DATE_LANGUAGE=SPANISH') returns the value 10-AUG-07 02.10.10.000000 PM.

3.98 TO_TIMESTAMP_TZ

TO_TIMESTAMP_TZ converts a text expression to a value of TIMESTAMPWITHTIMEZONE data type.

Return Value

TIMESTAMP WITH TIME ZONE

Syntax

TO_TIMESTAMP_TZ(char [, fmt ['nlsparam']])

Arguments

char is a text expression to be converted.

fmt is a datetime model format specifying the format of char. The default date format is determined implicitly by the NLS_TERRITORY initialization parameter or can be set explicitly by the `NLS_DATE_FORMAT` parameter. For data type formats, refer to the *Oracle Database SQL Language Reference*.

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

```
'NLS_DATE_LANGUAGE = language'
```

By default, `char` is in the session date language.

Examples

`TO_TIMESTAMP_TZ('2006-03-26 7:33:00 -4:00', 'YYYY-MM-DD HH:MI:SS TZH:TZM')` returns the value `26-MAR-06 07.33.00.000000 AM -04:00`.

`TO_TIMESTAMP_TZ('2006-AGOSTO-13 7:33:00 -4:00', 'YYYY-MONTH-DD HH:MI:SS TZH:TZM', 'NLS_DATE_LANGUAGE=SPANISH')` returns the value `13-AUG-06 07.33.00.000000 AM -04:00`.

3.99 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`.

3.100 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`.

3.101 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.

3.102 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!!`.

3.103 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.

3.104 TZ_OFFSET

TZ_OFFSET returns the time zone offset from Coordinated Universal Time (UTC).

Return Value

VARCHAR2

Syntax

```
TZ_OFFSET({ 'time_zone_name' | '{ + | - } hh:mi'
           | SESSIONTIMEZONE | DBTMEZONE
           }
          )
```

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.

3.105 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`.

3.106 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`.

3.107 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`.

3.108 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`.

3.109 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:

Channel	VSIZE
Catalog	7
Direct Sales	12
Internet	8

3.110 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.

A Reserved Words

This appendix lists the reserved words for the OLAP expression syntax.

A.1 Reserved Words

The following are reserved words for the OLAP Expression Syntax. You should not use them or Oracle Database reserved words as object names. However, if you do, be sure to enclose them in double quotes in the expression syntax.

Refer to the *Oracle Database SQL Language Reference* for a list of Oracle Database reserved words.

AGGREGATE
AGGREGATES
ALL
ALLOW
ANALYZE
ANCESTOR
AND
ANY
AS
ASC
AT
AVG
BETWEEN
BINARY_DOUBLE
BINARY_FLOAT
BLOB
BRANCH
BUILD
BY
BYTE
CASE
CAST
CHAR
CHILD
CLEAR
CLOB
COMMIT
COMPILE
CONSIDER
COUNT
DATATYPE
DATE

DATE_MEASURE
DAY
DECIMAL
DELETE
DESC
DESCENDANT
DIMENSION
DISALLOW
DIVISION
DML
ELSE
END
ESCAPE
EXECUTE
FIRST
FLOAT
FOR
FROM
HIERARCHIES
HIERARCHY
HOUR
IGNORE
IN
INFINITE
INSERT
INTEGER
INTERVAL
INTO
IS
LAST
LEAF_DESCENDANT
LEAVES
LEVEL
LEVELS
LIKE
LIKEC
LIKE2
LIKE4
LOAD
LOCAL
LOG_SPEC
LONG
MAINTAIN
MAX
MEASURE
MEASURES
MEMBER
MEMBERS
MERGE

MLSLABEL
MIN
MINUTE
MODEL
MONTH
NAN
NCHAR
NCLOB
NO
NONE
NOT
NULL
NULLS
NUMBER
NVARCHAR2
OF
OLAP
OLAP_DML_EXPRESSION
ON
ONLY
OPERATOR
OR
ORDER
OVER
OVERFLOW
PARALLEL
PARENT
PLSQL
PRUNE
RAW
RELATIVE
ROOT_ANCESTOR
ROWID
SCN
SECOND
SELF
SERIAL
SET
SOLVE
SOME
SORT
SPEC
SUM
SYNCH
TEXT_MEASURE
THEN
TIME
TIMESTAMP
TO

UNBRANCH
UPDATE
USING
VALIDATE
VALUES
VARCHAR2
WHEN
WHERE
WITHIN
WITH
YEAR
ZERO
ZONE

A.2 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>
<tr><th>Symbol</th><th>Description</th></tr>
<tr><td>,</td><td>Comma</td></tr>
<tr><td>||</td><td>Concatenate</td></tr>
<tr><td>/</td><td>Divide</td></tr>
<tr><td>=</td><td>Equals</td></tr>
<tr><td>></td><td>Greater than</td></tr>
<tr><td>>=</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><</td><td>Less than</td></tr>
<tr><td><=</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><></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>
</table>

Index

O

P

Q

R

S

T

U

V

W